HOUDINI FOUNDATIONS

PROCEDURAL ASSETS FOR UNREAL

To create game assets using Houdini's node-based workflow, it is important to start learning how to think and work procedurally. In this lesson, you will learn how to create game assets using procedural nodes and networks then deploy them directly into Unreal using the Houdini Engine.

Along the way, you will get to use different aspects of Houdini's user interface. You will learn how the different UI elements work together to support you as you build your game assets.

This lesson was completed using **Unreal Engine 5**. Note that while the lesson focuses on Unreal, you can import the same assets into Unity using the Houdini Engine.

LESSON GOAL

To create assets that can be imported into Unreal as game art.

WHAT YOU WILL LEARN

- How to work with **Nodes and Networks** to control the flow of data
- How **Houdini Digital Assets** can be used to package and share your solution with others
- How to load Houdini Digital Assets into the **Unreal Editor**
- How to **instance** objects to points and control the orientation and scale of the objects using attributes
- How to create a game asset with **Collision Geometry** for use in Unreal
- How to export a **Rigid Body Simulation** as an FBX file for Unreal.

LESSON COMPATIBILITY

Written for the features in Houdini 19.5+

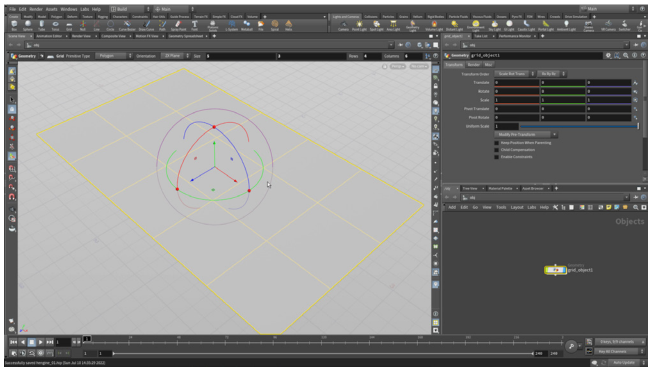
The steps in this lesson can be completed using the following Houdini Products:

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

Document Version 3.0 | July 2022

PART ONE:
Create a Simple Building

Learn how to build a simple building using a procedural network of nodes. Some of the nodes will be created by interacting in the Scene View and others in the Network view. You will use channel references to connect parts of one node with other nodes in the system. This creates a procedural solution that you will wrap up into a Houdini Digital Asset.



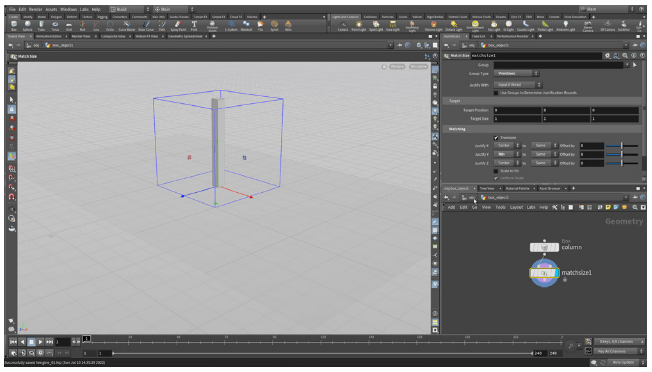
01

Select **File > New Project**. Call it *hengine_lesson* and press **Accept**. Select **File > Save As...** Set the file name to *hengine_01.hip* and click **Accept** to save.

In the viewport, **press c** and choose **Create > Geometry > Grid**. **Press Enter** to place it the grid at the origin. In the **Operation Controls** bar at the top of the Scene View, set:

- **Size** to **5, 3**
- **Rows** to **4**
- **Columns** to **6**

Press v and choose **Shading > Smooth Wire Shaded.**

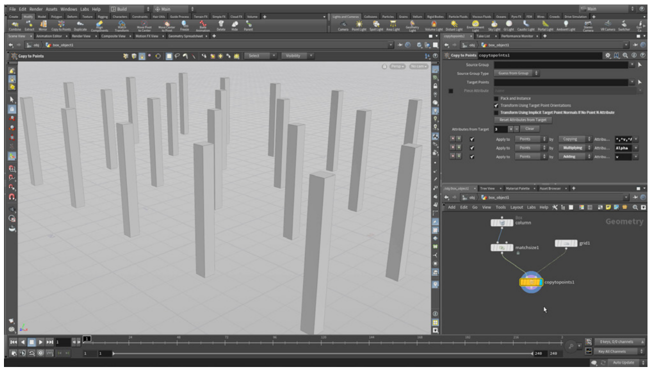


02

Press c and choose **Create > Geometry > Box**. Press **Enter** to place it at the origin. **Double click** on the *box_object* in the Network view. This puts you inside the object at the geometry level. Set the following:

- **Size** to **0.1, 1, 0.1**

Rename the *box* node to *column*. In the Scene view press tab and start typing **Match Size** then select **Match Size** . **Press n** to select the box and then press **Enter**. This adds a new node. In the Parameter pane, under **Matching**, set **Justify Y** to **Min**. This raises the box up so that it sits on the ground.

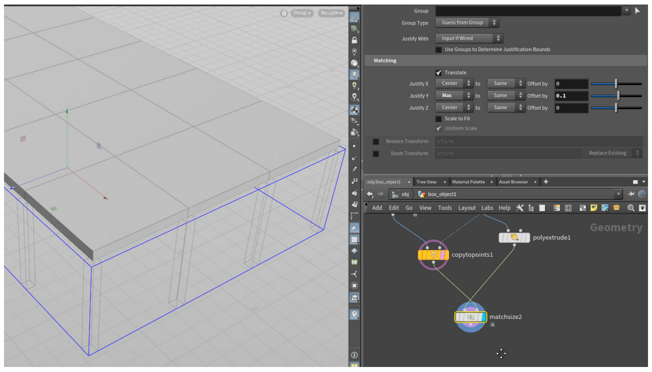


03

Press u to go back up to the object level or click on **obj** on one of the Network Path bars. Click on the **Select** tool and click in empty space to deselect all the objects.

On the **Modify** shelf, click on **Copy to Points.** Select the column as the *geometry to copy* and press **Enter** then click on the grid as the *geometry on whose points to copy* and press **Enter.**

In the Parameter pane, set **Transform Using Implicit Target...** to **Off**. Now the columns will be pointing up. Turn on **Pack and Instance** to ensure instanced geometry in Unreal. **Press 4** to go to primitive selection mode. This will hide all the corner points.

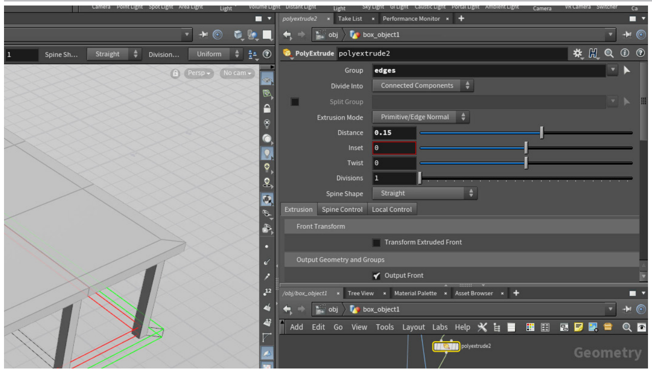


04

Press **tab > PolyExtrude** in the Network View. Place it to the side. Wire the grid node into the *polyextrude* node and set its **Display flag**. Set **Distance** to **0.1** and under **Output Geometry and Groups** turn on **Output Back**. Set the **Template Flag** on the **copytopoints** node.

Add a **Match Size** node to the network and wire *polyextrude* into the first input and *copytopoints* into the second. Set **Justify Y to Max to Same**, **Offset** by **0.1.** Now the extruded shape will sit on top of the columns.

Add a **Merge** node and wire *copytopoints* and *matchsize* into it.


PolyExtrude polyextrude2

Group: edges
Divide Into: Connected Components
Split Group
Extrusion Mode: Primitive/Edge Normal
Distance: 0.15
Inset: 0
Twist: 0
Divisions: 1
Spine Shape: Straight

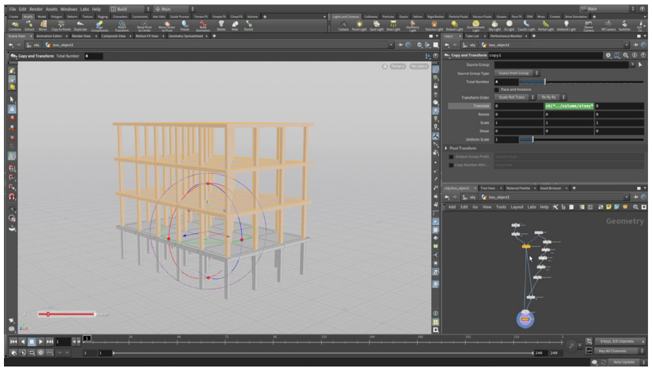
Front Transform
Transform Extruded Front
Output Geometry and Groups
Output Front ✓

05

In the Network view, press **tab > group**. Wire it between *polyextrude* and *matchsize*. Change **Group Name** to *edges*.

Alt drag to create a another *group* node and set its **Display flag.** Keep **Group Name** set to *edges* and then set **Initial Merge** to **Subtract from Existing**. Turn off **Enable** under **Base Group** and turn on **Enable** under **Keep by Normals**. Set **Direction** to **0, 1, 0** and **Spread Angle** to **0**. **Alt drag** on this node to make a copy and wire it into the chain. Change **Direction** to **0, -1, 0**. Now only the sides of the box are in the *edges* group.

Add a **PolyExtrude** node. Set **Group** to *edges* and **Direction** to **0.15**.

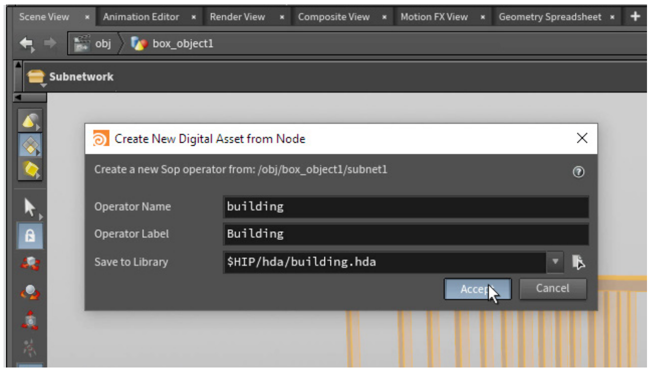


06

Set the *merge* node's **Display Flag** to see the first floor of the building complete. Turn off the **Template** Flag on the *copytopoints* node.

In the Scene view, **press n** to select all and then press **tab > Copy and Transform**. Select the *column* node and **RMB-click** on **Center Y** and choose **Copy Parameter**. Go to the *copy* node and **RMB-click** on **Translate Y** and choose **Paste Relative References**. Add a **+ 0.1** to the expression.

Now increase **Total Number** to 4 to add three more floors.


Create New Digital Asset from Node

Create a new Sop operator from: /obj/box_object1/subnet1

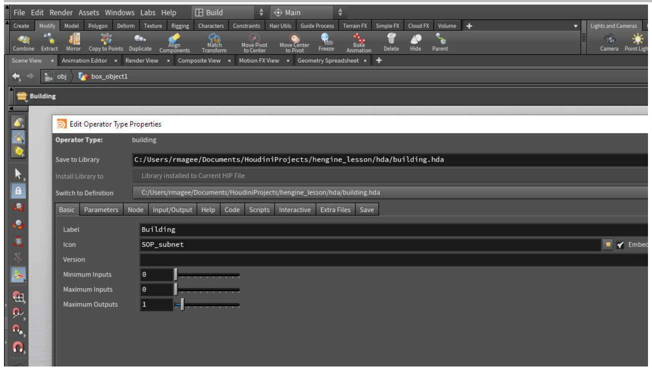
Operator Name: building
Operator Label: Building
Save to Library: $HIP/hda/building.hda

Accept | Cancel

07

Select all the nodes in the Network editor. From the **Asset** menu, choose **New Digital Asset from Selection**. This will collapse the network into a subnetwork then use that subnet node to create the Digital Asset.

Set the **Operator Name** to *building* which will change the **Operator Label** to *Building*. Click on the button to the far right of **Save to Library.** In the *Locations* sidebar, click on $HIP/ and then double-click on the *hda* directory. Press **Accept** and then **Accept** again to save the asset to disk.


Edit Operator Type Properties

Operator Type: building
Save to Library: C:/Users/rmagee/Documents/HoudiniProjects/hengine_lesson/hda/building.hda
Install Library to: Library installed to Current HIP File
Switch to Definition: C:/Users/rmagee/Documents/HoudiniProjects/hengine_lesson/hda/building.hda

Basic | Parameters | Node | Input/Output | Help | Code | Scripts | Interactive | Extra Files | Save

Label: Building
Icon: SOP_subnet
Version
Minimum Inputs: 0
Maximum Inputs: 0
Maximum Outputs: 1

08

The **Edit Type Properties** window opens up. This panel is where you will build the user interface for your asset. You will revisit this window later. Click **Accept** to close this window.

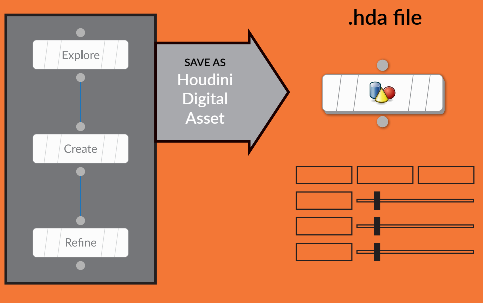
Rename the node in the network view to *building*.

The nodes you have used to build the asset will remain a part of the asset even after you save it. This will allow you to continue making changes even after you have started using it in your Unreal game levels.

WHAT IS AN HDA FILE?

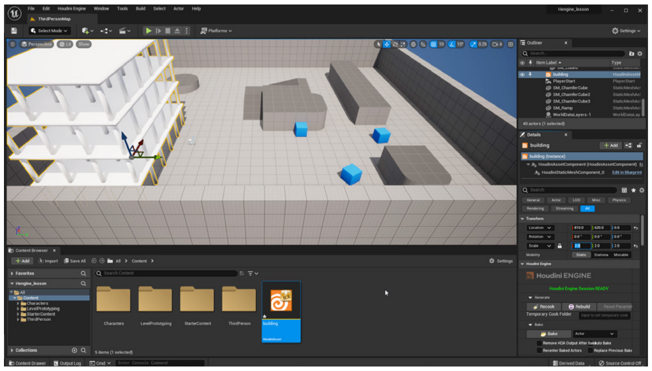
Houdini nodes and networks can be encapsulated into single nodes called **Houdini Digital Assets** which let you share your techniques with colleagues. These assets are saved to disk inside files known as **.hda** files.

Asset files created in older versions of Houdini may have a different extension **.otl** which means Operator Type Library - both these file types work the same way.



PART TWO:
Import the Asset into Unreal

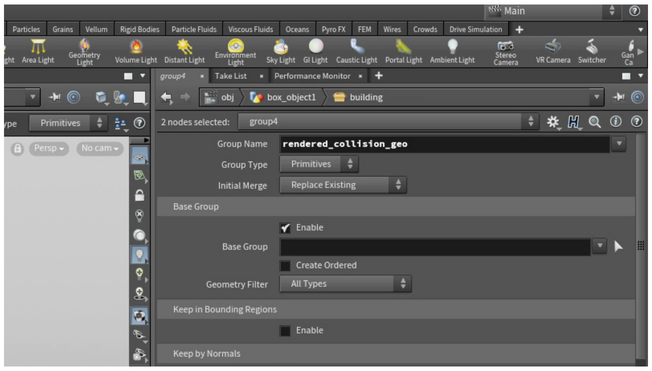
Now that you have a digital asset file on disk, you can import it into the Unreal Engine. This is possible because of the Houdini Engine plug-in which connects the two applications. Houdini Digital Assets that are loaded into Unreal are cooked using Houdini under the surface. **The Houdini Engine for Unreal plug-in must be installed to complete this lesson (See Sidefx.com/unreal for instructions).** *Please note that Houdini Apprentice does not work with Engine.*



01

IN UNREAL – Set up a **Third Person Template** with **Blueprint** support. Delete the *TextRenderActor*. Open the **Content Drawer** and click on the **Import** button**.** You may want to dock the **Content Drawer.** Navigate to your Houdini project and find the *building* asset. Click **Open**.

Set the **Scale** to **2, 2, 2** and move it into the corner. If you press the **Play** button you will see the building during gameplay.



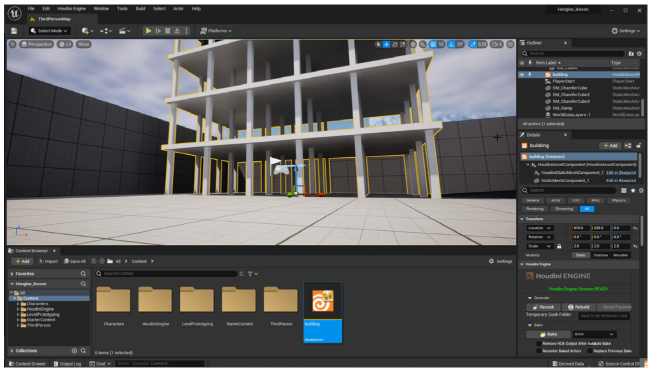
Group Name: rendered_collision_geo
Group Type: Primitives
Initial Merge: Replace Existing
Base Group
✓ Enable
Base Group:
Create Ordered
Geometry Filter: All Types
Keep in Bounding Regions
Enable
Keep by Normals

02

IN HOUDINI – To add normals and collisions to your geometry you need to add some nodes to the network.

Dive into the *building* network and between the *polyextrude* and *matchsize* nodes, add a **Normal** node. This will add normals for the geometry to display properly in Unreal. Under the *normal* node add a **Group** node and set the **Group Name** to *rendered_collision_geo*. This will turn your geometry into collision geometry.

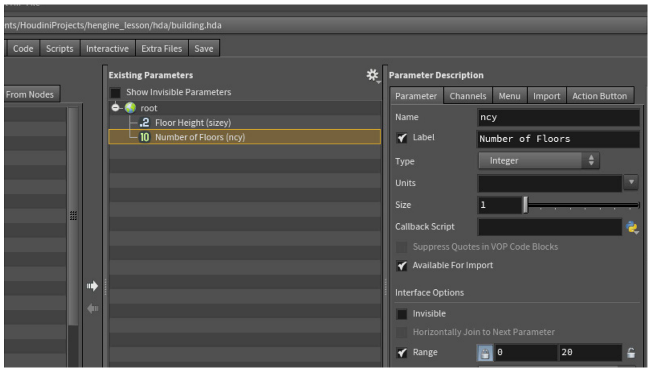
Copy and Paste these two nodes and insert the new nodes after the *column* node.



03

IN UNREAL – In the Generate section of the *building* asset's **Details** panel, press **Rebuild**. Press **Play** to explore the scene.

The normals are working properly and now you will collide with the columns if you try to run through them.

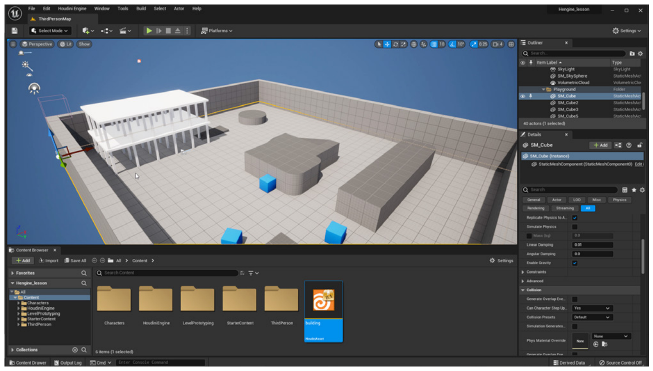


Existing Parameters
root
Floor Height (sizey)
Number of Floors (ncy)

Name: ncy
✓ Label: Number of Floors
Type: Integer
Units:
Size: 1
Callback Script:
Suppress Quotes in VOP Code Blocks
✓ Available For Import
Interface Options
Invisible
Horizontally Join to Next Parameter
✓ Range: 0 20

04

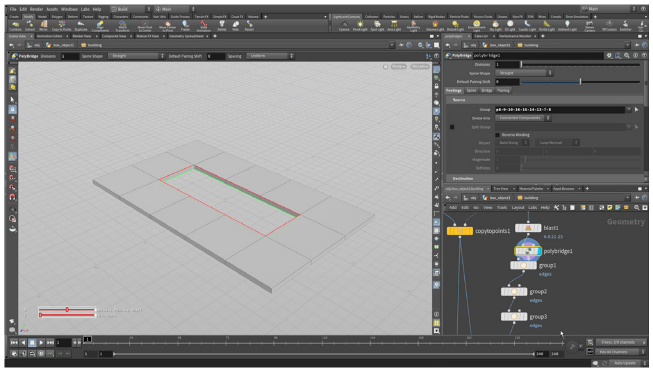
IN HOUDINI – Select **Assets > Edit Asset Properties > Building**. In the properties window, click on the Parameters tab. Now select the *column* node and drag the **Size Y** parameter to the Parameter tab. **Rename** it **Floor Height**. Now from the *copy* node, drag the **Total Number** parameter over and rename it **Number of Floors**. Click **Accept**.



05 **IN UNREAL** – Press **Rebuild** in the building asset's **Details** panel. Scroll down to see that there are now parameters for **Floor Height** and **Number of Floors**.

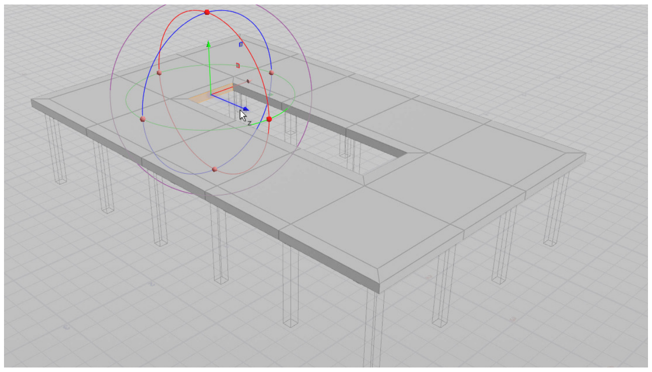
Set **Floor Height** to **1.5** and **Number of Floors** to **2**.

You can **Play** the level to walk around the asset and review the changes.



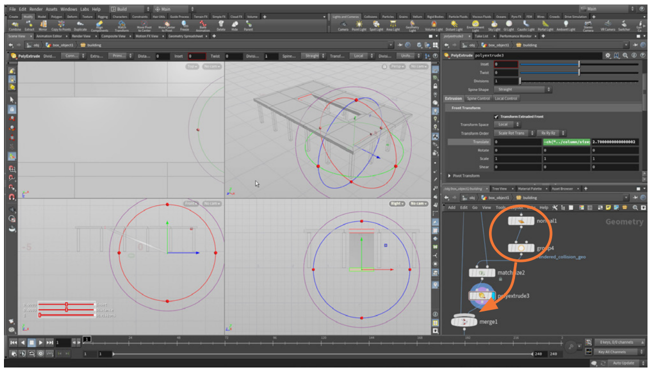
06 **IN HOUDINI** – Go back and set the **Display flag** on the floor slab's *polyextrude* node. **Press 4** to get primitive selection. Select the middle three primitives from the top then tumble around and shift-select the three primitives from the bottom of the slab. Press **Delete**. This adds a **Blast** node.

Press 3 to get edge selection. **Double click** on the bottom edge of the hole to select the whole loop. Go **tab > Polybridge.** Press **Enter**. Now **double click** on the top edge of the hole then press **Enter**. This will add geometry to the inside of the hole.



07 Set the **Display Flag** on the *matchsize* node. You will see the new hole with an overhang.

Press 4 to get primitive selection. Get the **Select** tool then select the end face of the opening. It may not appear selected but it is. Press **tab > PolyExtrude**. Under Extrusion, turn on **Transform Extruded Front.** Start pulling the face out and down.

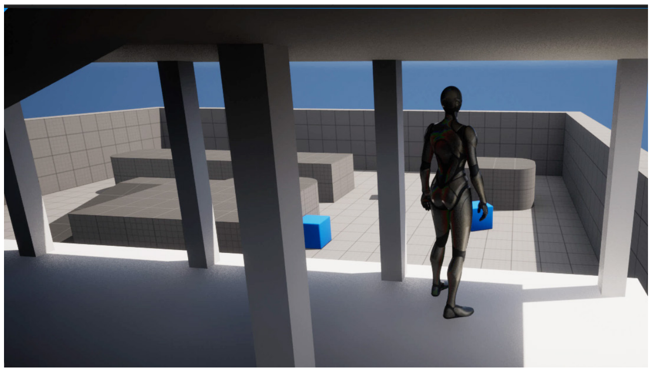


08 Copy the *column* node's **Size Y** channel and **Paste Relative Reference** to the new *polyextrude* node's **Translate Y**. Add a - sign to the expression then subtract 0.1. The expression should read: `-ch("../column/sizey")-0.1`

Set **Translate Z** to **2.7**.

Move the *normal* and *group* nodes to after this *polyextrude* node. Otherwise, the new extrusion won't be collision geometry and the ramp won't work properly.

Set the **Display Flag** on the *output* node. Choose **Assets > Save Asset > Building** to save the changes to the HDA file on disk.

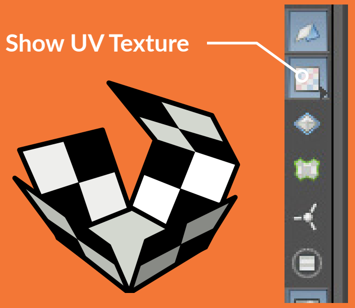


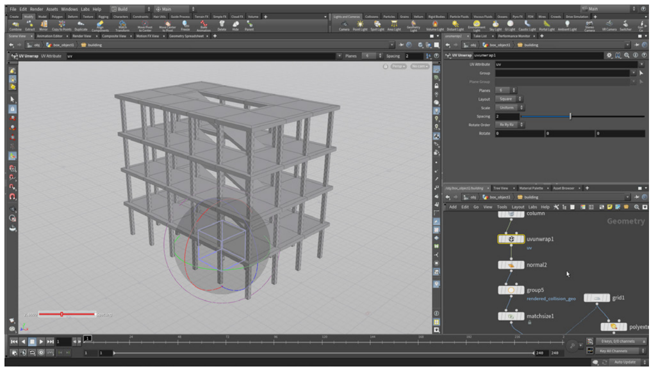
09 **IN UNREAL** – Press **Rebuild** then set the **Floor Height** to **1** and the **Number of Floors** to **6**. Press **Play** and walk up ramps and around the building.

UVS

Houdini has a number of nodes at the geometry level for setting up and managing UVs. In this lesson, you will use **UV Unwrap** and **UV Transform** to add UVs to the building. These work well with the simple shapes used to model the building. For more complex shapes you will use tools such as **UV Flatten** and **UV Layout**.

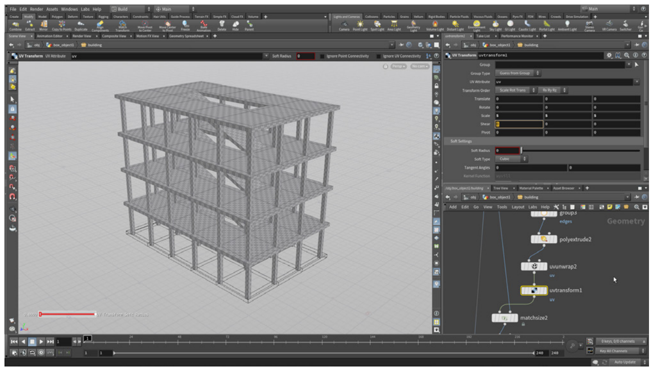
You can see a UV Grid on your geometry once UVs are in place. To hide the grid, you can click on the **Show UV Texture** button on the **Display Options** bar to toggle it on and off.





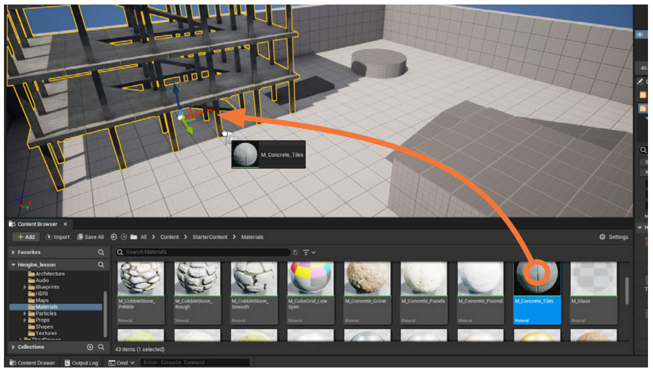
10 **IN HOUDINI** – You need to add some UVs to the geometry for use in the game editor. Go to the part of the network with the *column* node. Press tab> **UV Unwrap** and place the node between the *column* node and the *matchsize* node.

Go to the part of the network with the *second polyextrude* node. Press **tab > UV Unwrap** and place the node between the *polyextrude* node and the *matchsize* node.



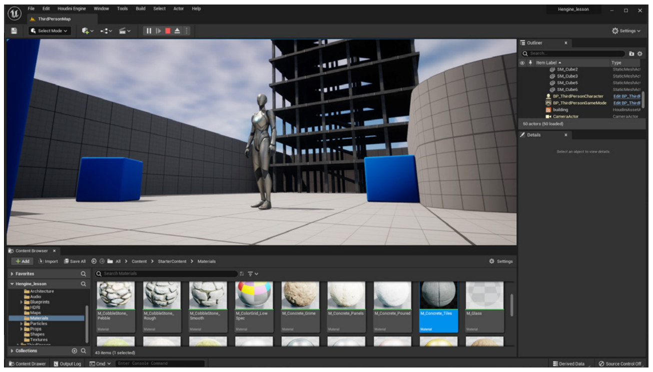
11 You will see a UV grid on the columns. They are a different scale compared to the columns. Press **tab > UV Transform** and place the node between the *uvunwrap* node and the *matchsize* node. Set the **Scale** values to **5, 5, 5**. Now the UVs appear more alike.

Choose **Assets > Save Asset > Building** to save the changes to the HDA file on disk.



12 **IN UNREAL** – Press **Rebuild.** You won't see the UVs until you add a Material.

In the **Content Browser,** navigate up to **Content > Starter Content > Materials**. Make sure the *Building* asset is selected then add a Material such as *M_Concrete_Tiles* to the **building geometry** in the scene view. You may need to drag it to both the columns and the slabs.

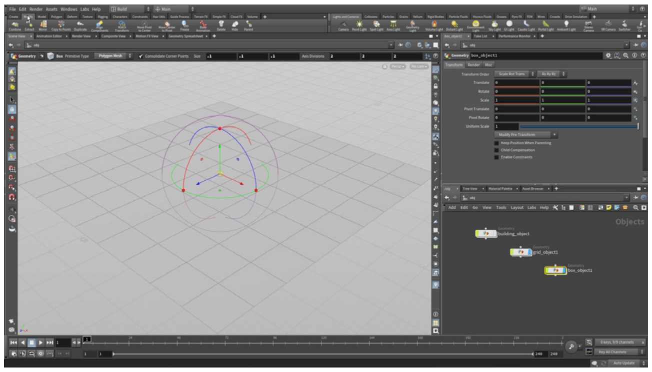


13 Press **Play** to walk through the level and explore the textured building.

This asset could be used more than once on a level to create different buildings with different floor heights and building heights. You could also promote more parameters to the asset to add more control. All the assets on your level that come from this digital asset can all be updated at once when a change is made to the asset's network or its parameter interface.

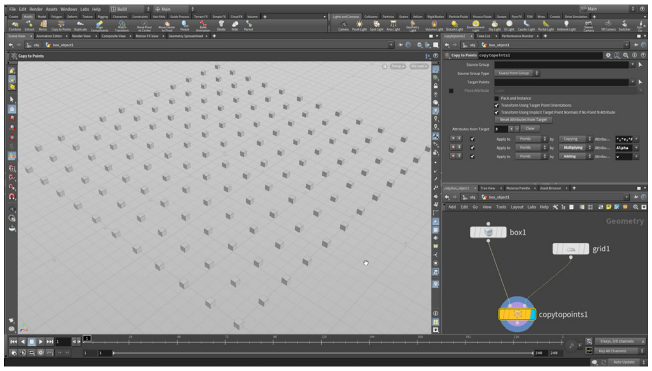
PART THREE: Copy to Points

In this part of the lesson, you are going to copy some boxes to points of another grid. You will then randomize those points to achieve a more organic look and add random attributes to rotate and scale the boxes to create more variety in the distribution of the shapes. This creates another procedural system that you will wrap up into a Houdini Digital Asset.



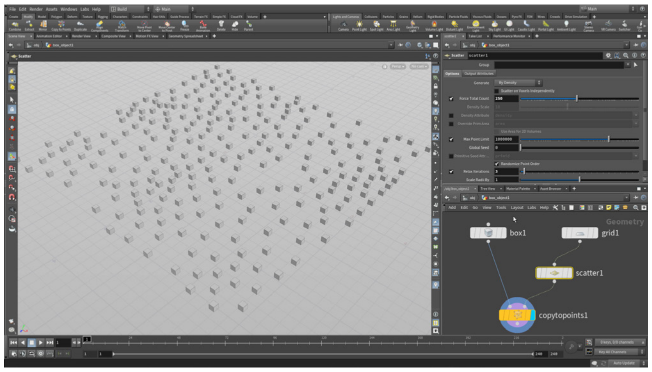
01 **IN HOUDINI** – Go to the Object level and hide the building. In the viewport, press the **c** key to bring up a radial menu and select **Create > Geometry > Grid**. Release and then press **Enter** to place at the origin. You will start by using the points on this grid surface to instance geometry.

Use the same radial menu to **Create > Geometry > Box** and again press **Enter** to place at the origin. In the **Operation Control** bar at the top of the viewport, set the **Size** to **0.1 0.1 0.1**. This is the geometry that you will be copying to the points on the grid.



02 With the *box* still selected, go to the **Modify** shelf and click on the **Copy to Points** tool. Select the *grid* and press **Enter**. Now the boxes have been copied to all of the grid points.

You can now adjust the look of the system by editing parameters. Select the *grid* node and change the **Size** to **6, 6** and the **Rows** and **Columns** to **12**. The grid gets bigger and has more points but the boxes aren't being copied to all the points. Click on the *copytopoints* node where you can see that **Target Points** are set to **0-99** to match the points on the original grid. Remove the **0-99** to copy boxes to the whole grid.



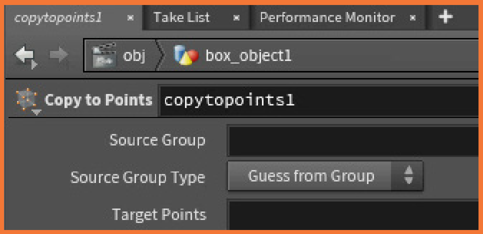
03 In the Network Editor, press **tab > Scatter**. Place the *scatter* node in the Network editor between the *grid* and the *copytopoints* nodes. It will wire itself into the network automatically and the boxes will now be copied to these new points.

Set **Force Total Count** to **250**. Play with the **Relax Iterations** to adjust how the points are being laid out. The scatter node gives you a more organic layout compared to the original grid points.

SOURCE GROUP

When you model in the Scene view, your selected geometry will get placed into the **Source Group** field as either numbered points or primitives depending on the tool. If the field is empty then the tool will work on **ALL** of the points or primitives.

When you use tools at the geometry level this field will be left empty when you choose **Select All [n]** before using the tools. Because you set up ***copytopoints*** at the object level, it filled in the points as 0-99.

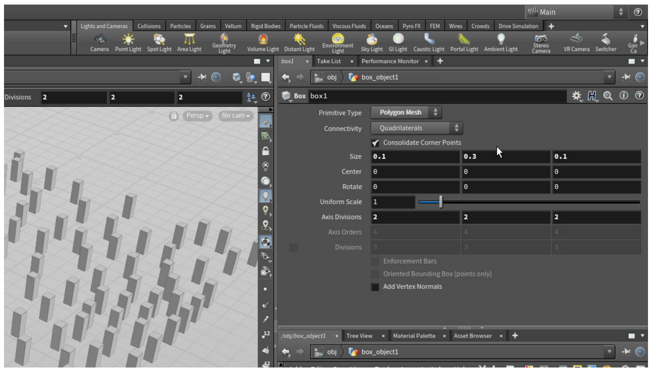


Copy to Points copytopoints1

Source Group

Source Group Type Guess from Group

Target Points



Box box1

Primitive Type	Polygon Mesh		
Connectivity	Quadrilaterals		
✓ Consolidate Corner Points			
Size	0.1	0.3	0.1
Center	0	0	0
Rotate	0	0	0
Uniform Scale	1		
Axis Divisions	2	2	2
Axis Orders	4	4	4
Divisions	3	3	3

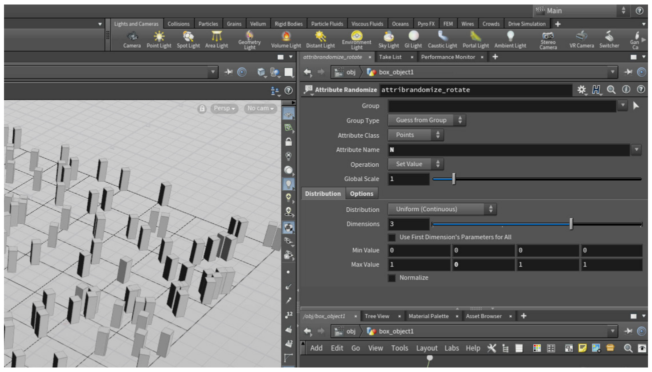
Enforcement Bars

Oriented Bounding Box (points only)

Add Vertex Normals

04 In the Network view press tab and start typing **Match Size** then select **Match Size** . Place this node between the *box* node and the *copytopoints* node. In the Parameter pane, under **Matching**, set **Justify Y** to **Min.** .

On the *box* node, set **Size Y** to **0.3** to test the expression. You can see that all of the boxes are sitting on the ground.



Attribute Randomize attribrandomize_rotate

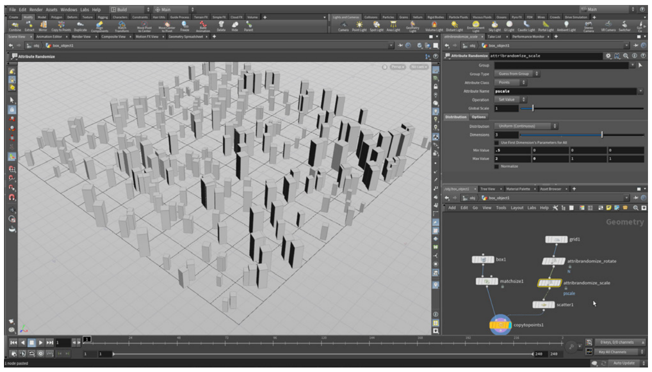
Group				
Group Type	Guess from Group			
Attribute Class	Points			
Attribute Name	N			
Operation	Set Value			
Global Scale	1			
Distribution	Uniform (Continuous)			
Dimensions	3			
Min Value	0	0	0	0
Max Value	1	0	1	1

Use First Dimension's Parameters for All

Normalize

05 In the Network Editor, press tab and begin to type **rand...** then choose the **Attribute Randomize** tool. Place the *attributerandomize* node between the *grid* and the *scatter* nodes. At first, you will see random colors on your boxes because the default attribute is color (Cd).

Set the **Attribute Name** to **N**. This changes the attribute to the normal direction and all the boxes are now pointed in different directions. Set **Max Value Y** to **0** which will limit the randomness to the X and Z directions. Rename it *attrrandomize_rotate*.

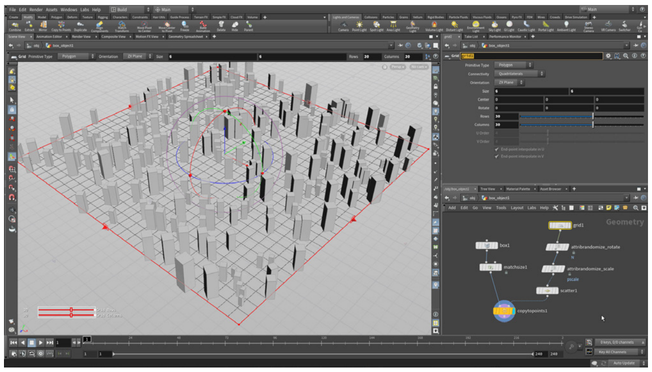


06 Press the **Alt** key and drag on the *attrrandomize_rotate* node to make a copy of it. Drop it between the *attrrandomize_rotate* and the *scatter* nodes. Rename it *attrrandomize_scale*.

Set the following:

- **Attribute Name** to *pscale*.
- **Min Value** to **0.5**
- **Max Value** to **2**.

This gives you a nice variety of sizes for the boxes on the grid.



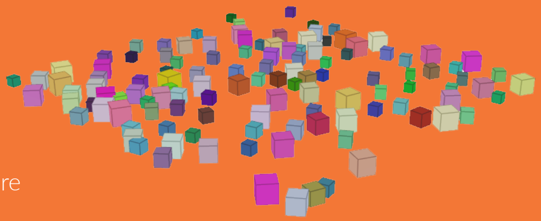
07 Select the *grid* node and increase the **Rows** and **Columns** to **30** – this creates more points on the grid which creates a wider variety of values on the scatter points.

Save your work.

HOW ATTRIBUTES WORK

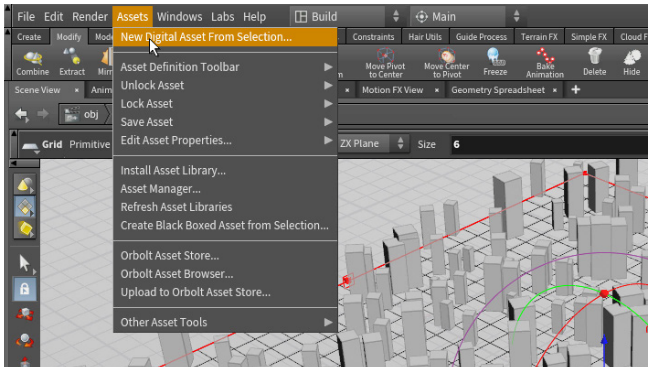
Attributes that are assigned to geometry are passed down the network chain to different nodes. In the case of this network, the attributes being assigned to the grid geometry are passed on to the scattered points which in turn affects the copied geometry. This is an important way to control the flow of data in Houdini.

The attributes are initially assigned to the points on the grid therefore more points lead to more randomization in the attribute values.



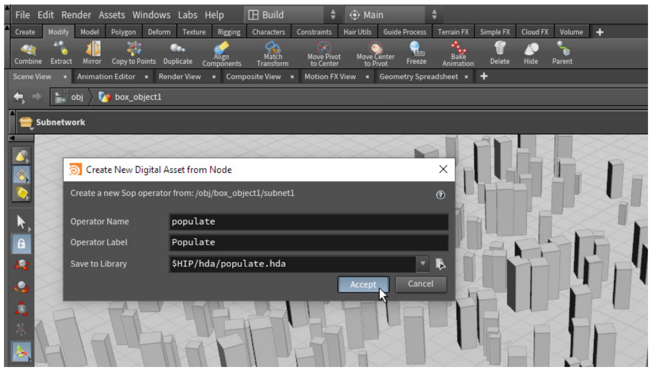
PART FOUR:
Create another Houdini Digital Asset

In this part of the lesson, you are going to create a digital asset and test the system in Unreal. Just like the building this means wrapping up the network and saving the results to disk as an HDA file. You will promote some parameters to allow you to control the grid size, number of points and the relaxation. The parameters will then be available to you in Unreal.



01 Select all the nodes in the Network editor. From the **Asset** menu, choose **New Digital Asset from Selection**. This will collapse the network into a subnetwork then use that subnet node to create the Digital Asset.

The nodes you used to build the asset will remain a part of the asset even after you save it. This will allow you to continue making changes even after you have started using it in your game levels.



Create a new Sop operator from: /obj/box_object1/subnet1

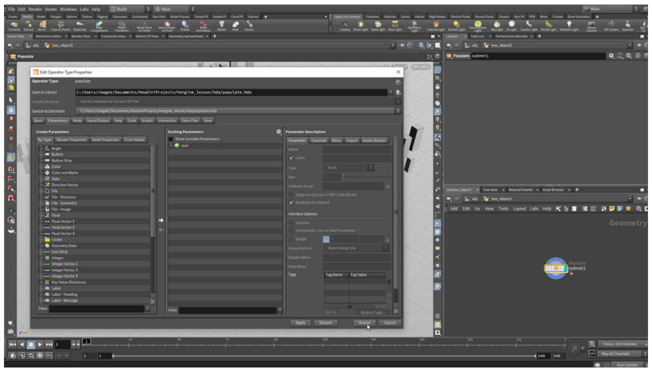
Operator Name: populate

Operator Label: Populate

Save to Library: $HIP/hda/populate.hda

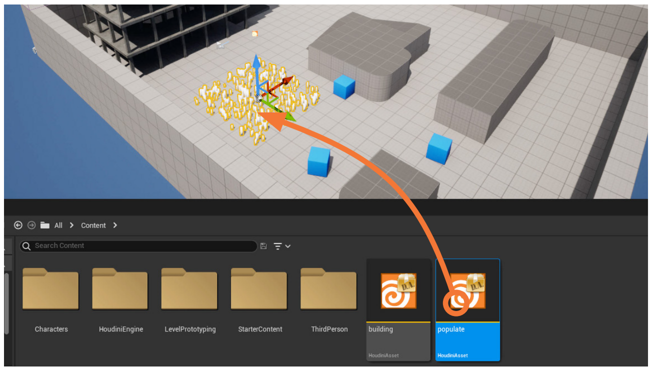
02 Set the **Operator Name** to *populate* which will change the **Operator Label** to *Populate*. Click on the button to the far right of **Save to Library.** In the *Locations* sidebar, click on $HIP and then double-click on the *hda* directory. Press **Accept** and then **Accept** again to save the asset to disk.

This creates a new Houdini Digital Asset file (.hda) on disk that is being referenced by this scene. It can also be referenced into other Houdini scenes or into other applications such as Unreal using the Houdini Engine.



03 The **Edit Type Properties** window opens up. This panel is where you will build the user interface for your asset. You will revisit this window later. Click **Accept** to close this window.

In order for you to maintain the procedural nature of the Houdini Digital Asset, you can build a high level interface that can be used to access the nodes inside the network. You will add to the interface for this asset later on in the lesson.



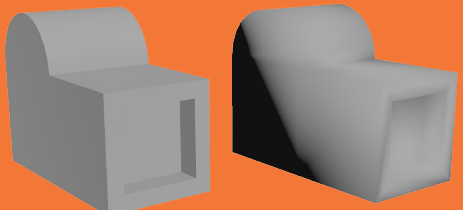
04 **IN UNREAL** – In the Content Browser, go back to the *Content* directory. Click **Import to** and find the *populate.hda* asset file in the current project directory. Drag the asset from the Content Browser to the 3D workspace.

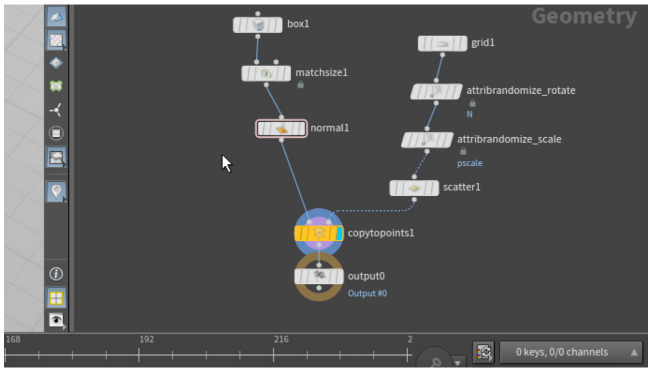
Press **Play** and walk around the asset. It is there but there is nothing special about it. Press **Esc** to return to the asset UI.

VERTEX NORMALS

By default, Houdini objects like the box have point normals but don't have vertex normals. To make sure that you have proper vertex normals, you can add the **Normal** node which uses a cusp value to decide which edges should appear hard and which ones appear soft.

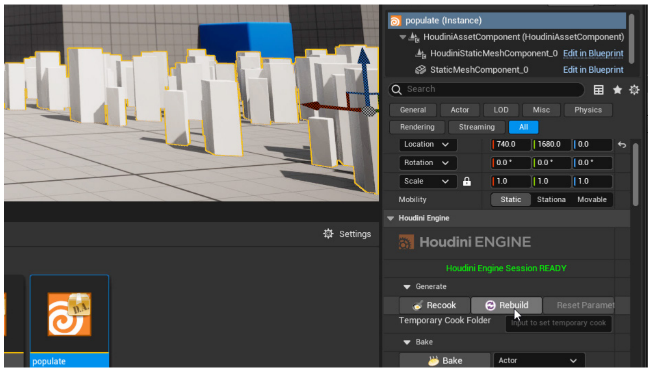
Game editors such as Unreal need vertex normals to display properly which can be set up easily in the existing network.


With Normals Without Normals



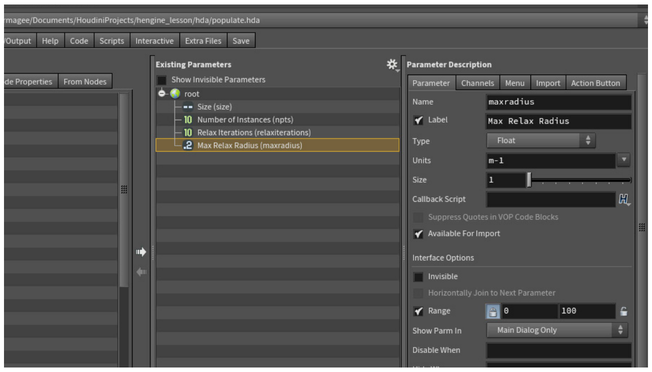
05 **IN HOUDINI** – One thing that you will notice about the asset in Unreal is that the boxes don't have sharp corners. To fix this you should go back to Houdini and in the Network editor, press **tab > "norm..."** then choose the **Normal** tool.

Add the **Normal** node right after the box node. From the **Asset** menu, choose **Save Asset > Populate**. This saves the change to the .hda file which makes the updated asset available to anyone who is has it loaded. In this case it means that you can update the asset definition inside Unreal and the correct normals will display.



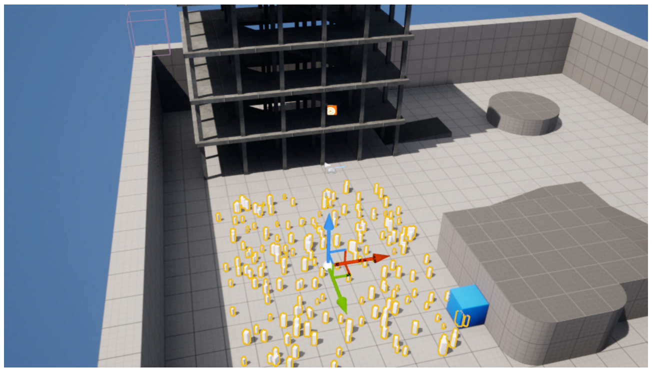
06 **IN UNREAL** – In the details panel, under Houdini Asset, open the **Cooking Actions** section. Click the **Rebuild Asset** button to accept the changes. This makes sure that the changes you made inside Houdini are properly updated in the Unreal scene.

The asset now has proper normals but you don't have any control over the procedural network. To create an interface that you can use in Houdini, you are now going to promote some parameters from inside the asset to the top level.



07 **IN HOUDINI** – Choose **Assets > Edit Asset Properties > Populate**. Click on the **Parameters** tab. In the Network editor, click on the *grid* node. From the Parameter pane, drag the **Size** parameter onto root in the **Existing parameters** list.

In the Network editor, click on the *scatter* node. From the Parameter pane, drag the **Force Total Count** parameter onto root. In the Parameter description, change the **Label** to **Number of Instances**. Drag the **Relax Iterations** and **Max Relax Radius** to add these parameters to the asset. Click **Accept** which saves these new parameters to the asset.

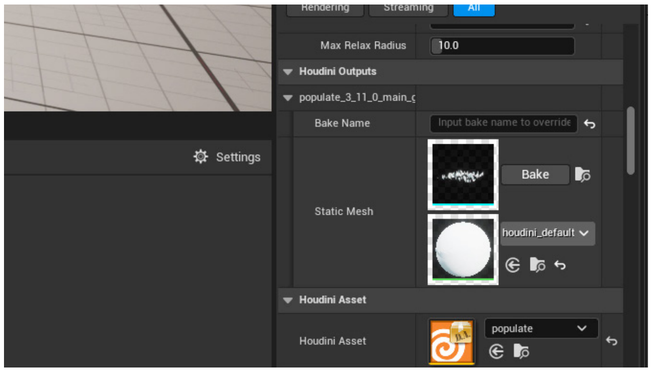


08 **IN UNREAL** – Click the **Rebuild Asset** button to accept the changes. The promoted parameters show up in the Parameter pane. Change the **Size** and **Number of Instances** to see how they affect the look of the resulting grid of boxes.

Now the procedural nature of the asset has been exposed and you can create unique versions of the asset in different levels. You can add more than one of these *populate* assets in this level and each of them can have unique settings while referencing the same asset in the .hda file. You can also use this asset in multiple levels being worked on by multiple artists.

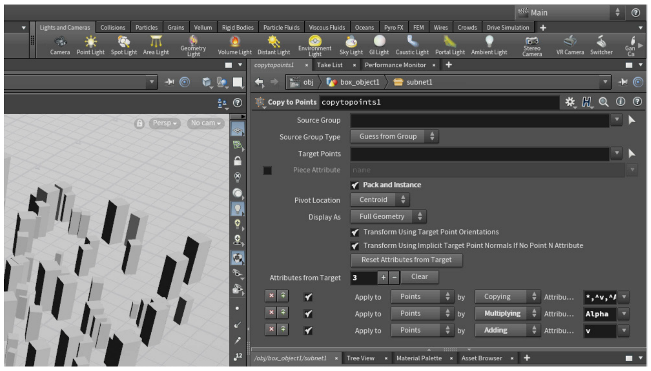
PART FIVE: Set up Instancing

When you set up instancing in Houdini properly, you can replace the default shape you have copied to the points with other Unreal props. You can add more than one prop that will be randomly distributed in place of your default shape. You are now going to make sure that instancing is in fact set up properly and then you will use this to add some other props into the system.



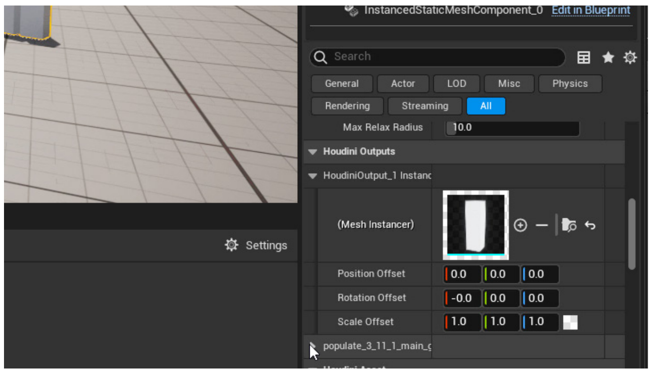
01 **IN UNREAL** – In the Details panel, go to the **Houdini Outputs** section. You can see that the whole collection of boxes is being imported as a single mesh. This means that you are not yet using instancing. Houdini is copying the boxes to the points then outputting the resulting geometry for Unreal.

This is not very efficient for gameplay. Therefore you need to set things up just a little differently to get the instancing that you need for this tool to work properly.



02 **IN HOUDINI** – Select the *copytopoints* node and turn on **Pack and Instance**. From the **Asset** menu, choose **Save Asset > Populate**.

By using packed primitives, the box geometry feeding into the *copytopoints* node is treated as a single primitive. This sets up instancing in Houdini and in turn triggers instancing in Unreal when this asset is loaded into the editor using the Houdini Engine plug-in.

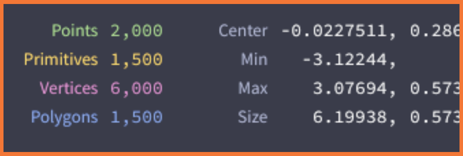


03 **IN UNREAL** – Click the **Rebuild Asset** button to accept the changes. Go to the **Houdini Outputs** section. You can see that only a single box is being imported and instanced to the points. It is then rotated and scaled based on the attributes you set up earlier.

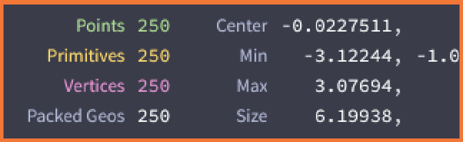
Now you can replace this default box with other geometry in your Unreal environment. This is the ideal way of populating a series of points because you have lots of flexibility in the editor to add different objects to the system.

PACKED PRIMITIVES IN HOUDINI

In Houdini, packed primitives provide an efficient way of managing instances for viewport display and rendering. The geometry feeding the copy node is packed into a single primitive and then treated like an instance. For the copied boxes shown in this lesson, this takes you from 1,500 primitives down to 250 once packing is in place. With the Houdini Engine plug-in for Unreal, the packed primitive is recognized as a Unreal instance which will allow for more efficient gameplay.



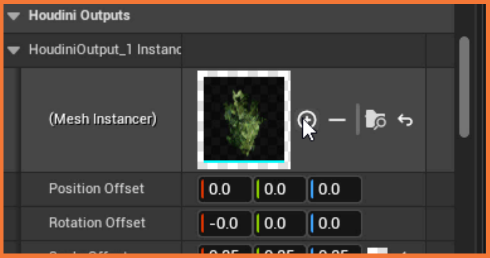
```
    Points  2,000      Center  -0.0227511,  0.286
Primitives  1,500         Min    -3.12244,
  Vertices  6,000         Max     3.07694,  0.573
  Polygons  1,500        Size     6.19938,  0.573
```



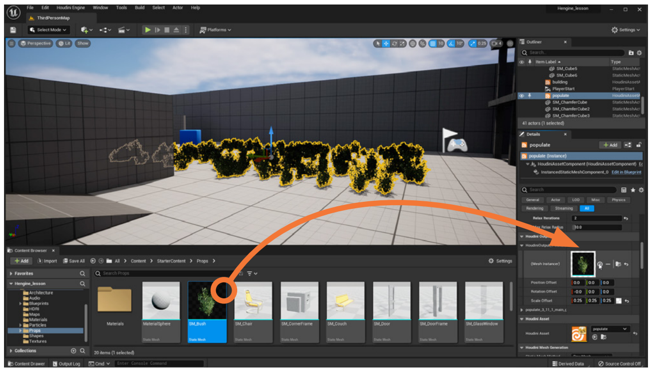
```
     Points  250      Center  -0.0227511,
 Primitives  250         Min    -3.12244,  -1.0
   Vertices  250         Max     3.07694,
Packed Geos  250        Size     6.19938,
```

INSTANCES IN UNREAL

When the Houdini Engine plug-in detects packed primitives, a Houdini Output Instancer is created in the Details tab that contains the input geometry and some parameters for Rotating and Scaling the instance. You can replace this with geometry from your Unreal content window and size it accordingly. The Plus sign lets you add more instanced inputs to create more variations in the same system.

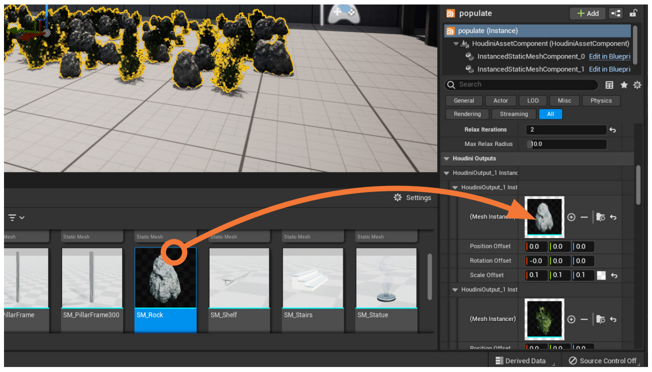


Position Offset	0.0	0.0	0.0
Rotation Offset	-0.0	0.0	0.0



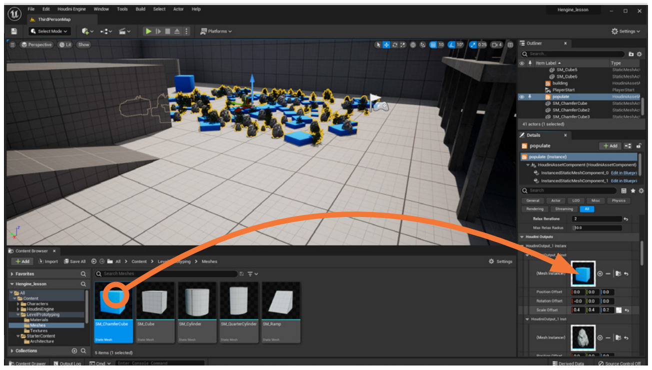
04 Go to the **Houdini Outputs** section and expand *HoudiniOutput1 instancer*. This is the box instance which you can replace with content from within Unreal.

In the Content Browser, open **Starter Content > Props**. Drag the *SM_Bush* prop over to the **Houdini Outputs**. Set the **Scale Offset** to **0.25** in all three axes. The geometry is instanced to the points in the populate asset and rotated and scaled just like the boxes.



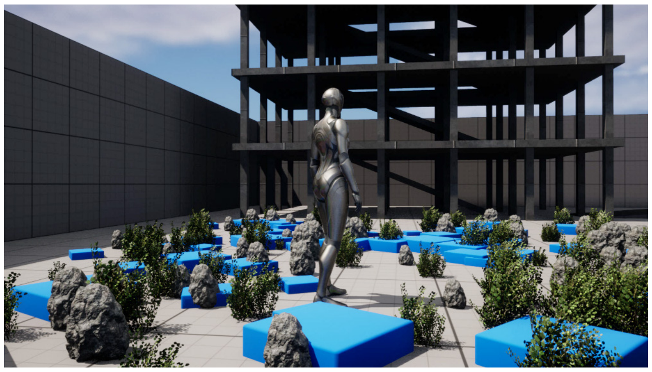
05 Now click the **Plus [+] sign** next to the instance object. This adds a second one. Drag the *SM_Rock* prop over to the new **Houdini Instanced Input**. Set the **Scale Offset** to **0.1** in all three axes..

In the Details panel, scroll to the **Houdini Engine** section and click the **Bake** button. In the Outliner, scroll to the *HoudiniAssetActor*. Click the eye icon to hide it so you can focus on the digital asset. Press **Play** and walk around the scene to see the instances in action.



06 Now click the **Plus sign** next to the bush object. This adds a third one. Navigate to the **Content > LevelPrototyping > Meshes** folder. Drag the *SM_ChamferCube* over to the new **Houdini Instanced Input**. Set the **Scale Offset** to **0.4 0.4 0.2.**

You can continue to add more instanced objects to create more variety and maybe change the size. You can use the **Relax Iterations** parameter if you need to separate the instances from each other a bit more.

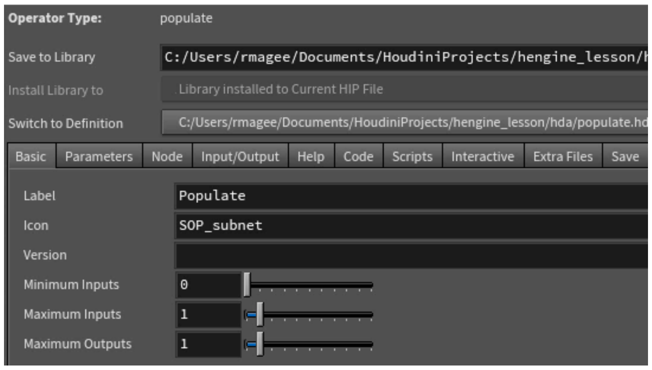


07 In the Generate section of the *building* asset's **Details** panel, press **Rebuild**. In the Bake section, turn on **Replace Preview Bake** then click on the **Bake** button.

Press **Play** to walk around the scene to see the instanced geometry in action. Now you can see that you are colliding with the cubes which already had collision geometry set up properly. You should always make sure your instanced objects have collision geometry set up properly.

PART SIX:
Use Geometry to Drive Asset

So far this asset supplies the input geometry for the populate asset. Another option is to use existing geometry in the Unreal scene to instance the geometry. You will now add an input node to your asset which will accept this Unreal geometry. The ability to work with objects on our level creates better integration between your procedural assets and existing game art.



Operator Type: populate

Save to Library: C:/Users/rmagee/Documents/HoudiniProjects/hengine_lesson/h

Install Library to: Library installed to Current HIP File

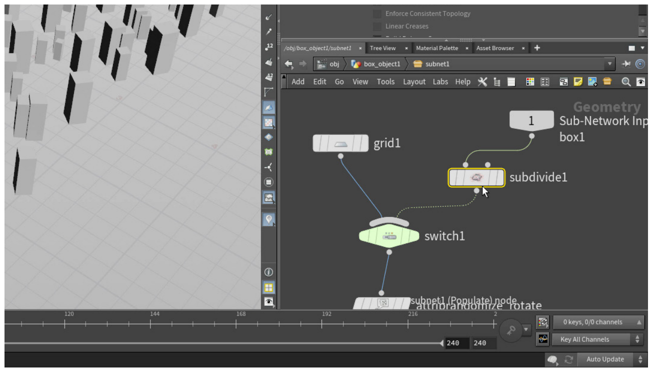
Switch to Definition: C:/Users/rmagee/Documents/HoudiniProjects/hengine_lesson/hda/populate.hd

Basic | Parameters | Node | Input/Output | Help | Code | Scripts | Interactive | Extra Files | Save

Label	Populate
Icon	SOP_subnet
Version	
Minimum Inputs	0
Maximum Inputs	1
Maximum Outputs	1

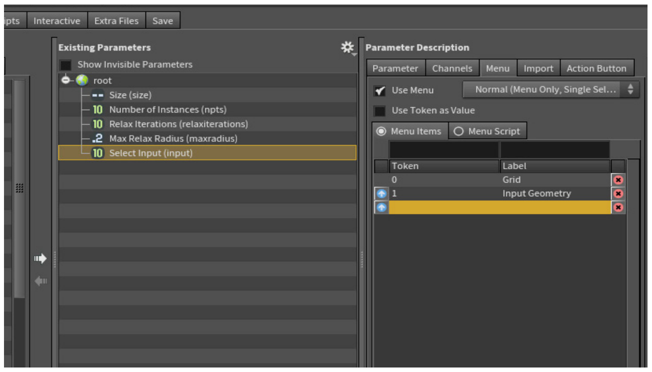
01 **IN HOUDINI** – Choose **Assets > Edit Asset Properties > Populate.** Stay on the **Basic** tab and set **Maximum Inputs** to **1**. Click **Accept** on the Type Properties window to save the changes. This will create an input node inside the asset that you can wire into the network. This input will let you grab geometry from the Unreal scene later.

Make sure you leave **Minimum Inputs** set to **0**. If it is set higher than that then the asset will ONLY work if the minimum input requirement is met. Otherwise the asset won't cook and you won't see anything.



02 In the Network editor, add a **Switch** node after the grid and wire in the new Input. Add a **Subdivide** node then wire it after the input node to make sure there is enough detail for the randomizing of attribute values. Set **Depth** to **5**.

You are getting an **Error** on the subdivide node because in Houdini there is nothing feeding into the third input. Go back up one level and feed a **box** into the asset's input to help preview how this is going to work. The box will subdivide into a sphere in Houdini which is not how it will work in the final scene.



Existing Parameters

Show Invisible Parameters

root
- Size (size)
- Number of Instances (npts)
- Relax Iterations (relaxiterations)
- Max Relax Radius (maxradius)
- Select Input (input)

Parameter Description

Parameter | Channels | Menu | Import | Action Button

✓ Use Menu — Normal (Menu Only, Single Sel...

Use Token as Value

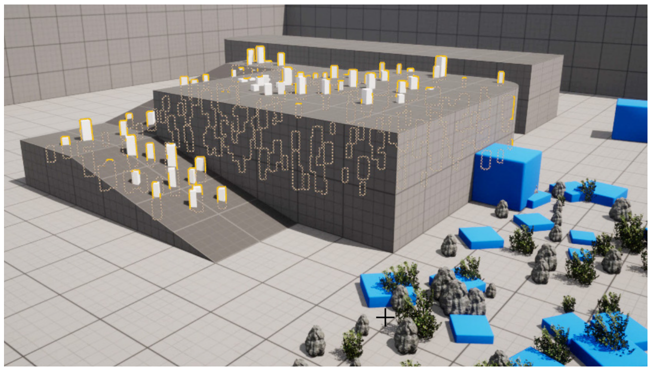
● Menu Items ○ Menu Script

Token	Label
0	Grid
1	Input Geometry

03 Choose **Assets > Edit Asset Properties > Populate.** From the *switch* node, drag the **Select Input** parameter to the Parameter list. Select the **Select Input** parameter. Click on the **Menu** tab then turn on **Use Menu**.

Add **0, Grid** then **1, Input Geometry**. Click **Accept** on the **Type Properties** window to save the changes.

This will add a menu to the UI of the asset which you can use in Unreal.



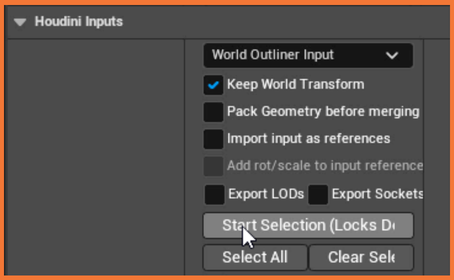
04 **IN UNREAL** – Click the **Rebuild Asset** button to accept the changes. Drag a new *populate* asset into the foreground. Go to the **Houdini parameters** section and from the **Select Input** menu choose **Input Geometry**.

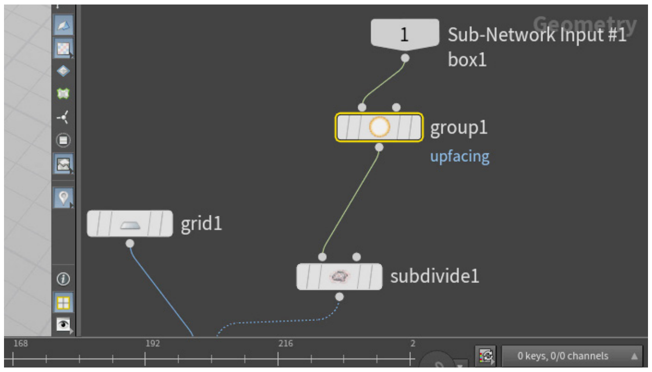
In the **Details** panel, go to the **Houdini Inputs** section and from the menu, choose **World Outliner Input**. Click on the **Start Selection** button and in the 3D scene, select all the parts of the ramp and platform. Click **Use Current Selection** and now the instances are being scattered inside the subdivide platform. This is not exactly what we need.

USING THE INPUT NODE IN UNREAL

When you set up an input node in your asset, there are a number of options for accessing input geometry. You can use Geometry from the content browser.

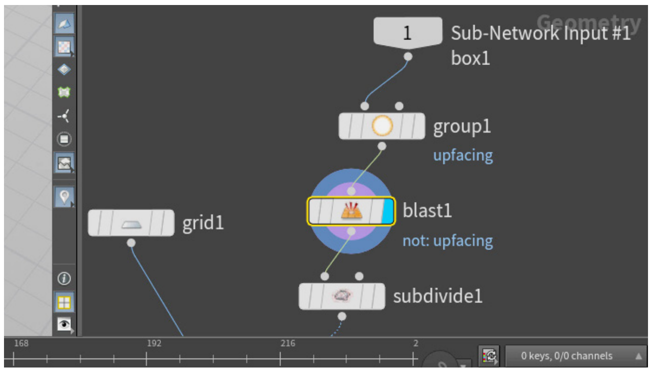
You can set up **Curve Input** where you draw a curve in Unreal. You can also grab content from the **World outliner** or input **Unreal landscapes** at heightfields for use with Houdini's new terrain toolset.


Houdini Inputs
World Outliner Input
✓ Keep World Transform
Pack Geometry before merging
Import input as references
Add rot/scale to input reference
Export LODs
Export Sockets
Start Selection (Locks D
Select All
Clear Sele



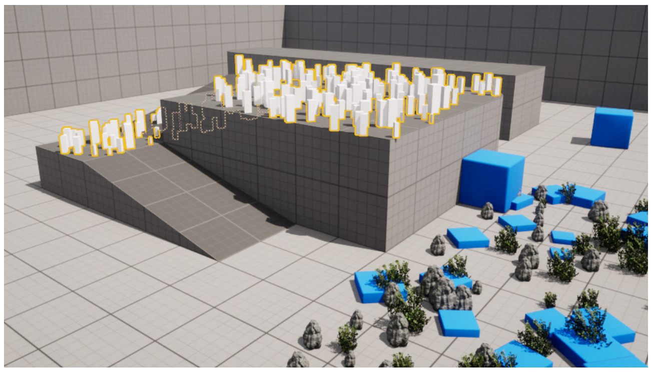
05 **IN HOUDINI** –You are now going to isolate the top faces of the box to limit where the points are being copied. Insert a **Group** node after the *input* node in the network editor. Set the **Group Name** to **upfacing**. Under **Base Group**, turn **Enable** to **Off**. Under **Keep by Normals**, turn **Enable** to **On**. Set the **Direction** to **0, 1, 0** and **Spread Angle** to **0**.

Instead of scattering points to all the faces of the geometry in Unreal, you are going to isolate the top faces and use them instead.



06 Insert a **Blast** node after the *group* node and set the **Group** to *upfacing* and turn **Delete Non Selected** to **On**. Now only those primitives facing up will be kept - the others will be deleted. Choose **Assets > Save Asset > Populate**.

The key here is that because you used a group to identify the top faces, it doesn't matter what geometry is input into the asset the solution will work. This is how a procedural asset works because it provides a generalized solution instead of a specific one-off solution.



07 **IN UNREAL** – Click the **Rebuild Asset** button to accept the changes. Now only the top faces of the geometry have boxes on them.

This asset can now use a default grid or geometry sourced from the level to work. There are always lots of options available when building Houdini Digital Assets for use in Unreal.

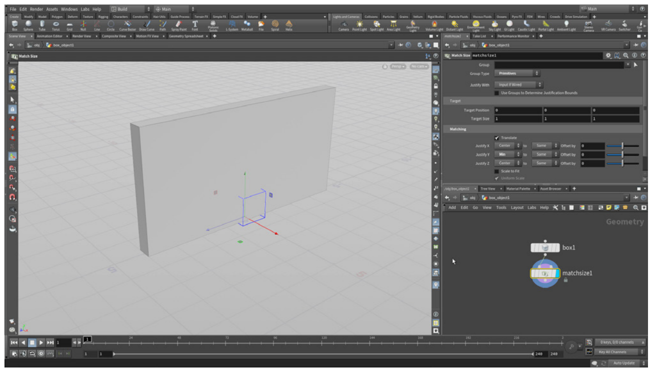
08 Set the **Number of Instances** to **40**. Go to the **Houdini Outputs** section and expand the instanced box.

In the Content Browser, open **StarterContent > Particles**. Drag the *P_Fire* prop over to the **Houdini Instanced Input**. Set the **Rotate Y** to **90**. Press **Play** to test the level.

The instanced points in this asset can be used for more than just geometry. They are now a part of the Unreal level and can be used to solve different problems. You may want to hide *populate2* with the fire while you add more to your scene. It will still appear when you play the level.

PART SEVEN: Import RBD Simulation into Unreal

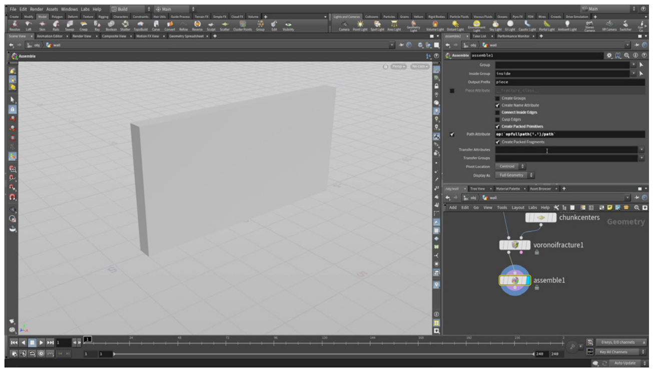
In this part of the lesson, you are going to create a wall then smash it using rigid body dynamics. Export this system to FBX then import into Unreal for use in your game. This is a simple example of bringing visual effects from Houdini to Unreal.



01

IN HOUDINI - Create a **Box**. Go to the geometry level and set its **Size** to **0.5, 4, 8.**

Add a **Match Size** node and in the Parameter pane, under **Matching**, set **Justify Y** to **Min.** This raises the box up so that it sits on the ground.



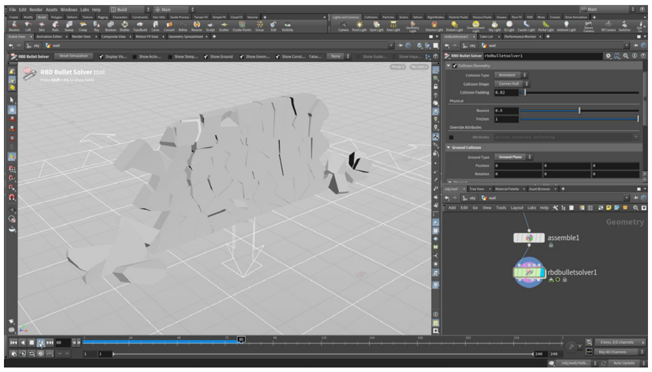
02

Go to the Object level. Rename the node *wall*. Select the wall node and from the **Model** shelf, select **Shatter**.

Dive down to the geometry level, select the *chunkcenters* node and set the **Force Total Count** to **100**.

Add an Assemble node at the end of the chain. Turn off **Connect Inside Edges** and turn on **Create Packed Primitives** and **Path Attribute.** Change the **Path Attribute** to:

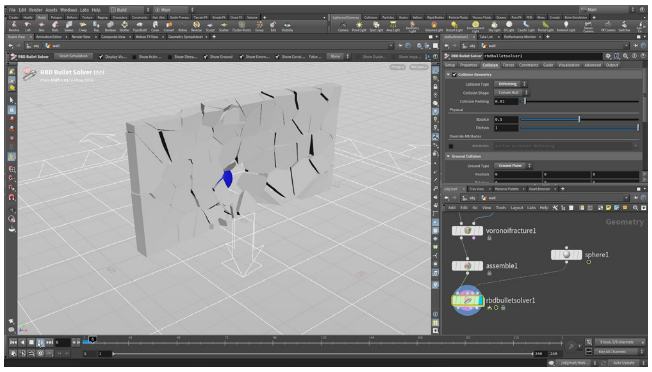
```
op:`opfullpath('.')/path'
```



03

Add an **RBD Bullet Solver** node to the end of the chain. On the **Ground** tab, set **Add Ground Plane** to **Ground Plane**.

Press **Play** to run the simulation. Not much happens because the pieces of the wall are just falling down.



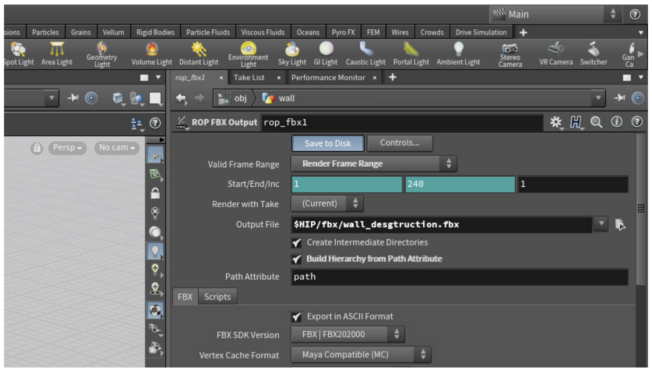
04

Add a **sphere** node to the network. Set it in front the wall with a **Center X** of 2 and a **Center Y** of **1**. Press the **Alt** key and click on **Center X** to keyframe it.

Go to **frame 9.** Set **Center X** to **-3** and **Alt-click** on **Center X** to set a second keyframe.

Wire the sphere into the **fourth** collision input of the *rbdbulletsolver* node. Under **Collisions** set **Collision Type** to **Deforming.**

Press **Play** to run the simulation. The wall is now being smashed by the sphere which will be hidden in the simulation.



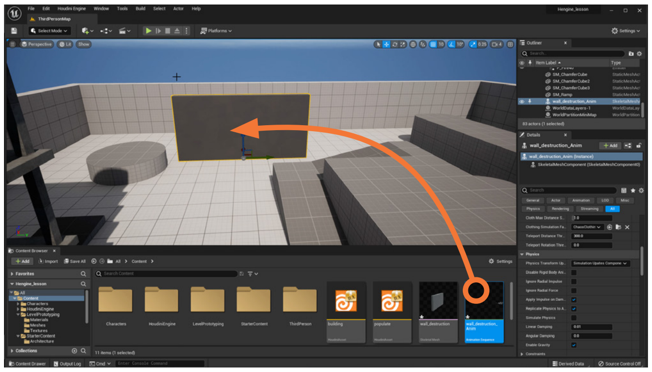
Valid Frame Range: Render Frame Range
Start/End/Inc: 1 240 1
Render with Take: (Current)
Output File: `$HIP/fbx/wall_desgtruction.fbx`
✓ Create Intermediate Directories
✓ Build Hierarchy from Path Attribute
Path Attribute: path
✓ Export in ASCII Format
FBX SDK Version: FBX | FBX202000
Vertex Cache Format: Maya Compatible (MC)

05 Add a **Transform** node after the *rbdbulletsolver* node then set **Uniform Scale** to **100**. This scales the simulation to the size needed in Unreal.

Add an **FBX Output** node. Set **Valid Frame Range** to **Render Frame Range** then set **Output File** to *$HIP/wall_destruction.fbx*.

Turn on **Build Hierarchy from Path Attribute.**

Click **Save to Disk** to save to disk.



06 **IN UNREAL** – In the Content Browser, go back to the *Content* directory. Press **Import** and grab the *wall_destruction.fbx* file. Check **Skeletal Mesh, Import Mesh, Import Animations**.

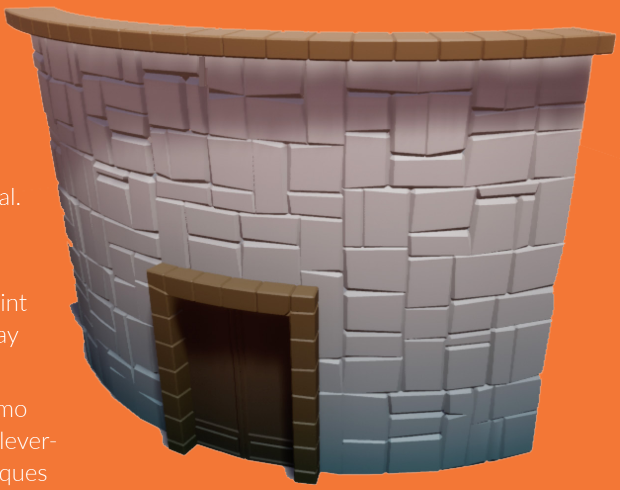
Expand the Animation section and make sure **Import Meshes in Bone is checked off.** Expand the **Mesh** section and set **Normal Import Method** to import **Normals and Tangents**. Click **Import**.

A panel will come up saying that UVs haven't been set up yet. Close this. Four new items appear in the content list.

Drag the *wall_destruction_Anim* asset into your workspace.

07 Press **Play** to see the simulation working in game. Right now the animation just loops and doesn't interact with anything. You could set up a Blueprint to trigger the animation based on some action on the character's part.

CONCLUSION



You have now created a variety of game assets for use in Unreal. From Houdini Digital Assets to FBX files containing rigid body simulations, you now have a good foundation to build from. These Digital Assets let you integrate Houdini's node based workflow inside host applications such as Unreal. The same workflow works with other applications such as Unity, Autodesk Maya and Autodesk 3DS Max.

go to **SideFX.com/unreal** for more information**.** This will point you to a starter kit of assets you can use in Unreal right away along with a link to more tutorials.

Be sure to take a look at **Project Titan** an in-house tech demo designed to explore the creation of a 3D environment that leverages the latest technologies in Unreal. The tools and techniques created for Project Titan have been shared with the community as learning materials and downloadable content.