Packed Primitives

What Are Packed Primitives?

Packed Primitives are a way to express a procedure to generate geometry at render time.

Packed Primitives have **information** about other pieces of geometry embedded inside of them. This information could be an actual piece of geometry stored in memory, a reference to a smaller part of another piece of geometry, or even a path to geometry stored on disk.

The information can then be used throughout Houdini to more efficiently represent geometry in the viewport, in the bullet solver, and in Mantra.

The Types of Packed Primitives

Packed Primitives can "embed" different types of data about geometry for use in different scenarios. Each of these Packed Primitive Types have advantages and limitations and are generally tailored to be used in specific circumstances.

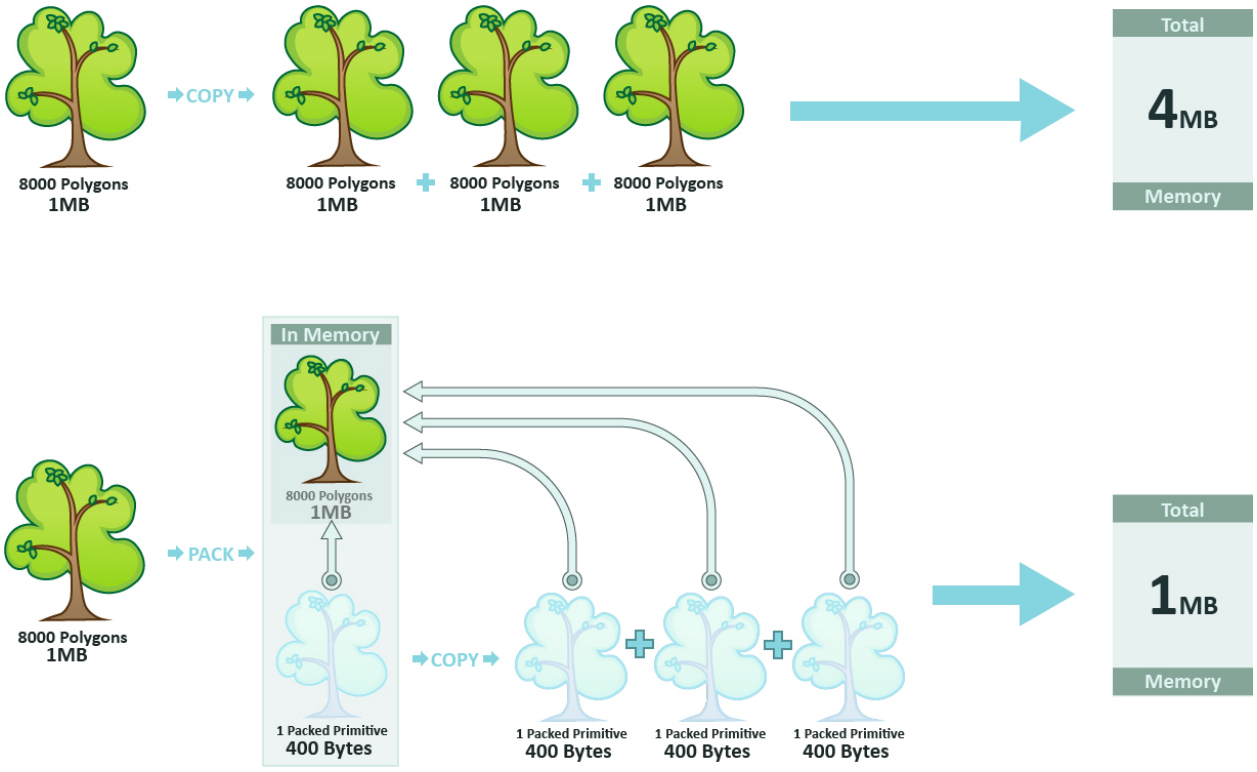
In Memory Packed Primitives

An "In Memory" packed primitive is generated by "Packing" the geometry directly in a SOP network. This creates a **Packed Geometry Primitive** with an embedded reference to the current version of your geometry stored in RAM.

In practice, the "embedded" geometry essentially becomes a single un-editable "primitive" with a single transform.

Advantages

Because the "embedded geometry" simply ***refers*** to a piece of geometry in memory, copying a packed primitive creates a **copy of the reference** rather than a copy of the geometry itself. This means that the referenced geometry is shared among all copies of the packed primitive. This stands in contrast to copying standard Houdini geometry, which creates duplicates of all points, primitives, etc., in the original piece of geometry.



Copies of packed primitives use less memory, are simpler to transform, and can be drawn more efficiently in the viewport or rendered by Mantra.

Additionally, because the geometry can exist in a traditional SOP network before being packed, you can easily generate procedural geometry which adapts to your scene, use stamping to generate variations of your packed geometry, or make interactive edits to your geometry while viewing the results live. Essentially, working with “In Memory” Packed Primitives is a more interactive and user-friendly version of traditional instancing workflows.

Individual copies of an “In Memory” Packed Primitive can also be “Unpacked” in a SOP network, loading the referenced geometry into memory. This allows you to generate procedural workflows which are a hybrid of traditional Houdini geometry and Packed Geometry Primitives.

A Helpful Reminder

“Packing” geometry has an associated memory cost. Since you are storing the original piece of geometry in RAM as well as the memory overhead for the “In Memory” Packed Primitive itself, a single packed primitive is not necessarily any more efficient than the original piece of geometry. The benefit of “In Memory” Packed Primitives comes from the efficient representation of large number of copies where the referenced geometry can be shared.

This is important to remember when copy-stamping packed geometry. If every instance of your packed geometry is unique, then you will not receive any of the memory or performance benefits. In fact, in this scenario you will use more memory than using standard Houdini geometry, since each packed primitive has its own data on top of the embedded geometry.

TIP*: It's possible to somewhat offset the cost of packing stamped geometry when there are limited numbers of stamped variations using the 'cache stamping' parameter - see the help for the Copy SOP for more info.*

Packed Disk Primitives

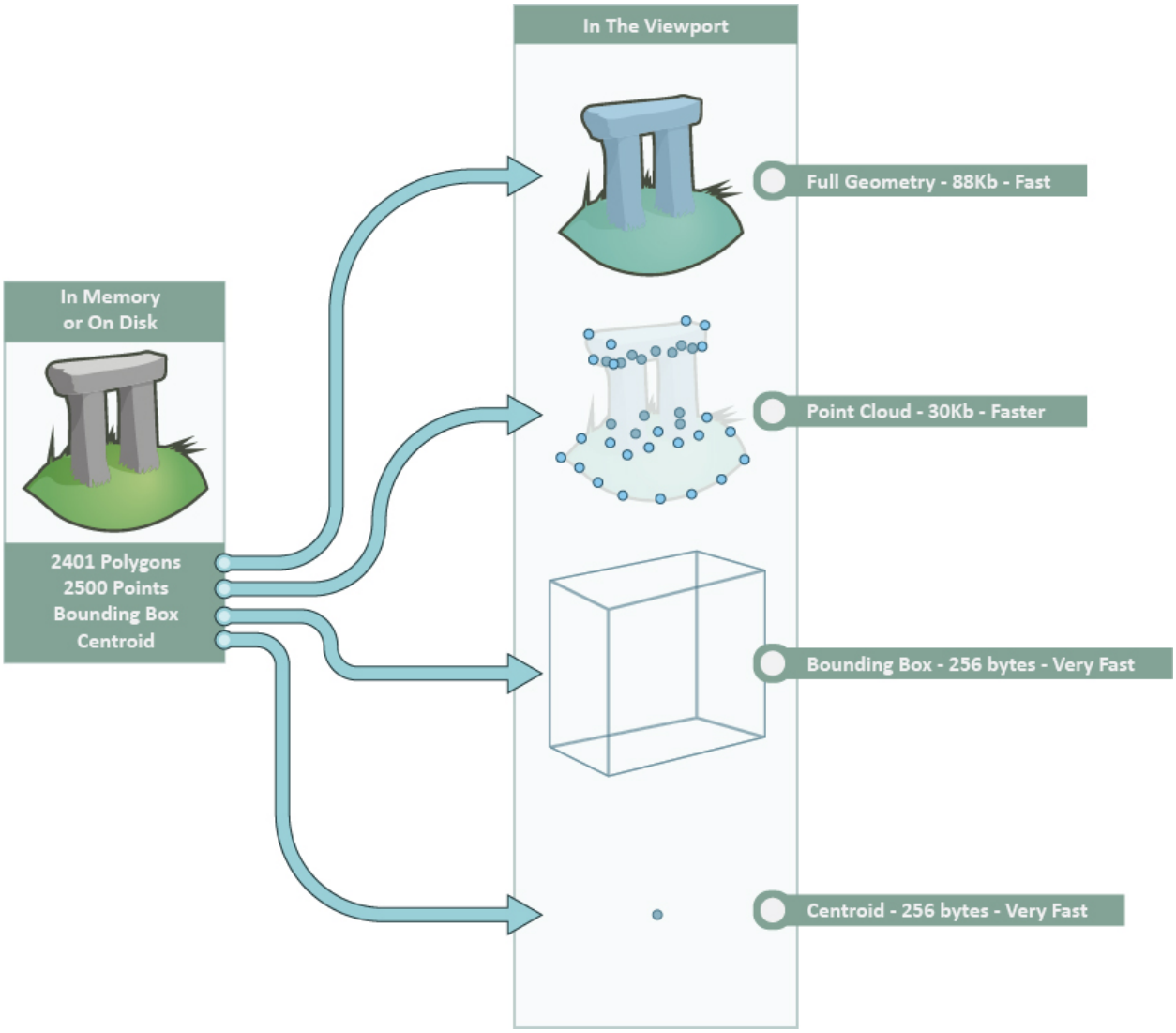
A "Packed Disk" Primitive has an embedded "path" to a file on disk rather than a reference to a piece of geometry stored in RAM. Generally speaking, "Packed Disk" primitives are standard Houdini Geometry which have been written to disk as a .bgeo, or .bgeo.sc file then loaded into Houdini through a File SOP as a "Packed Disk Primitive".

A "Packed Disk" primitive behaves in a very similar fashion to "In Memory" Packed Primitives, the "embedded" geometry is represented as a single un-editable primitive with a single transform

Advantages

Much like the "In Memory" packed primitives, a "Packed Disk" primitive is an excellent choice for efficiently representing copies of geometry in the viewport and in Mantra. Copying a "Packed Disk" primitive creates a **copy of the path to the geometry on disk** rather than a duplicate of the geometry itself.

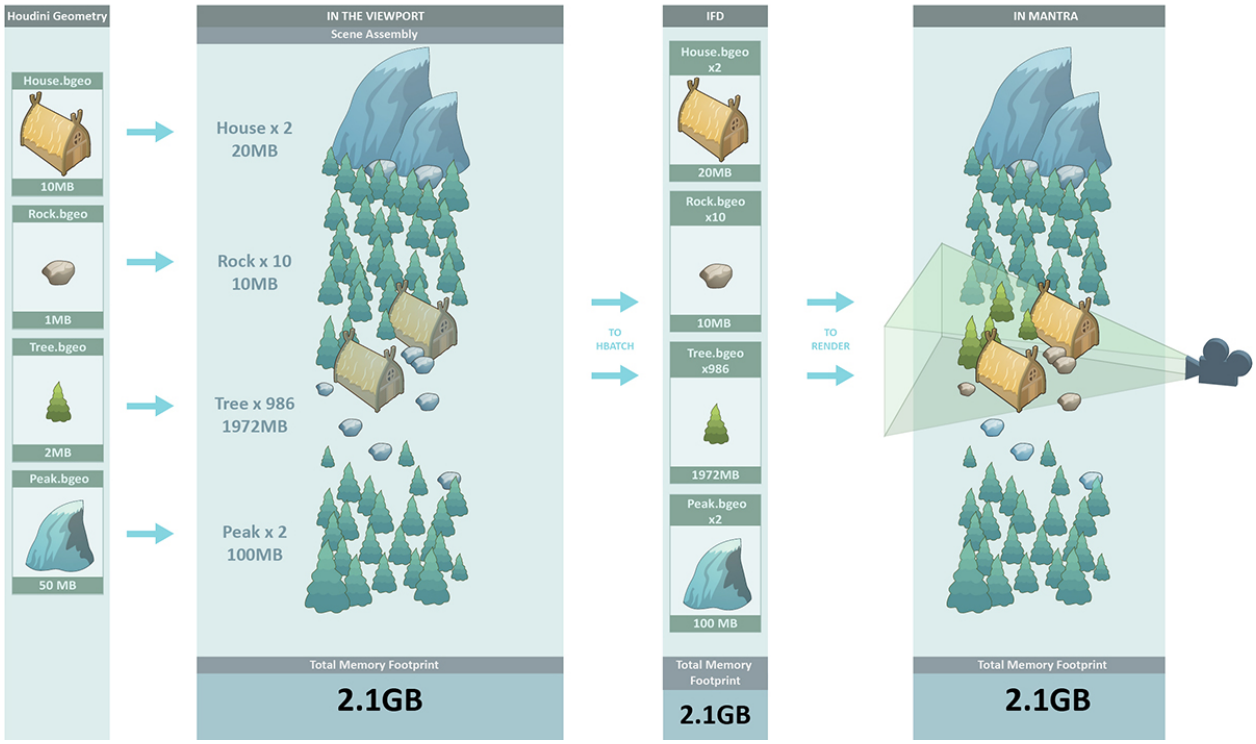
Another advantage shared between "In Memory" and "Packed Disk" primitives comes from how the geometry can be represented in the viewport. The viewport does not copy the geometry, but simply draws it multiple times with different transforms. This means that the viewport can also refer to a smaller subset of the referenced geometry and display that instead.



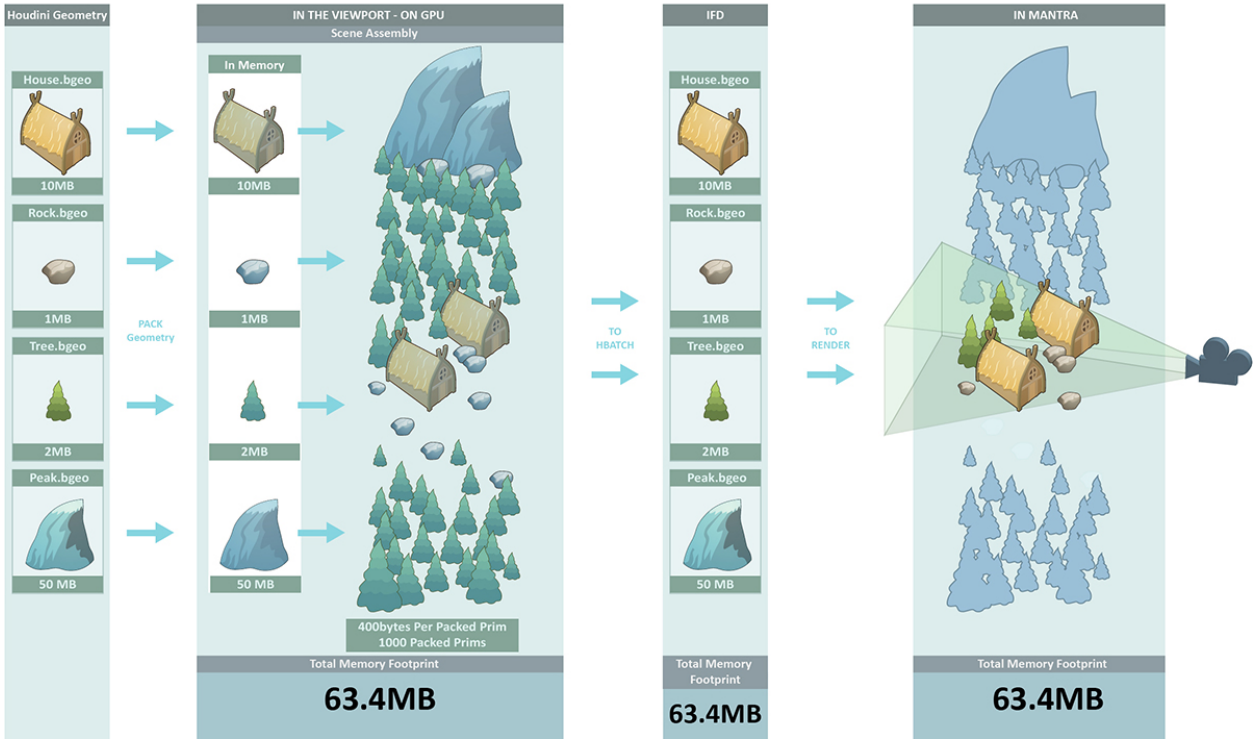
Since “Packed Disk” primitives, by their nature, are loaded from pre-generated geometry stored on disk they are less dynamic than “In Memory” packed primitives whose embedded geometry can be generated procedurally. The only way to make edits to “Packed Disk” primitives is to “unpack” them, however this causes the geometry to be loaded into memory as standard Houdini geometry, negating the benefits. In this sense, “Packed Disk” primitives are less flexible than “In Memory” packed primitives and best used for static geometry.

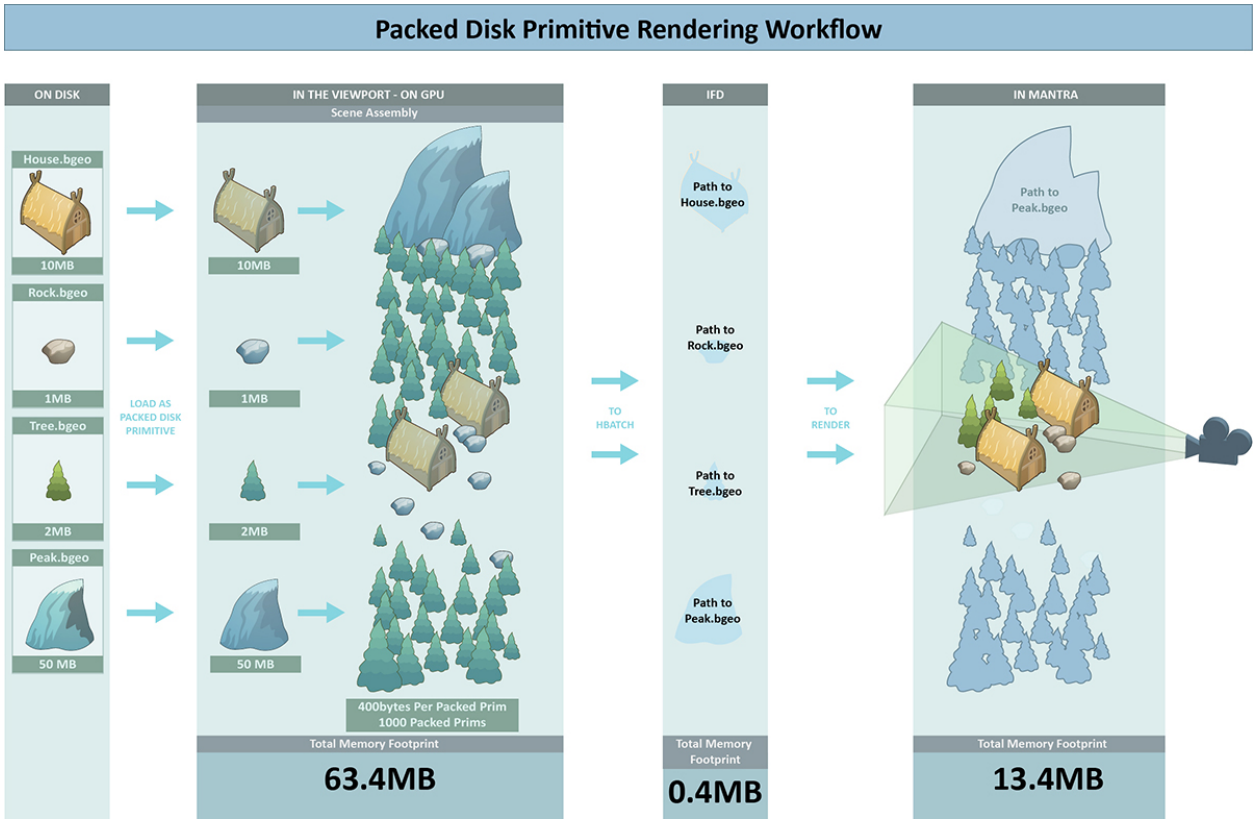
However, there are several advantages “Packed Disk” primitives have over “In Memory” packed primitives when used at render time. When generating an IFD (A file which contains a complete description of a scene and how to render It.), a “Packed Disk” primitive can be represented simply as **path to the file on disk**. In contrast, an “In Memory” primitive must have the entire piece of Geometry copied into the IFD in order to be referenced by Mantra. Both of these methods are superior to standard Houdini geometry which must include all of the geometry as well as all of the duplicates of the geometry in the IFD file.

Geometry Rendering Workflow



Packed Primitive Rendering Workflow





Additionally, Mantra never has to load into memory a “Packed Disk” primitive which isn’t currently being used to render the scene. Instead, “Packed Disk” geometry is streamed into the scene when necessary and then unloaded when no longer in use.

This means that single copies of “Packed Disk” primitives can still be useful at render time, saving memory in the IFD file, as well as reducing the amount of geometry Mantra needs to load at any given time.

The lightweight representation of “Packed Disk” primitives makes them ideal candidates for scene assembly, especially for static background objects. That said, the very small memory footprint in the IFD file also makes them very useful for objects with large on-disk footprints. (Like Fluid, Smoke, or RBD simulations).

Packed Fragments

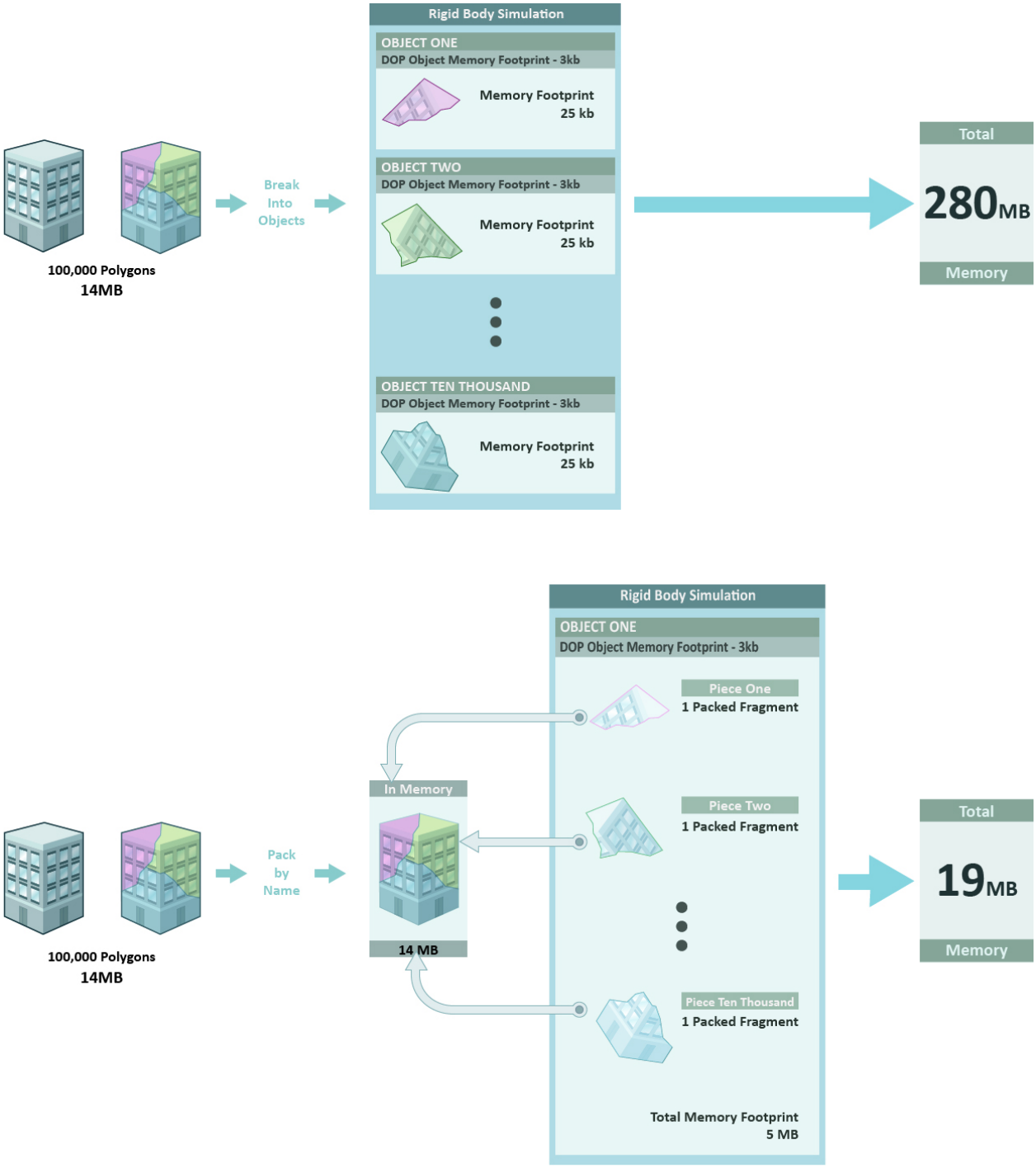
A “Packed Fragment” primitive is generated by “Packing” some piece of geometry along with a "name attribute”. Each piece of the geometry with a unique “name attribute” will become a “Packed Fragment” primitive with an embedded reference to the “complete” piece of geometry which is **shared** across all “Fragments”.

In practice, each “Fragment” essentially becomes a single un-editable “primitive” with a single transform.

Advantages

“Packed Fragment” primitives are ideal for representing many pieces of a greater “complete” piece of geometry. Each “Fragment” refers to some subset of the embedded geometry which is shared across all “Fragments”. When “Unpacked”, only the smaller subset of geometry will be loaded into memory.

Additionally, because each “Packed Fragment” represents a single reference and a transform, they are useful for cases where each “Fragment” will receive some unique transformation such as a Rigid Body Simulation. This stands in contrast to standard Houdini geometry which does not share its geometry, so each individual piece must be considered its own object.



"Packed Fragment" primitives use less memory, are simpler to transform, and are more efficiently displayed in the viewport.

A Helpful Reminder

Each “Packed Fragment” contains a reference to the larger piece of embedded geometry stored in memory. When you have many “Fragments”, this is a very efficient way of representing the geometry because each “Fragment” only refers to a small subset of the **shared** Geometry. However, if you were to delete many of your fragments, leaving only a small number, each “Packed Fragment” is still referring to the original “complete” piece of geometry which is stored in memory. This can potentially mean a large amount of memory overhead which is no longer necessary.

Consider “Unpacking” your fragments when you have much fewer “Fragments” than in the original piece of Geometry.

Rendering Packed Primitives

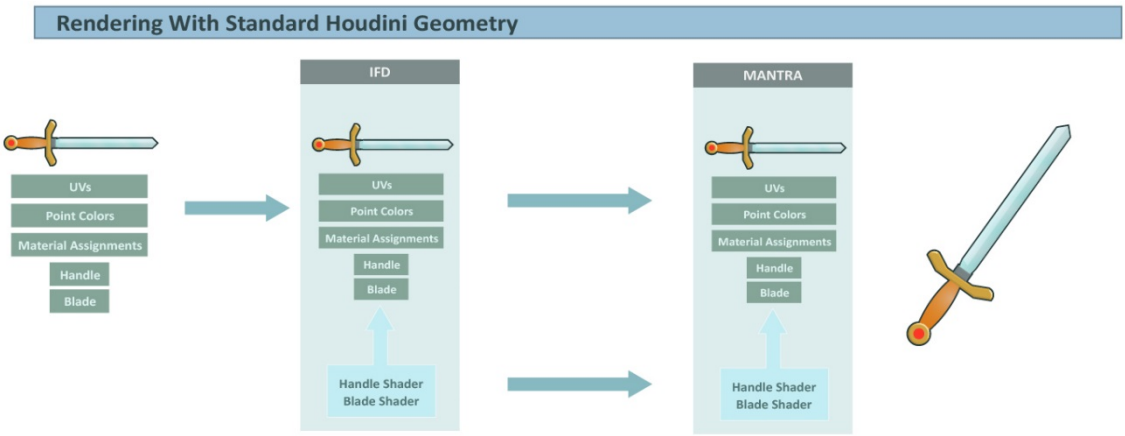
Packed Primitives are extremely useful for rendering in Mantra. In general, the proper use of packed primitives will allow you to increase the speed of your renders as well as reduce the overall amount of memory needed. Additionally, IFD generation will be faster and use less on-disk memory.

However, it is important to understand how Mantra deals with Packed Primitives and the data stored inside of them in order to take full advantage of them at render time.

Material Assignment

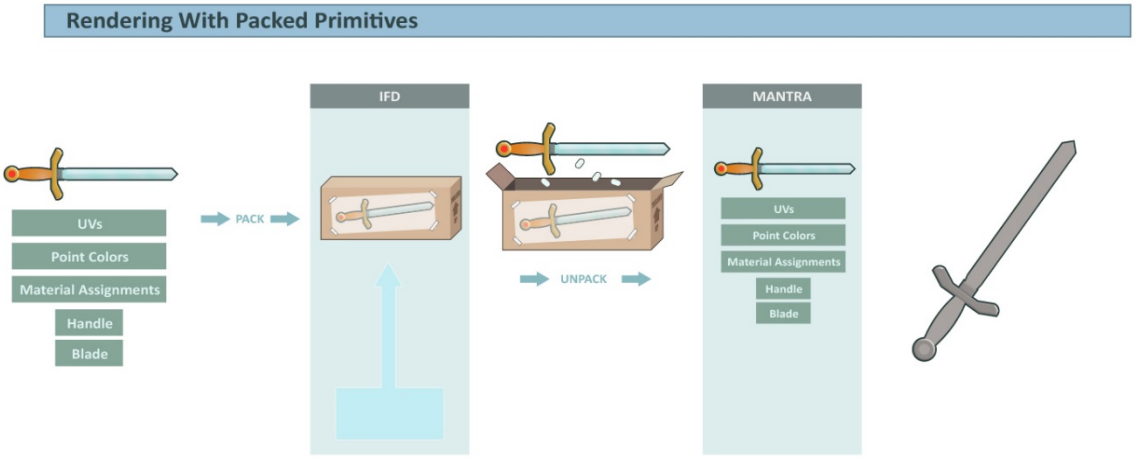
With standard Houdini geometry, Material assignment can occur at two levels – the object level (On the object Node), or the primitives inside of the object (Using a Material SOP). Materials assigned at “lower levels” override materials in the higher levels.

When sending a scene to be rendered, it is first analyzed to see which materials will actually be needed in the final render. Houdini checks the objects for any material assignments, along with any geometry attributes which apply materials, then makes sure to include the appropriate Shaders in the IFD file.

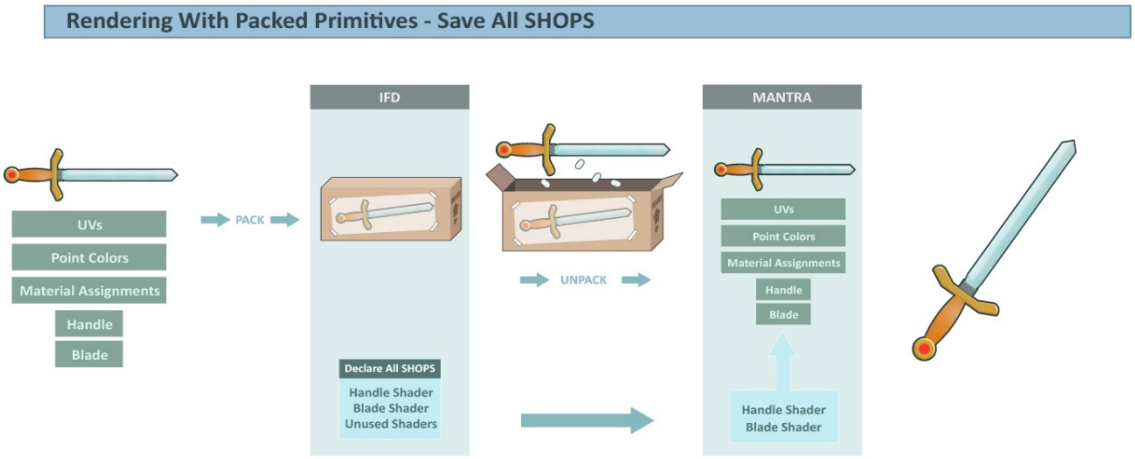


When using Packed Geometry, this process is made more complex by adding a third level of material assignment – materials assigned by attributes **inside** the Packed Primitive. Like the previous example, materials assigned at "lower levels" override
Materials in the higher levels.

However, in this case, when the scene is sent to be rendered, the material assignments "inside" the packed primitives are hidden from Houdini. (Remember the Packed Geometry may simply refer to an object on disk). This means that Houdini will be unable to add the appropriate Shaders to the IFD file for use at render time. During rendering, Mantra will unpack the object, find the material assignment, but it will not have the necessary Shaders to apply to the object.



The solution to this problem is to tell Houdini to include the Shaders in the IFD regardless of whether or not they have been assigned to any objects or primitives. On the Mantra node, there is an option called "Save All SHOPS" which will embed all Shaders in your scene in the IFD. (This will increase the on-disk size of your IFD by a small amount). This way, when Mantra unpacks the geometry at render time and finds the material assignment, the necessary Shaders will be available.



In general, when working with Packed Geometry, it is important to remember that "packed" data is only accessible once it has been unpacked. For more information about assigning shaders and overriding shading parameters "inside" Packed Geometry, please see the documentation on Material Style Sheets.

Displacement and Subdivision Surfaces

In general, when using Packed Geometry, displacement shading and subdivision surfaces are handled in the same way as with any other piece of geometry. However, if you are primarily using your Packed Geometry as instanced geometry, then some care must be taken to get the most out of your workflow.

Before rendering a displaced or a subdivided surface, the geometry is "diced" into smaller primitives such that there will be one primitive for every pixel (with shading quality set to 1). This means that objects closer to the camera will be diced more than objects in the distance (which have less pixel coverage). However, for instancing, this can cause a problem. As discussed previously, the benefit of instancing comes from the fact that geometry is shared across all instances. In the case of displacements or subdivision surfaces, the objects must be evaluated and diced individually which means the geometry is no longer being shared.

To avoid this problem, there is a rendering property which can be added to the object containing your instances – "vm_sharedisplace". Enabling this parameter will tell Mantra to use the **highest level of dicing** needed for the scene on one object and then share the diced geometry across all instances. Keep in mind that this means that objects far away from the camera will have the same level of dicing as objects very close to the camera. There is some potential for this to cause problems in your renders, however, the benefits of instancing the geometry most likely outweigh any downside.

In the worst case scenario, where “incorrect” dicing levels cause problems in your rendering, you could “split” your instances into two objects, foreground and background, so that distant objects are evaluated separately from nearby ones. Alternatively, you could also unpack any objects close to the camera, essentially removing them from the instancing hierarchy.