Instancing

When rendering very complicated scenes, it is often the case that much of the scene is comprised of objects that are identical, or nearly identical. For example, the external facade of a building may be made up of many identical windows or bricks. RenderMan provides a powerful **object instancing** facility that allows the renderer to take advantage of this repetition by allowing the sharing and reuse of the geometric representation between near-identical copies. In the following image, comprised of over 75,000 subdivision mesh teapots, rendering without instances requires a memory footprint of over 12.5 GB, even with the coarsest possible tessellation settings; rendering with object instances requires a memory footprint of less than 400 MB! This allows you to render more objects without running into limits on your machine.

Geometry is represented by a prototype and an instance. The prototype geometry representation is shared by all instances and includes all the relevant geometric details (vertices, faces, primitive variables, etc) Each instance has its own transformation, bxdf assignments, light linking, and attribute state, but is otherwise an identical clone from the prototype. The renderer will reuse most of the memory associated with the prototype with each instance.

> You can alter the Dicing Strategy on Instances by applying the attribute but it is not required to render them.

Prototype Attributes

Prototype Attributes are typically the heavyweight geometric data that is best shared for efficiency reasons, like normals, vertices, etc. Note that these attributes *can only be used* on the prototype object. You can find these listed here.

Instance Attributes

Instance Attributes allow for variation and more for multiple different instances and are *only available* to instances, not the prototype. Things like bxdf assignment, light linking, trace sets, and more can be assigned per instance. You can find this list here.

Tessellation

RenderMan tessellates geometry using an adaptive technique such that the resulting micropolygons have a length measured in a small number of pixels. This is typically the instance closest to the camera. However, this technique is not useful because, with this screen projection-based technique, no single tessellation can be reused for an object prototype instanced at different distances from the camera. Instead, the default tessellation is a world distance-based measurement: the micropolygon length is measured directly in the units used in the scene, without projecting to the camera.

Nested Instances and Flattening

By default, RenderMan attempts to flatten instances to improve performance and allow for more options when applying variation. However, there are times when this might not be desired. Large scenes, like a forest scene, may have worse performance when flattened especially when it comes to memory usage. Flattened instances multiply when nested, i.e. 1,000 instances of 1,000 instances becomes 1,000,000 instances. Nested instances add, i.e. 1,000 instances of 1,000 instances is 2,000 instances. As such we have an option to manually disable this behavior by choosing Nested Instancing exposed in the bridge products.

Attributes that are not supported when nested are:

- Lighting Subsets and Exclude (light linking)
- Lightfilter Subsets
- Visibility Flags (camera, indirect, transmission visibility, etc)

Note that they will inherit the properties of the parent instance, procedural, or archive.

There is an internal limit on nested instances, after 4 levels/parents, after this limit we automatically flatten the hierarchy.

Shading Variation

Even though the driving factor behind instancing is to minimize the variability between copies in order to maximize reuse and save memory, it is often still very desirable to be able to shade each variant differently. RenderMan supports shading variation primarily by the following methods:

- by binding different materials (Bxdfs and upstream pattern graphs) to each object instance. Materials bind to object instances only. Materials bound to a group or procedural instance bind to any unassigned objects but do not override local bindings.
- by linking different lights and light filters to the object instance.
- by setting different user attributes per object instance, and driving Bxdfs using these attributes. RenderMan supports the use of completely arbitrary user attributes that may be set differently for every object instance in the scene. The use of these user attributes, combined with a pattern node, lends itself well to driving shading variation, particularly if the values of those user attributes can be derived automatically.
- by varying instance attributes across instances such as matte, holdout, and visibility settings.
- PxrVolume allows for instance overrides on BxDF parameters, *except* for density and motion blur. This data cannot be manipulated per instance.