Geometry

Viewport vs final render - Soul © Disney/Pixar

RenderMan supports a full range of geometric primitives, including polygons , NURBS , subdivision surfaces , curves , volumes , and implicit surfaces . Obj ect instancing is fully supported as well.

Choosing Primitives

Polygons are simple and individually lightweight, but detailed models and curved surfaces may require many polygons for acceptable quality, quickly becoming memory-intensive (even with RenderMan's very efficient implementation). However, their efficiency is hard to beat especially when rendering collections of hard objects such as buildings or cities.

NURBS are true curved surfaces, are efficient, and yield very smooth results. However, they require a restrictive quadrilateral-only topology that may be demanding to model.

Subdivision surfaces generally provide an easier way to produce smooth, curved surfaces with almost no modeling restrictions that are well suited for animation purposes. Many polygon meshes can be converted arbitrarily to subdivision surfaces and will immediately gain the benefit of a smooth appearance. However, note that subdivision surfaces inherently have a higher memory cost, and dense subdivision surfaces may be less efficient than the equivalent polygon mesh if the geometry is highly over-detailed.

Curves are specialized primitive which excel at rendering long, thin geometry such as hair, fur, or blades of grass.

Implicit surfaces are another type of specialized primitive, most often used when dealing with fluid simulations generated by a third-party package. They can also be used to render viscous fluids directly.

Volumes are used to render participating media with a heterogeneous density such as fog, smoke, or clouds.

Tessellation

In RenderMan, some primitives may get tessellated at run-time. This may happen for NURBS and subdivision surfaces (in order to provide a smooth surface), or for primitives associated with a displacement shader (in order to provide a detailed mesh to displace).

Note that the **tessellation will not impact the frequency at which shading occurs** (because shading happens at each ray hit). In particular, a coarse tessellation shouldn't affect any texture filtering, or yield blurred patterns.

However, **tessellation will impact the frequency at which displacement is computed** (because displacement happens for each vertex of the tessellated mesh).

Micropolygonlength

Primitives are tessellated into *micropolygons*. The degree of tessellation is driven by the ***dice micropolygonlength*** .

```
Attribute "dice" "constant float micropolygonlength" [1]
```

This is the average *micropolygon edge length* that the tessellation will try to produce. **The larger the value, the larger the micropolygons.**

- larger *micropolygonlength* means *coarser* tessellation, your object may show faceting or sharp edges but should have lower rendering cost
- smaller *micropolygonlength* means *finer* tessellation, your object should be smoother and better fit your ideal shape with a higher rendering cost

By default, the micropolygonlength value is expressed in term of pixels on the screen (i.e. the length in pixels, of the micropolygon as projected on the screen). This can be modified by using the *dice* attribute *strategy*.

```
Attribute "dice" "string strategy" ["planarprojection"]
```


The *micropolygonlength* attribute is a successor to the (deprecated) *shading rate*. While shading rate was expressed in terms of *area*, micropolygonlength is (as the name indicates) a *length*.

This means that in order to get a similar tessellation level, using a *shading rate* value of X would translate in using a *micropolygonlength* value of sqrt(X):

- shading rate = 0.25, micropolygonlength = 0.5
- shading rate = 1, micropolygonlength = 1
- shading rate = 4, micropolygonlength = 2

This is a useful conversion if you need to match previous tessellation.

Choosing a micropolygonlength value

Most of the time, a *micropolygonlength* value of 1 (default) should work well. If your surfaces are smooth and non-displaced, you may be able to use larger values (2 or 4) before seeing any difference.

Occasionally, if the geometry is very detailed and/or displaced, you may need to use smaller values of *micropolygonlength*. However, note that smaller micropolygons will result in larger memory consumption and slower render times.

You can find more about settings on geometry by looking at the Prototype Attributes or "PrimVars" offered by RenderMan.

Examples

RenderMan includes examples RIB files and shading plugins demonstrating its capabilities. These examples can be found in the `PixarRenderMan-Examples-XX.X/scenes` directory.