Implicit Surfaces

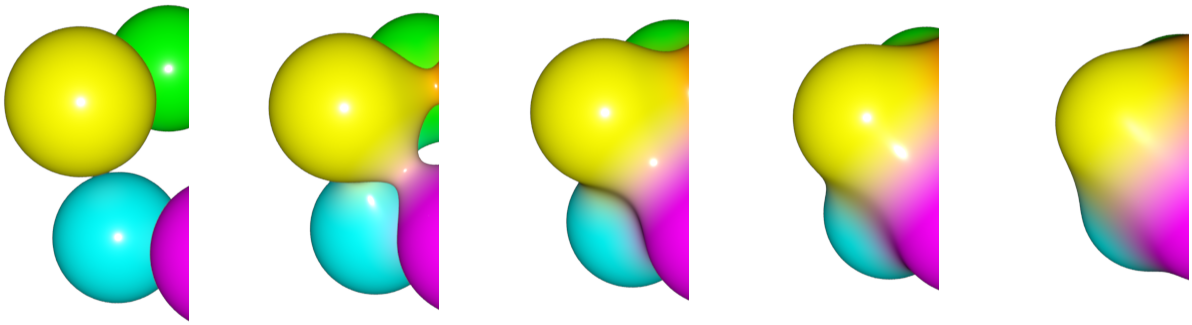
Introduction

RenderMan provides the ability to render **implicit surfaces** (sometimes referred to in RenderMan as "RiBlobby" or blobby surfaces). Most geometry types in RenderMan have explicit control meshes: the geometry is specified by a collection of vertices that are on or near the surface. Implicit surfaces are different: they are built up from primitives that are defined *implicitly*, by use of a mathematical function that is equal to some threshold value at every point on the surface. This complicated math is hidden by the renderer but leads to a representation that allows for a very efficient way to model geometry in situations where users want to create amorphous shapes, and are willing to trade away fine control over details of shape for ease of modeling gross features.

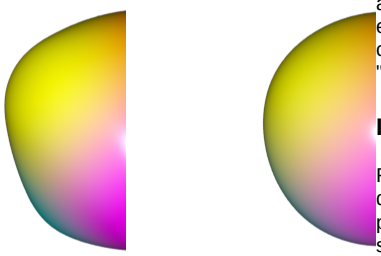
Built-In Primitives

One way of building an implicit surface in RenderMan is to build them up by combining a simple built-in primitive shapes: an ellipsoid. Here, the most important feature of implicit surfaces is *automatic blending*: adjoining primitives will meld together into rounded, tubby, globular masses. When the primitives get close enough to each other, this blending happens automatically (or not, as the modeler wishes), with no need to construct explicit round and fillet patches joining them. The following demonstrates six multi-colored spheres that start out at some distance from each other; at this distance, they do not interact. As they move closer, however, they begin to stretch and blend together, smoothly merging into a larger mass. This blending happens automatically and efficiently in the renderer, without incurring extra memory costs; implicit surfaces tend to be much more memory efficient than creating the equivalent polygonal tessellation in a modelling program.

-
-
-
-
-



-
-



When using the built-in primitive ellipsoid shapes, the underlying mathematical function is such the resulting blended surface has approximately the same volume as the constituent primitives. This makes this workflow an excellent choice for modelling things like viscous fluids, where this volume-preserving property is highly desirable. However, as mentioned above, there is no fine control over how the shapes are blended together: either the primitives interact and blend, or they do not. The threshold at which they blend can be set globally for the entire surface (which will cause the entire implicit surface to "bulk up" or "shrink down"), but the threshold cannot be set on a sphere by sphere basis.

Implicit Field Plugins

RenderMan allows users to extend the built-in primitive shapes to other mathematical descriptions that can seamlessly take part in automatic blending with the built-in primitives. This extension involves the use of **implicit field plugins**, which are C++ shared libraries loaded by the renderer. The renderer ships with several plugins that add the ability to primitives in the shape of cubes and segments (sausage-like cylinders with rounded ends).

However, implicit field plugins also lend themselves to a very different approach to using implicit surfaces: by taking as input an implicit field representation generated in another program. Implicit fields are a natural and efficient representation for fluid and volume effects, and can be output directly by programs such as Maya Fluids, programs that output OpenVDB files, or Bifrost. Implicit field plugins can be written to bridge the gap between these third party implicit data file formats, and the RenderMan implicit primitive, allowing the output of fluid simulations to be rendered directly and efficiently inside RenderMan. Typically in such cases, the implicit surface consists of a single primitive which is conceptually just the implicit field plugin itself, with parameters that have special meaning to the plugin. Blending with other primitives is usually not a concern nor is it desired in this case. The renderer ships with two plugins of this type: one that reads Maya fluid cache files, and one that reads OpenVDB files.

Some implicit field plugins (like the OpenVDB plugin) can be used **both** with implicit surfaces **and** with the volume primitive. This reflects that fact that implicit fields also happen to be a good way to represent volumes. In the case of OpenVDB, the choice of which primitive to use is predicated by the type of OpenVDB file you are using ("levelset" versus "fogvolume").