RenderMan Interface (Ri)

Introduction

The RenderMan interface is designed to be a standard interface between modeling programs and high-quality rendering programs. As such, it has provisions for the specification of 3-D scene descriptions that include: hidden surfaces, spatial filtering, dithering, motion blur, depth of field, flat and curved surfaces, objects, constructive solid geometry, and programmable shading to express lighting conditions, shadows, and surface appearances, with sophisticated control over color, texture, and reflectivity. The design of the interface emphasizes the ability to convey, compactly and efficiently, the information required to produce rendered images that resemble actual photographs.

What follows is a very general overview of the RenderMan Interface that introduces the interface routines and terminology. It concludes with a brief description of how certain concepts that are common to 3D graphics are handled in the interface.

Background

The RenderMan Interface was proposed by Pixar in May, 1988, as a standard interface between modelers and high-quality renderers. To date, it is the only proposed rendering interface standard that includes provisions for all the features in a scene description required to synthesize photorealistic images.

At the time of its introduction, the RenderMan Interface was defined in terms of a C language binding, a set of 96 C procedures that provides a complete rendering interface. Since then, a bytestream protocol has also been defined and incorporated into the interface specification. This protocol is known as the RenderMan Interface Bytestream (RIB), and it not only allows a scene description to be stored as an ASCII or binary coded file but also provides a means for transport over a network.

For the sake of simplicity, the interface will be presented here in its C language form. The conversion to RIB is quite straightforward, and a full description of it can be found in The RenderMan Interface, Version 3.1.

Understanding the Interface

The RenderMan Interface describes a scene as a sequence of geometric primitives. This sequence specifies all the objects in a scene in terms of the variety of flat and curved surfaces that RenderMan supports. To control the positioning and shading of these objects, the RenderMan Interface maintains a graphics state, which is primarily just a current setting for each of a large number of parameters associated with the rendering of an object in a scene.

The graphics state consists of options, attributes, and the interface mode. Options are the rendering parameters that affect how an entire scene will be rendered, while attributes are the parameters that can differ from object to object in a scene. The interface mode constrains and controls the context of allowed RenderMan procedure calls, and it also maintains the graphics state stack. This stack allows various parts of the graphics state to be saved and reinstated at will.

To control the graphics state, there are procedures in the interface that change the current mode, options, and attributes. These routines are classified here as mode-changing procedures, option-changing procedures, and attribute-changing procedures. The interface also includes a set of geometric primitive procedures that describe the actual geometry in a scene.

The interface starts up with default option and attribute values, which are defined in the interface specification. Calling a RenderMan option- or attribute-changing procedure sets the values of certain related options or attributes in the graphics state.

A scene is begun by calling the mode-changing routine RiWorldBegin(), and its options are set from the option values in the current graphics state. These values then become frozen, and it is forbidden to use any option-changing procedures until the scene's description is finished.

In the scene, attributes are still free to change. Each piece of the scene's geometry is described by calling a RenderMan geometric primitive procedure; a given primitive gets its attributes from the current graphics state at the time it is passed through the interface.

The Interface Procedures

RenderMan routines all have names of the form RiSomething(). They can be classified quite cleanly into the following categories:

Geometric Primitive Procedures
Mode-Changing Procedures
Option-Changing Procedures
Attribute-Changing Procedures
Texture and Bookkeeping Procedures
Geometric primitive procedures in the RenderMan interface place geometric primitives in a scene. They support polygons, bilinear and bicubic patches, non-uniform rational B-spline patches, quadric surfaces, and retained objects.

Procedures for changing modes are for basic renderer and scene control, or to control special modes of the interface for retained object, CSG, and motion blur specification. Though most modes also affect the attribute stack, there are modes for additional stack control.

There are also procedures for changing options. Each one sets a closely related set of option parameters in the graphics state, and the majority of them control either the effects that arise from the camera model or the display processing that occurs in the renderer.

Procedures for changing attributes alter a related set of attribute parameters in the graphics state. As attribute parameters pertain to operations that occur on each primitive, they tend to affect either a primitive's geometry and positioning in space or its lighting, shading and the opacity of its surfaces. Included as a geometric attribute is the current transformation, which defines the current coordinate system for all point values. Because of the special importance of this attribute, there are specific procedures to manipulate the current transformation matrix, to concatenate nonlinear transformations into the current transformation, and to attach a name to a coordinate system.

Finally, there are miscellaneous routines to produce texture map files from image files and perform various bookkeeping tasks.

Shaders and Parameterlists

From the point of view of RenderMan Interface procedure calls, a shader is just the name of a compiled shading language file that exists in a known place. However, shaders also have parameters that can be set through the interface when the shader is brought into the graphics state. To handle this capability, certain RenderMan procedures take a list of arguments known as a parameterlist. Each parameter has a default value that is given in the shader, so it only needs to be in the parameterlist if it is to have a value other than its default. The parameterlist always begins after the last procedure-specific argument, and it consists of an optional paired list of arguments followed by the terminating token argument RI_NULL.

The paired arguments in the parameterlist each contain a token followed by a pointer to an array of values. The token is just the string for the name of the parameter, and the values in the array are the values that get assigned to the parameter. Since there are both point and color type variables defined in the Shading Language, it is possible for a parameter to have more than one floating-point value associated with it. For this reason, the rendering interface must know the type of every parameter appearing in the parameterlist. The names and types of parameters that appear in the standard shaders are already known to the interface. However, nonstandard parameters must be declared with the RiDeclare() procedure before they can be used in a parameterlist. Altering a shader's parameters affects the shader's presence in the graphics state, but it does not affect the shader itself. Therefore, every invocation of a given shader starts with the default parameter values and alters them according to the parameterlist in the call.

Other Parameterlists

The routines that invoke shaders are not the only RenderMan procedures that use parameterlists. RenderMan geometric primitive routines use them to define a variety of surface parameters in addition to position and shape, and RenderMan texture routines use them to specify optional texture parameters. There are also a few other routines that are designed to take extended sets of parameters using parameterlists.

Varargs

Since the length of a parameterlist is not always known at compile-time, RenderMan also provides a varargs form for each procedure that uses a parameterlist. These routines have names of the form RiSomethingV(), and each does the same thing as its RiSomething() counterpart, taking a fixed number of pointers to variable-length argument arrays instead of a variable-length parameterlist.

The RenderMan Graphics Environment

Before leaving this introduction, it is worth noting how the RenderMan 3D graphics environment works. In particular, it is important to know how RenderMan handles coordinate systems, transformations, camera positioning, and light sources. Mentioned here are the primary graphics conventions in RenderMan that are likely to differ from other 3D environments.

Coordinate Systems

The natural RenderMan coordinate system is a left-handed one in which the x-axis points to the right, the y-axis points up, and the z-axis points into the screen. For any geometric primitive in a scene, RenderMan keeps track of several coordinate systems, including "object", "shader", "world", "camera", "screen", and "raster" spaces. In addition, any coordinate system can be named with the RiCoordinateSystem() procedure. The interface starts up in camera space.

Transformations

Before worldBegin() has been called, transformations are used to position the camera by defining world space (see Camera Positioning, below). After RiWorldBegin(), they set up the coordinate systems into which objects and shaders are placed. Shaders have coordinate systems associated with them in the same way that objects do. By interspersing RenderMan transformation and stack routines between geometric primitive routines, it is possible to control how object space is defined for each object in the scene. The same is true for shader space, based upon where the call that invokes each attribute shader is made.

RenderMan applies given transformations onto objects in reverse sequence. This means that the last transformation concatenated onto the current transformation before an object is defined is the first one that will be applied to the object.

Camera Positioning

In RenderMan, the camera is not specified as a separate object. Instead, transformation routines are called before RiWorldBegin() to define the mapping from world space to camera space. This process is conceptually the same as having a camera that is fixed at the origin, looking down the z axis, and transforming the world, as though it were an object, until it is in proper view.

Light Sources

RenderMan places light sources into the scene as shaders. The positioning of a light source is set through parameters to the light shader associated with the light, and each light is included in the scene with a call to RiLightSource(). Since any point values passed to a light shader are interpreted as being in the current coordinate system, it is also possible to use transformations to control light source positioning.