1 A System Overview

1.1 Introduction

Computer animation takes place in a virtual 3D world. However, it is normally visualized through a 2D screen made up of pixels. The rendering process is the computer animator's camera, which records the virtual world in a format that can be broadcast. Programmers have been developing renderers for many years and a range of techniques have been established. However, these approaches with their respective strengths and weakness have traditionally been considered mutually exclusive – a rendering system based on ray tracing would handle shiny reflective surfaces efficiently, but users would simply have to accept that it could not handle displacement or diffuse surfaces as well as some other architectures could.

While computer graphics researchers have the luxury of being able to work with only certain kinds of scenes, and can explore the limitations and strengths of different approaches to rendering, there is increasing demand placed upon rendering software from the commercial sector. In particular, feature film production requires software that can deliver a wide range of images in a timely manner. Driven by this, modern production rendering systems generally make use of a hybrid approach, incorporating a range of techniques to produce software that is both efficient and flexible.

Many of the requirements of commercial production rendering are embodied in the RenderMan standard (Pixar, 2000). While not all rendering software used in production is based upon this standard, it defines a feature set which is typical of a high-end commercial renderer. It is presupposed that most surfaces will be curved rather than polygonal, ensuring that they appear smooth, even when viewed at the high resolution of film. RenderMan also includes procedural shading, which gives users almost total control over the appearance of surfaces. In order to support these features efficiently in limited memory most implementations of the standard sacrificed ray tracing and global illumination support. However, almost all have recently been converted to a hybrid architecture, effectively consisting of several renderers in one package, each solving one part of the rendering problem.

To support such a broad feature set requires a large number of modules each interacting in a carefully controlled manner with its neighbours. This chapter takes a systems approach to rendering by considering the implementation of a simple production style renderer based on the RenderMan standard. An overview

of each module is developed, and the flow of data through the system is considered. Following chapters will each embellish upon one area of this roadmap, considering its implementation in greater detail.

1.2 Input

Any rendering system must begin with the parsing of a scene description. The RenderMan standard defines both a C API and a RIB (RenderMan interface bytestream) file format, either of which allows a scene to be fed into the renderer. Any renderer that can support these forms of input may be used interchangeably. A modelling package simply generates a RIB, which allows an animator to select any compliant renderer, based on the requirements of the project. It also allows us to develop a renderer, which may be used for "real" work without concern as to how the scene is to be produced.

A complete description of both the C and RIB APIs can be found in the RenderMan Specification (Pixar, 2000).

1.2.1 RIBs

Although the C API and the RIB interface are functionally almost identical, the RIB interface is simpler, as it avoids the lexical complexities of C. In practice, a renderer would be written to implement the C API internally. A RIB parsing layer can read in RIB files and translate them into C API calls.

A simple lexical analyzer (written using lex) can identify keywords, strings and numbers in a RIB file. For each keyword a unique function is called, which uses lex to extract the parameters to that command. These parameters are then repackaged and a call to the appropriate C API function is made. Because there is a direct correspondence between RIB and the C API, the RIB parser needs to know very little about the semantics of the commands it is processing.

The most complex aspect of the C API and the RIB parsing library, is that RenderMan commands can take a variable number of parameters. While in a RIB file these are simply a list of key/value pairs, passing these into a C function is slightly more complex. RenderMan documentation (Upstill, 1990; Apodaca and Gritz, 1999; Stephenson, 2002) typically demonstrates the handling of these parameter lists through the use of *varargs* functions, where the key/value list is terminated by a NULL. For example, the `Surface` command:

```
Surface "plastic" "Ks" [0.6] "Kd" [0.2]
```

would be implemented in C as:

```
RtFloat spec[1] = {0.6};
RtFloat diff[1] = {0.2};
RiSurface("plastic", "Ks", spec,"Kd", diff, RI_NULL);
```