1

INTRODUCTION

DAVID S. EBERT

PROCEDURAL TECHNIQUES AND COMPUTER GRAPHICS

Procedural techniques have been used throughout the history of computer graphics. Many early modeling and texturing techniques included procedural definitions of geometry and surface color. From these early beginnings, procedural techniques have exploded into an important, powerful modeling, texturing, and animation paradigm. During the mid- to late 1980s, procedural techniques for creating realistic textures, such as marble, wood, stone, and other natural material, gained widespread use. These techniques were extended to procedural modeling, including models of water, smoke, steam, fire, planets, and even tribbles. The development of the RenderMan shading language (Pixar 1989) greatly expanded the use of procedural techniques. Currently, most commercial rendering and animation systems even provide a procedural interface. Procedural techniques have become an exciting, vital aspect of creating realistic computer-generated images and animations. As the field continues to evolve, the importance and significance of procedural techniques will continue to grow. There have been two recent important developments for real-time procedural techniques: increased CPU power and powerful, *programmable* graphics processors (GPUs), which are available on affordable PCs and game consoles. This has started an age where we can envision and realize interactive complex procedural models and effects.

What Is a Procedural Technique?

Procedural techniques are code segments or algorithms that specify some characteristic of a computer-generated model or effect. For example, a procedural texture for a marble surface does not use a scanned-in image to define the color values. Instead, it uses algorithms and mathematical functions to determine the color.

The Power of Procedural Techniques

One of the most important features of procedural techniques is *abstraction.* In a procedural approach, rather than explicitly specifying and storing all the complex details of a scene or sequence, we abstract them into a function or an algorithm (i.e., a procedure) and evaluate that procedure when and where needed. We gain a storage savings, as the details are no longer explicitly specified but implicit in the procedure, and the time requirements for specification of details are shifted from the programmer to the computer. This allows us to create inherent multiresolution models and textures that we can evaluate to the resolution needed.

We also gain the power of *parametric control,* allowing us to assign to a parameter a meaningful concept (e.g., a number that makes mountains rougher or smoother). Parametric control also provides amplification of the modeler/animator's efforts: a few parameters yield large amounts of detail; Smith (1984) referred to this as *database amplification.* This parametric control unburdens the user from the low-level control and specification of detail. We also gain the serendipity inherent in procedural techniques: we are often pleasantly surprised by the unexpected behaviors of procedures, particularly stochastic procedures.

Procedural models also offer *flexibility.* The designer of the procedures can capture the essence of the object, phenomenon, or motion without being constrained by the complex laws of physics. Procedural techniques allow the inclusion in the model of any desired amount of physical accuracy. The designer may produce a wide range of effects, from accurately simulating natural laws to purely artistic effects.

PROCEDURAL TECHNIQUES AND ADVANCED GEOMETRIC MODELING

Geometric modeling techniques in computer graphics have evolved significantly as the field matures and attempts to portray increasingly complex models and the complexities of nature. Earlier geometric models, such as polygonal models, patches, points, and lines, are insufficient to represent this increased complexity in a manageable and controllable fashion. Higher-level modeling techniques have been developed to provide an abstraction of the model, encode classes of objects, and allow high-level control and specification of the models. Many of these advanced geometric modeling techniques are inherently procedural. Grammar-based models (Smith 1984; Prusinkiewicz and Lindenmayer 1990), including graftals and L-systems, allow the specification of a few parameters to simulate complex models of trees, plants, and other natural objects. These models use formal languages to specify complex growth rules for the natural objects.

Implicit surface models—also called blobby molecules (Blinn 1982b), meta-balls (Nishimura et al. 1985), and soft objects (Wyvill, McPheeters, and Wyvill 1986)—are used in modeling organic shapes, complex manufactured shapes, and "soft" objects that are difficult to animate and describe using more traditional techniques. Implicit surfaces were first introduced into computer graphics by Blinn (1982b) to produce images of electron density clouds. They are surfaces of constant value, *isosurfaces,* created from blending primitives (functions or skeletal elements) represented by implicit equations of the form $F(x, y, z) = 0$. Implicit surfaces are a more concise representation than parametric surfaces and provide greater flexibility in modeling and animating soft objects. For modeling complex shapes, several basic implicit surface primitives are smoothly blended to produce the final shape. The detailed geometric shape of the implicit surface is not specified by the modeler/animator; instead, it is procedurally determined through the evaluation of the implicit functions, providing higher-level specification and animation control of complex models.

Particle systems are procedural in their abstraction of specification of the object and control of its animation (Reeves 1983). A particle system object is represented by a large collection (cloud) of very simple geometric particles that change stochastically over time. Therefore, particle systems do use a large database of geometric primitives to represent natural objects ("fuzzy objects"), but the animation, location, birth, and death of the particles representing the object are controlled algorithmically. As with other procedural modeling techniques, particle systems have the advantage of database amplification, allowing the modeler/animator to specify and control this extremely large cloud of geometric particles with only a few parameters. Particle systems are described in more detail in Chapter 8.

These advanced geometric modeling techniques are not the focus of this book. However, they may be combined with the techniques described in this book to exploit their procedural characteristics and evolve better modeling and animation techniques.

Additionally, some aspects of image synthesis are by nature procedural; that is, they can't practically be evaluated in advance (e.g., view-dependent specular shading and atmospheric effects). Our primary focus is *procedural textures, procedural modeling,* and *procedural animation.*

AIM OF THIS BOOK

This book will give you a working knowledge of several procedural texturing, modeling, and animation techniques, including two-dimensional texturing, solid texturing, hypertextures, volumetric procedural models, fractal and genetic algorithms, and virtual procedural actors. We provide you with the details of these techniques, which are often omitted from technical papers, including useful and practical guidelines for selecting parameter values.

We provide a toolbox of specific procedures and basic primitive functions (noise, turbulence, etc.) to produce realistic images. An in-depth description of noise functions is presented, accompanied by several implementations and a spectral comparison of these functions. Some of the procedures presented can be used to create realistic images of marble, brick, fire, steam, smoke, water, clouds, stone, and planets.

ORGANIZATION

This book follows a progression in the use of procedural techniques: from procedural texturing, to volumetric procedural objects, and finally to fractals. In each chapter, the concepts are illustrated with a large number of example procedures. These procedures are presented in C code segments or in the RenderMan shading language.

The details of the design of these procedures are also explained to aid you in gaining insights into the different procedural design approaches taken by the authors. You should, therefore, be able to reproduce the images presented in this book and extend the procedures presented to produce a myriad of effects.

Darwyn Peachey describes how to build procedural textures in Chapter 2. This discussion includes two-dimensional texturing and solid texturing. Two important procedures that are used throughout the book are described in this chapter: *noise* and *turbulence*. Aliasing problems of procedural techniques are described, and several antialiasing techniques suitable for procedural approaches are presented and compared.

Real-time issues for procedural texturing and shading are described by William R. Mark in Chapter 3. This chapter also provides an overview of the Stanford real-time shading language, a high-level interface for writing real-time shaders.

The design of procedural textures is also the subject of Chapters 4, 5, and 6 by Steve Worley. Chapter 4 contains useful guidelines and tricks to produce interesting textures efficiently. Chapter 5 describes some additional approaches to antialiasing, and Chapter 6 describes cellular texturing for generating interesting procedural textures.

The next four chapters by David Ebert describe how turbulence and solid texturing can be extended to produce images and animations of three-dimensional gases (particulate volumes). Chapter 7 concentrates on techniques to produce still images of volumetric “gases” (steam, fog, smoke). Chapter 8 discusses how to animate these three-dimensional volumes, as well as solid textures and hypertextures. Chapter 9 describes how to extend these volumetric procedural models to model and animate realistic clouds, and Chapter 10 discusses more real-time issues for hardware-accelerated implementation of procedural textures and models.

Chapters 11 and 13 by John Hart discuss alternative approaches for procedural models and textures. Chapter 11 concentrates on procedural models of geometry, including procedural geometric instancing and grammar-based modeling. Chapter 13 concentrates on procedural textures using the texture atlas approach.

Chapter 12 by Ken Perlin discusses other types of volumetric procedural objects, such as hypertextures and surflets. Ken also gives a detailed explanation of his famous *noise* function and its implementation, as well as the rendering of procedural objects and antialiasing. Ken also describes the use of high-level, nondeterministic “texture” to create gestural motions and facial animation for synthetic actors. By blending textural controls, the apparent emotional state of a synthetic actor can be created.

Chapters 14 through 20 by Ken Musgrave (Chapter 18 is also co-authored by Larry Gritz and Steve Worley) describe fractals and their use in creating realistic landscapes, planets, and atmospherics. Ken begins by giving a brief introduction to fractals and then discusses fractal textures, landscapes, and planets. The discussion proceeds to procedural rendering techniques, genetic procedural textures, and atmospheric models.

Finally, Chapter 21 by Ken Musgrave discusses the role of procedural techniques in the human-computer interface.