Solid Spaces: A Unified Approach to Describing Object Attributes

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the Graduate

School of The Ohio State University

By

David S. Ebert, B.S, M.S.

* * * * *

The Ohio State University

1996

Dissertation Committee:

Prof. Richard E. Parent

Prof. Wayne E. Carlson

Prof. Mukesh Singhal

Approved by

Adviser
Department of Computer
and Information Science

To my family and friends who have helped me accomplish this work.

Acknowledgments

This work could not have been accomplished without the help of many people. The author wishes to thank Keith Boyer for design help and writing the network distribution software, Jeff Ely for mathematical problem solving, Julia Ebert for reviewing this dissertation and general support, Steve May for generating the teacup data, and the computer staff of the Department of Computer and Information Science for providing a stable computer environment with high availability. The author also wishes to thank Hewlett-Packard for their donation of equipment to the graphics lab. I am also indebted to Bob Manson for amazing software support. Bob developed the animation language used to produce my animations and provided other software help to numerous to mention. Rob Rosenblum, Jeff Ely, and Keith Boyer helped in optimizing the turbulence functions. Jim Kent supplied the Slab intersection C Code. Rob and Jim both developed "anictl" (an animation playback systems) which made my life much easier. Pat Wilcox, J. Ramanujam, and Con O'Connell worked on the dissertation style so that it met the Graduate School requirements.

I owe a special debt to Keith Boyer and Julie Ebert for supporting me and helping me accomplish this dissertation. Amazingly they both are still willing to work with me even after completing "Getting Into Art."

Several graduates students made helped make my life as a graduate student memorable and sometimes even enjoyable: Keith Fish, Rob Rosenblum, Doug Roble, Tony Leclerc, Mike Fortin, Steve May, Dan Waylonis, Catherine Stringfellow, Joe Ault, Brian Guenter, Kevin Rogers, Jim Kent,John Chadwick, and Scott Whitman.

I also am deeply indebted to the faculty with whom I have worked, especially Rick Parent and Wayne Carlson. Their guidance and perseverance helped me accomplish this work. They were more than mere academic advisors; they were also good friends. Ed Tripp, Ray Moore, Mukesh Singhal, and Merv Muller also provided me with great support. My parents and family also provided me with exceptional support and encouragement throughout this process.

Finally,this research is built upon the EDGE graphics software library developed at the Department of Computer and Information Science, The Ohio State University [9].

Vita

April 2, 1964	Born - Sandusky, Ohio
1986	B.S. Computer and Infomation Science The Ohio State University Columbus, Ohio
1987-Present	Graduate Teaching Associate Computer Science Department The Ohio State University Columbus, Ohio
1987	M.S. Computer Science The Ohio State University Columbus, Ohio

Publications

Ebert, David, and Parent, Richard. Animation of Gaseous Phenomena Using Turbulent Flow Based Solid Texturing. Tech. Rep. OSU-CISRC-7/89-TR36, Department of Computer and Information Science, The Ohio State University, 2036 Neil Ave, Columbus, Ohio 43210-1277, July 1989.

Ebert, David, Boyer, Keith, and Roble, Doug. Once a Pawn a Foggy Knight ... [videotape]. In *SIGGRAPH Video Review 54* (November 1989), ACM SIGGRAPH, New York. segment 3.

Carlson, Wayne, Ebert, David, Parent, Richard, and Boyer, Keith. EDGE - Educational and Developmental Graphics Environment. Tech. Rep. OSU-CISRC-2/90-TR3, Department of Computer and Information Science, The Ohio State University, 2036 Neil Ave, Columbus, Ohio 43210-1277, February 1990.

EBERT, DAVID, EBERT, JULIA, AND BOYER, KEITH. Getting Into Art. [videotape], Department of Computer and Information Science, The Ohio State University, May 1990.

EBERT, DAVID, AND PARENT, RICHARD. Rendering and Animation of Gaseous Phenomena by Combining Fast Volume and Scanline A-buffer Techniques. Proceedings of SIGGRAPH'90,(Dallas, Texas, Aug 6-10,1990). In *Computer Graphics 24,4* (August 1990), 357–366.

Fields of Study

Major Field: Computer and Information Science

Studies in:

Computer Graphics and Animation:	Prof. Richard E. Parent
Distributed Systems:	Prof. Mukesh Singhal
Numerical Analysis:	Prof. Ramon Moore

Solid Spaces: A Unified Approach to Describing Object Attributes

By

David S. Ebert, Ph.D.

The Ohio State University, 1996

Prof. Richard E. Parent, Adviser

Realistic rendering and animation are important problems in computer graphics. Many previous systems for realistic rendering and animation have only addressed certain issues in realism. This dissertation proposes a sytem that unifies many previous attempts at producing realistic images and animations into a cohesive, extendable system. This system also extends the state of the art in producing realistic images and animations. This system is built upon the concept of *solid spaces*. Solid spaces are three-dimensional spaces that represent attributes of an object, ranging from the object's geometry to the object's color, and even to the motion of the object. A unified solid space data structure is presented which allows the storage of all of the object's attributes in this one simple structure. Unified solid space access functions are also presented which allow the access of all of the object's attributes in a unified manner. The rendering system that supports solid spaces is also described. The

power of solid spaces for controlling the attributes of objects is shown for gaseous objects. The geometry, shadowing, and motion of the gases are all represented by solid spaces. These examples show how the state of the art in producing realistic images and animations has been extended by the the work described in this dissertation.

Table of Contents

List of Tables

List of Figures

CHAPTER I

Introduction

Throughout the history of computer graphics, advances have been driven by the quest for visual realism. This quest for visual realism encompassed all aspects of image creation from object definition to object color, illumination, and shadowing. For example, early models for representing objects used polygonal meshes. These early polygonal models for representing objects were not used to represent abstract artistic shapes; they were models of actual objects. Spline patch models were later used for object modeling to more accurately represent curved surfaces in the real world. Early rendering systems rendered faceted shaded, flatly illuminated, alias-prone images. Techniques were then developed by Gouraud and Phong to approximate curved surfaces from polygonal models [52]. The illumination models for these objects have also greatly improved from simple Lambertian models to complex illumination models using radiosity, ray tracing, and Cook-Torrance illumination models.

This dissertation is an attempt to further the capability of computer graphics systems to produce realistic images and animations. This work combines much recent work in producing state-of-the-art images, such as texturing, geometric modeling, and animation. Three-dimensional solid texturing, commonly used to produce accurate image of objects carved from wood, marble, granite, and other stone materials, is just one technique for realistic rendering that is incorporated into a cohesive system in the work presented here. Many techniques for simulating complex geometry, including modeling natural phenomena, are also incorporated in this research. Even several animation techniques are incorporated into the unified framework described in this dissertation. Therefore, the work presented here provides a unified framework which incorporates many recent advances in rendering and animating realistic images.

The rest of this introductory chapter will review advances in creating state-of-the-art images. The use of texture mapping techniques for modeling the surface attributes of objects will be discussed, followed by a discussion of object modeling, and concluding with a discussion of animation control issues.

1.1 Texture Mapping Techniques

Texture mapping is a technique for simulating the surface characteristics of an object. This technique was originally proposed by Catmull in 1974 [52] and is still an important technique for creating realistic images. Texture mapping varies the surface characteristics of an object through the use of mathematical functions, or two-dimensional or three-dimensional tables. This technique has been used to modulate many surface characteristics, including color, roughness, reflection, transparency, and even the actual surface geometry (displacement mapping).

Texturing is commonly used in applying a two-dimensional image onto an object to produce a color pattern on the object. Examples of this include creating a label on a wine bottle, the woven fabric color pattern on a sofa, and the red letters on an "EXIT" sign.

This technique is also commonly used to simulate bumps, wrinkles, and imperfections on the surface of objects by modifying the normal to the surface of the object. This helps in creating realistic images by reducing the smooth "antiseptic" quality of computer generated images. Scratches in a table, the wrinkles on the surface of an orange, and the bumps in a stucco wall can all be simulated with two-dimensional texturing of surface normals (bump mapping).

To model the actual surface geometry created by bumps, dents, and scratches, displacement mapping can be used. Displacement mapping differs from bump mapping in that the actual geometry of the surface is modulated (or displaced) instead of just the normal to the surface. Displacement mapping solves two problems that occur with bump mapping: smooth silhouettes and smooth intersections. In bump mapping, the silhouette of the object will not show the effects of the bump mapping, since the normal vector is just perturbed. The actual geometry of the silhouette (and the entire object) remains unchanged. The intersection of two bump mapped objects will also not show the effects of the bump mapping for the same reason. However, with displacement mapping, the geometry of the object is actually changed, so these problems will not occur.

Displacement mapping and other types of texture mapping are incorporated as follows into the rendering process. Texture mapping normally occurs just before the illumination calculations are performed to determine the final color of the surface element. Texture mapping of surface color, surface normal vector, surface reflection, and other surface illumination parameters is performed just prior to the illumination calculation to determine the final color of this surface element. Texture mapping of surface transparency normally occurs just prior to the combination of this surface element with the surface element behind it. In most rendering systems, displacement mapping needs to be performed before the hidden surface algorithm and actually changes the geometric representation of the object. In most rendering algorithms, texturing is only applied to the visible points on the surface of the object (after the hidden surface calculations). To avoid perspective distortions in the texture, the object space location of the visible point on the surface of the object is usually used in the texture mapping calculation. Different mapping algorithms will be discussed in the following subsections.

1.1.1 Two-dimensional texturing

Texture mapping was originally a two-dimensional technique. In two-dimensional texture mapping, each visible point p on an object is mapped into a two-dimensional space (normally a two-dimensional table). The two-dimensional texture space is then evaluated to determine the value associated with the particular location in two-space. This value is then used for the particular surface characteristic of the point p on the object. Two-dimensional texturing was originally used in modeling the surface color of objects [10]. Blinn and Newell extended texture mapping to modulate the specular and reflection characteristics of objects in 1976 [7]. Blinn further extended texturing to modulate the normal vector of surfaces as a way of simulating bumps and wrinkles on objects [5]. Finally, Cook suggested using two-dimensional texturing to manipulate all aspects of the illumination of an object, including surface displacement and shadowing [14].

There are several techniques for mapping the point p into this two-dimensional texture space. For a survey of different mapping techniques, see [31]. These techniques can be classified into two different classes of mapping techniques. The first class maps each polygon or patch of the object into the entire two-dimensional space. Therefore, using this class of mapping techniques repeats the texture on each polygon or patch of the object. Within this class of mappings, two different mapping techniques are commonly used. The first mapping technique is referred to as inverse bilinear interpolation [31]. This technique first uses the inverse perspective transformation to transform the image space location of the point p on the surface of the object into object space. Then, the point p in object space is mapped into the texture definition space through the use of inverse bilinear interpolation. The second technique uses simple bilinear interpolation in screen space, where the location of the point p in screen space is used in the texture mapping, and not the location of p in object space, as in the previous technique. This technique, however, suffers from perspective distortion. A more detailed description of these techniques can be found in [31].

The second class of mapping techniques maps the entire surface of the object into the texture space. This class of mappings "wraps" the texture around the entire object, instead of repeating the texture on each polygon or patch. The simplest technique of this class uses a spherical mapping. With spherical mapping, the polar coordinates of the point p, with respect to the object coordinate system, are used to map the surface of the object into the texture space. Simple linear interpolation in screen space can also be used to map the entire object surface into the texture space. In this technique, the user assigns texture space coordinates with each vertex of a polygon (or control point of a patch). Then, during scan conversion, these texture space coordinates are simply linearly interpolated to determine the texture space coordinate for each visible point p on the surface of the object. Object unfolding and two-part texture mapping, which are additional mappings in this class, are described below.

The mapping of each point on the surface of a three-dimensional object into the two-dimensional texture space creates many problems in the resultant appearance of the texture applied to the object. The two-dimensional texture may suffer distortions when applied to the three-dimensional object. For, example, when a simple spherical projection is used to map a two-dimensional texture onto an object, the texture is normally compressed near the poles and stretched near the equator of the object. Another problem caused by the texture mapping is discontinuities at the seams of the texture applied to the object. A seam is where two separate sides of the texture map meet when the texture is applied to the object. Discontinuities can occur at the seams for two reasons. First, the values in the texture map at the two edges that meet may not be the same. Second, the scale can change abruptly at the seams because of the mapping technique. The surface area that each section of the texture map occupies on opposite sides of the seam may be different because of the mapping. Blurring or averaging of values at the seams can be used to solve this problem [8].

A few techniques have also been proposed to solve the distortion problem. The first technique is referred to as two-part texture mapping[4]. In this technique, the texture is first projected onto an intermediate three-dimensional shape. Then the texture is mapped from the intermediate three-dimensional shape onto the final object. The choice of the separate parts of the mapping allows for choosing a combination of techniques that minimize distortion for the particular object.

Another solution can be termed object unfolding [54]. In this technique, the polygons of the object are unfolded onto a a flat two-dimensional plane. Then vertices in the object are associated with locations in the texture map. The user interactively assigns a texture space coordinate with each vertex in the unfolded object. The location of the vertices in the texture space are then used during the mapping to apply the texture to the object. Problems with this technique include the possibility of introducing discontinuities into the texture applied to the image and loss of texture when applied to the object (parts of the texture map may not be applied to the object). This technique is used to reduce distortion and scale changes when the texture map is applied to the object. It still does not solve the problem with seams.

1.1.2 Three-dimensional texturing - solid texturing

In 1985, Gardner [24], Peachey [44], and Perlin [46] all independently suggested extending two-dimensional texturing to three-dimensional texturing or solid texturing. Solid texturing differs from two-dimensional texturing in that solid texturing maps each point on the surface of an object to a three-dimensional texture space as opposed to a two-dimensional texture space. The location of the point in this three-dimensional "solid space" is used in calculating the value for modulating the surface characteristic of the point on the object.

Solid texturing is incorporated into the viewing algorithm in a manner similar to two-dimensional texture mapping. As in two-dimensional texturing, the three-dimensional screen space location of each visible point p on the surface of the object is mapped back to object space. The main difference is that then this three-dimensional object space location is used by the solid texturing functions to determine the value for the corresponding screen location, as opposed to the two-dimensional location used in two-dimensional texturing.

Gardner [24] uses three-dimensional texturing for simulating clouds, which will be discussed later in this chapter. This technique is used as a model for clouds by using Fourier synthesis to control the transparency of hollow ellipsoids.

Peachey [44] and Perlin [46] both proposed solid texturing for controlling the color patterns of objects. They both also use solid texturing as a model for simulating objects carved from solid materials. In solid texturing, the texture is determined from evaluating three-dimensional functions based on the location of each point on the surface of an object in the solid texture space. Therefore, solid texturing is an extremely powerful technique for simulating objects carved from solid materials such as wood, marble, granite, and other stone materials.

Peachey [44] uses both three-dimensional functions and projected two-dimensional images for defining his three-dimensional textures. Peachey has suggested several functions which can be used to create interesting three-dimensional textures. By randomly placing spheres of random size throughout a solid space, he simulates materials in which bubbles of one material are captured in the solidification of another material. To simulate wood, Peachey uses concentric cylinders of light and dark colors aligned along an arbitrary axis in three-space. Peachey simulates marble through placing starting locations for veins within the solid space. The direction of the veins is then controlled through the use of sinusoidal functions. The diameter of the cross-section of the veins is also controlled by more sinusoidal functions.

Perlin [46] also chose three-dimensional functions to define his three-dimensional textures. Perlin makes extensive use of function composition to create interesting three-dimensional textures. The basis of most of these functions are two functions, one of which simulates random noise and the other which provides a "visual" simulation of turbulent flow. Through the use of these functions Perlin is able to create very realistic images of marble, fire, water, and block glass (see [46]).

Solid texturing solves some problems of two-dimensional texturing. Since solid texturing uses an affine mapping from the three-dimensional object space to the three-dimensional texture space (normally scale and translation transformations), there is no distortion of the texture when it is to the object. In two-dimensional texturing where a two-dimensional image is mapped onto a complex three-dimensional manifold, distortion and warping of the texture applied to the object can occur. Solid texturing avoids these problems.

1.2 Simulating Complex Geometry

There have been many approaches to modeling the complex geometry of objects in our environment. Polygonal and patch models were the first models used. Constructive solid geometry (CSG) models were introduced later. More recent models can be categorized as either particle systems or procedural models.

1.2.1 Polygonal and patch modeling

The first geometric models of objects in computer graphics were polygonal meshes. A polygonal mesh is a collection of planar polygons containing vertex, edge, and connectivity information. Polygonal mesh models, however, do not accurately model smooth surfaces which are so commonly found in our environment. To circumvent this problem, Gouraud [52] developed an intensity interpolation method for polygonal mesh objects to simulate the appearance of a curved surface. Phong [52] developed an improved approximation method which uses interpolation of the normal vector to approximate the illumination from a curved surface. Both of these techniques, however, suffer problems since they do not correctly model the geometry of a curved surface. For example, the silhouette of the object is still polygonal and so is the line of intersection of two polygonal objects.

Spline surface patches solve some of the problems with polygonal patch models, since they are an actual curved surface model useful for simulating smooth surfaces in our environment. Coons [40], Bezier [40], and others have developed surface patch models. Surface patch models have been extended to formulations that are easier to control by Barsky [2] and to other formulations that have advantages over earlier formulations, such as hierarchical spline models. Forsey,[21], has recently created a hierarchical spline model that allows for uneven spacing of control points to provide an efficient formalization for objects that have areas of varying degrees of surface detail.

Both polygonal and surface patch models are usually created by digitizing a real world model, or using an interactive system to create the model.

1.2.2 Constructive solid geometry models

Polygonal and surface patch models are boundary representations of objects [40]. The polygons and patches define the surface of the object. They separate points inside the object from points outside the object. The geometry of the solid interior of the object is not defined (the objects are hollow). Constructive solid geometry (CSG)[40], on the other hand, is a solid model for objects. CSG objects are solid objects, not simply surfaces. The interior geometry of the object is part of the CSG representation of the object. CSG models use Boolean operations on simple shapes to define complex shapes. CSG models have several advantages over boundary representations. First, they are a very compact way of representing complex shapes and can be generated rapidly by solid modeling systems. Second, they contain the full three-dimensional volume geometry of the object, as opposed to just the surface information. CSG models, however, do have some disadvantages when compared to boundary representations. First, they either require a volume rendering or ray tracing system to render the full the three-dimensional volume of the object or they require the boundary representation to be calculated for use in a surface-based renderer. Calculating the boundary representation from the CSG model requires extensive computations. Second, some shapes are not easily described as Boolean operations on simple shapes.

1.2.3 Particle systems

A more recent approach to modeling complex geometric objects is particle systems. Reeves [49] originated the use of particle systems. Particle systems have mainly been used for modeling natural phenomena, such as fire, trees, and water where the intricate detail of the phenomena is represented by a large collection of particles. Particle systems normally involve the use of a large number of small spherical particles. The animation of these particles is controlled through the use of procedures which simulate the specific natural object, such as fire. The rendering of particle systems normally uses a simple constant shading model and often the color of the particle is determined by its three-dimensional location in space. For example, in Reeves' fire simulation, [49], the color of the particle is determined by its elevation from the base of the fire.

Particle systems have several advantages over patch and polygonal models. Particle systems normally use simple spheres for representing the geometry of the object or phenomenon, instead of complex patch or polygonal models. Therefore, the rendering system only needs to handle simple spherical models. The particle system rendering process also usually uses constant flat shading for the illumination of the particles, which is much simpler and quicker than normal illumination algorithms used for patch and polygonal models.

However, particle systems do have several disadvantages. They require an extremely large geometric database of particles to represent complex objects or phenomena. Shadowing algorithms and illumination algorithms for particle systems, although quicker, are not as realistic as the algorithms normally used for patch and polygonal models. Also, complex shapes can only be approximated by a large collection of spheres; therefore, sharp edges are hard to simulate.

1.2.4 Procedural modeling

Many different modeling techniques can be termed procedural modeling. Fractal synthesis [36], Fourier synthesis [24], hypertextures [47], L-systems [48], volume density functions,[20], and inverse particle systems, described later in this dissertation, can all be considered to be procedural models. Procedural modeling uses algorithms to represent the geometry of objects. Most physically-based modeling techniques are also procedural modeling techniques, for example Kajiya's cloud modeling technique [32].

Procedural modeling techniques have many advantages over polygonal and patch modeling techniques. The procedural model is evaluated during the rendering process to determine the geometry of the object/ phenomenon. Usually, the model is evaluated at the resolution of image rendering. Therefore, the models provide the needed amount of detail without introducing high frequency details which will result in aliasing artifacts. Secondly, complex shapes can be represented with very little data storage space. For instance, a fractal mountain can be represented with a procedure of less than 50 lines of C language code; whereas, to achieve an equivalently detailed polygonal model might require 50,000 polygons. Because of this, procedural modeling techniques are often used to represent objects of very high degrees of detail such as natural phenomena.

Procedural models, however, do have disadvantages over particle systems, patch, and polygonal modeling. The main disadvantage is the computation time required to evaluate the procedure. Often the procedure is evaluated during rendering and many floating point computations are usually performed during each evaluation of the procedural model. The computation time of rendering a procedural model is directly related to the computation time to evaluate the procedure for the model.

1.3 Modeling Natural Phenomena

Natural phenomena are some of the hardest things to model in computer graphics. Modeling phenomena such as mountains, trees, plants, fire, water, clouds, and smoke have inspired much research in computer graphics. Realistic models of these phenomena, however, still elude current computer graphics techniques. The complex intricate geometry and motion of these phenomena make them very difficult to model. Some natural phenomena, such as mountains, trees, and plants have intricate rigid shapes, while others, such as clouds, fire, and water have intricate amorphous shapes. Most models for natural phenomena can be classified as fractals, partical systems, or other types of procedural models.

1.3.1 Modeling terrain and vegetation

Most approaches to modeling terrain have used a fractal modeling approach, [39]. Mandelbrot and Musgrave [41] have done much work on the modeling of terrain using fractals. Recent work has included modeling terrain with fractals and then simulating the natural erosion processes that affect terrain to create a more realistic model [41].

There has been a wider range of techniques used to model vegetation. Iterated function systems,[15], fractals, particle systems [50], and L-systems,[48], have all been used to model plants and trees. Very realistic images of plants have recently been produced through the use of L-systems [48].

1.3.2 Modeling fire and water

The modeling of water has received as much attention as the modeling of terrain and vegetation. Early models for water used simple cycloidal models for controlling the height of the surface of the water [37]. This model had many problems, including its inability to simulate breaking or curling waves since there can only be one height value for each x-y location. 1986 and 1987 saw a flurry of research in modeling ocean waves [56, 22, 45]. Some of these models provided more physically based models that simulated breaking waves and even spray from breaking waves created by particle systems [22]. More recently, Kass and Miller,[33], have developed a model based on wave equations that even allows for the net transport of water volume. The main drawback with their approach is that they again use a height field, so they can not simulate cresting or breaking waves.

The modeling of fire has received very little attention compared to the modeling of water. The main approach to modeling fire has been the use of particle systems [49, 55].

1.3.3 Modeling and animating gaseous phenomena

The rendering of scenes containing clouds, fog, atmospheric dispersion effects, and other gaseous phenomena has received much attention in the computer graphics literature. Several papers deal mainly with atmospheric dispersion effects [58, 43, 53], while many cover the illumination of these gaseous phenomena in detail [6, 32, 38, 34]. Most authors have used a low albedo reflection model, while a few, Blinn [6], Kajiya [32], and Rushmeier [53], discuss the implementation of a high albedo model. A low albedo reflectance model assumes that secondary scattering effects are negligible, while a high albedo illumination model calculates the secondary and higher order scattering effects.

Another issue is modeling the geometry of these gases. Some authors use a constant density medium [34, 43], but do allow different layers of constant densities. This allows for a very limited geometry for the gases. Voss uses fractals [57] and Max uses height fields [38] for modeling the geometry of clouds. Kajiya uses a physically-based model for clouds [32], which simulates water vapor, heat flow, wind, etc. to form a physical model for clouds. However, the resulting images are not very realistic. Gardner uses hollow ellipsoids [24] to model the geometry of clouds. He controls the transparency of hollow ellipsoids through the use of Fourier synthesis. To form larger cloud formations, he combines many different ellipsoids to form cloud groups. The main problem with his approach is that it is not a true three-dimensional model for the clouds, so accurate cloud shadowing is impossible. Another problem is that once you enter a cloud, you can clearly see that it is a simple hollow object and not a full three-dimensional cloud volume.

The author uses turbulent flow based functions to model the density of a variety of gases [20]. These functions are based on Perlin's visual simulation of turbulent flow [46] and are similar to the idea of hypertextures [47]. This model is a true three-dimensional model for the geometry of gases and provides realistic results. This technique seems to provide more realistic results than most previous efforts by providing visually realistic renderings and animations of gaseous phenomena and the shadows they cast. These techniques are based on a visual simulation of turbulent flow, so it is a visual simulation of the turbulent processes that determines the geometry of gaseous phenomena. These techniques can also be extended to use a physically-based turbulent flow model and can be very efficient when simplifying assumptions are made. This approach will be discussed in more detail later in this dissertation.

1.4 Animation Control Approaches

There have been many approaches to controlling the animation of objects. These approaches can be categorized along a continuum from low-level control to high-level control approaches. Low level animation control systems require the animator to specify very detailed information to control the animation. For example, in animating an articulated human figure, the animator may be required to specify every joint position and angle for many frames in the animation of figure. In a high-level control system, the animator specifies the control information at a higher, less detailed level. In the above example with a high-level control system, the animator may simply be able to specify that the figure should walk at a speed of 3 miles per hour in the north direction. The work related to animation control presented in this dissertation can be considered to be a high-level animation control system.

1.4.1 Low-level animation control

Many early approaches to realistic animation control can be considered low-level animation control systems. Scripted systems based on rigid body affine transformations, key-frame systems [35], track-based systems [27], and forward kinematics based systems, are all low-level animation control systems. Forward kinematics based systems for articulated figure animation requires the animator to specify joint angles and translations to produce the animations [26]. Inverse kinematics systems improved on this by allowing the animator to specify the start and end location of the affector and the system would calculate the necessary joint rotations and translations necessary to achieve these results. Therefore, inverse-kinematics is a slightly higher-level control systems than forward kinematics. Dynamics systems are another low-level control approach. In a dynamics system, the animator is required to specify torques and forces acting on the body to get the desired effects [1].

1.4.2 High-level animation control

Some high level control systems are enhancements to the low-level control systems mentioned above, while others are separate procedural modeling systems.

Most recent work in character animation systems [60, 26, 11] have developed high level control based on kinematics, inverse-kinematics, inverse-dynamics, or a combination of these techniques. High-level control systems allow the animator to specify “what” is to be done, instead of “how” to do it. The underlying system then figures out “how” to do it. This greatly reduces the amount of work an animator must perform. He no longer has to be concerned with low-level control issues (“how” to do something).

For example, in the PODA system [26], the animator can use a library of pre-defined gates, or can simply create a gate by specifying the location of the feet at the start, midpoint, and end of the gate as well as the timings of these positions. Once the gate is specified, the user can simply instruct the creature to walk from one location to the other.

Another high-level control animation system is "teleological" modeling proposed by Barr [3]. In this system for example, a user can specify that a linked chain fixed at one end connects the free end in a natural way to another location in space. This system then uses an inverse-dynamics approach to calculate the forces and torques necessary to achieve the desired task.

Procedural animation control systems

Procedural animation control systems use algorithms or procedures to control the animation. Some procedural animation control systems are concerned with physical-based or behavioral-based animation [30, 29, 28], such as the animation of rigid bodies, elastic and deformable bodies, and facial animation. Most procedural animation control systems are used for animating natural phenomena [49, 32, 24, 48, 22, 45, 33, 55] or complex behavioral patterns [51]. The complex behavior patterns normally deal with the animations of groups of objects, such as flocks of birds, schools of fish, etc.

Procedural animation control systems are the most prevalent technique for animating natural phenomena such as fire, water, and gases [49, 32, 24, 48, 22, 45, 33, 55]. These systems use procedures that allow for an algorithmic simulation of the phenomena, sometimes based on physical forces. These systems normally allow the animator to control the animation by specifying a few parameters. Systems for the simulation of ocean waves usually only require the animator to specify the geometry of the ocean floor and the initial direction and speed of the waves [45].

In the gas animation system developed by the author, the user can specify a varying amount of information to produce the gas animation. The minimal information the user specifies is the general gas direction and speed. The system then calculates the entire animation of the gas. The user simply specifies the overall motion of the gas and the system calculates the detailed motion for each individual gas element. By adding a specification of the location and shape of objects in the scene, the system will create gas flow patterns for the flow of the gas around a symmetric boundary representation of the objects in the scene. The system also allows the user to specify, at varying resolutions, direction vectors or procedures to control the gas motion for specific volumes in three-space. The details of this system will be described later in this dissertation.

CHAPTER II

Modeling and Animating Gases Using Solid Spaces

This chapter shows how the unification of solid spaces can be used to further extend the state of the art of computer graphics techniques. This chapter describes the use of unified solid spaces for extending the modeling, rendering, and animation techniques for gaseous phenomena. The use of solid geometry spaces to model gaseous phenomena and fluids is presented in this chapter, as well as the use of solid animation spaces for the animation of these volume objects. The discussion will focus on the use of these solid spaces for modeling, rendering and animating gaseous phenomena, although these techniques can also be used to model, render, and animate fluids. The only change needed is to use a phong illumination model and the rendering algorithm for solid volumes described in Chapter 3 instead of the gaseous illumination model and the rendering algorithm for gases also described in Chapter 3.

This chapter discusses the use of the solid geometry spaces to model the geometry (densities) of gaseous phenomena. The use of solid animation spaces is also demonstrated for controlling the animation of the gases. These solid animation spaces are shown to provide an easy and powerful tool for controlling the animation of gases.

Gases such as fog, steam, and smoke are often essential elements in the creation of realistic images. The realism and mood of outdoor scenes can be greatly increased by the addition of realistic clouds and fog. Surprisingly, the animation of gaseous phenomena has received little attention in the computer graphics literature. As mentioned in the introduction, Kajiya [32] has used a simple physical approximation for the formation and animation of clouds. Gardner [25] has recently produced animations of smoke rising from a forest fire. The author has previously used a very simple model for gaseous animations of fog and steam [20, 18, 17].

One explanation for the lack of gaseous animation in computer graphics is the sheer complexity of the problem. Computational fluid and gas dynamics is an area of very active research with many unsolved problems. Most problems can be represented using the Navier-Stokes equations, which are the equations of motion for Newtonian fluids [23] (a gas is type of fluid). These equations actually describe the conservation of momentum in a fluid. When combined with the equations for the conservation of mass, they provide a closed set of equations for fluid motion. However, the Navier-Stokes equations are non-linear partial differential equations. Therefore, they can be solved analytically for only a few "idealized" boundary conditions. Numerical approximation methods must be used to solve these general equation. Normally, simplifying assumptions are also used to eliminate some of the terms from the equation. Most solution methods use finite element or finite difference methods to solve these equations. This requires calculations using a very large three-dimensional grid, which takes large numbers of super-computer hours and creates large amounts of data (gigabytes)[59].

Even the turbulent flow, with an infinite Reynolds number (ratio of inertial forces to viscous forces), about a circular cylinder cannot be accurately computed today [13]. For example, current gas dynamics research is still being performed concerning the two-dimensional turbulent flow of gas about a circular cylinder for non-zero Mach numbers (for the speed of the gas) [59]. Gas flow in a sparsely furnished room also cannot be accurately modeled with current gas dynamics models [42]. As you can see, physically-based numerical simulation of gas flow is an unsolved problem and an area of much current research. Therefore, for the animations of gases, a reasonable visual simulation is a valuable alternative even though it is not valid from a physical simulation sense. This is the approach described in this dissertation.

The techniques described in this dissertation seem to provide more realistic results than most previous efforts by providing visually realistic renderings and animations of gaseous phenomena and the shadows they cast. These techniques are based on a visual simulation of turbulent flow, so it is a visual simulation of the turbulent processes that determines the geometry of gaseous phenomena. These techniques can also be extended to use a physically-based turbulent flow model and can be very efficient when simplifying assumptions are made.

This dissertation also presents a flexible method for animating gaseous phenomena and fluids. The method described is flexible enough to be used as a visualization tool for gas dynamicists as well as simple enough to allow easily choreographed animation control of the gas movement. The method also allows simple physically-based approximations to gas avoidance of objects as well as the ability to combine scientific flow data and choreographed animation control.

The system presented accomplishes this wide range of gaseous animations by making extensive use of three-dimensional tables (the unified solid space data structure) for controlling the gas's direction, maximum density, and velocity. A separate three-dimensional table is used for accurate gas self-shadowing and shadowing of other objects [20]. The use of these tables for representing solid spaces controlling attributes of the gas will be be described in detail later in this chapter.

2.1 Modeling the Geometry of the Gases

2.1.1 Surface defined gases

The author originally used solid texturing of polygonal mesh objects' transparency for modeling gaseous phenomena [19]. This was originally done to produce a static image of a butterfly emerging from fog [16]. The solid texturing functions are based on the turbulence functions described earlier. To simulate gaseous phenomena, solid texturing was used to control the transparency of objects that define the space that the gaseous substance occupies. In this way, solid textured transparency can be used to simulate fog, smoke, and clouds. For a cloud layer, one flat plane can be used. The transparency of this plane creates the clouds. The transparency is controlled by a turbulent flow based solid texturing function. A similar result could be obtained by having solid textured ellipsoids occupying the same solid texturing space. By using the ellipsoids, it is possible to get results similar to Gardner [24]. Gardner uses flat planes for creating cloud layers and ellipsoids for creating individual clouds positioned in space.

A simple function to generate clouds is given in Figure 1.

```
clouds(pnt,pixel_size,frequency,power)
    xyz_td  pnt; /* the location of the point in
                  * the solid texture space */
    float   pixel_size, frequency,power;
{
  /* add some noise to the pnt */
 pnt.x += noise(pnt); pnt.y += noise(pnt);
  /* use a sine wave for the basis of the shape */
 tmp=sin((turbulence(pnt, pixel_size)*frequency));
 return(1.0 -pow(tmp+1.0,power)*.5);
}
```

Figure 1: Function to generate clouds.

By changing the parameters in this function, static images of fog, mist, or smoke can be created. The *frequency* parameter controls the frequency of the sine wave through the turbulent space. The *power* parameter controls the distribution of the transparency values, basically determining whether linear interpolation, quadratic interpolation, or other types of interpolation are used to generate the transparency value. The other parameter that greatly affects the shape of the gaseous substance is the size of the solid texture space relative to the shape-defining object. Increasing this relative size creates the effect of zooming in on a portion of the gas. For smoke, using a function similar to the above to change the solid color space of the opaque portions will allow for a more realistic effect.

This approach for modeling gases has several advantages over previous approaches.

First, since it is a surface-based approach, it requires less computation time than volume-based approaches. Secondly, it allows for a greater variety in the geometry of the gas as compared to constant density approaches[34, 43].

This surface-based approach, however, has several disadvantages as compared to volume based approaches. First, accurate shadows cannot be cast from the gas since it is represented by a surface and not a volume. Secondly, since it is represented by a hollow object or flat plane, objects moving through the gas can be seen to pass through the defining surface-based object. Finally, if the observer enters the defining object, the entire gas will disappear or at least the front surface of the gas will disappear. This would be a major abrupt discontinuity in an animation.

2.1.2 Volume-modeled gaseous phenomena

The above technique has been extended to volume modeling of the gaseous phenomena. The turbulent flow based functions now control the density and geometry of the gases rather than the transparency of a polygonal object. In volume modeling these phenomena, a three-dimensional solid defines the space that these gases occupy. Any ray-traceable solid can be used. For example, a parallelepiped is used to define the space for most of the images in this dissertation. The author has also used spheres to define the space for the gases.

One problem with this approach is that the solid's boundaries can be observed. However, by controlling the density functions, the defining shape of the solid can be made undetectable (if desired). This is easily accomplished by decreasing the density based on the location within the volume. For example, in Figures 14 and 15 the density is ramped off cylindrically from the center of the gas. The horizontal radial distance from the center of the gas is calculated. If this distance is greater than a certain percentage of the radius of the glass or teacup, the density of the gas is scaled linearly to 0 at the edge of the glass or teacup.

A solid gas geometry space is built upon the turbulence space through the use of volume density functions. These functions define the density of the gas throughout the solid space. The volume density functions take the location of the point in world space, find it's corresponding location in the turbulence space and applies the turbulence function. The value returned by the turbulence function is then "shaped" to simulate the type of gas desired by using simple mathematical functions. Power, exponential, and sine functions are examples of some of the functions that the author has used for "shaping" the gas. For example, a simple function used to define the geometry of steam rising from a teacup (see Figure 15) is given in Figure 2. This function uses the power function to create the general shape of the gas.

In Figure 2, *parms[1]* is the maximum density for the gas (a value between 0 and 1.0). *Parms[2]* is the exponent to raise the density to in shaping the gas. The higher the value, the greater the rate of change of the density of the gas. *Parms[3]* controls how much randomness to add to the density value of the gas.*Parms[4]* is a minimum threshold density value for the gas. Any density value less than *parms[4]* becomes a zero density value. *Parms[9]* specifies how many iterations of the turbulence function should be performed. The density value is shaped using the power function to create

```
steam(pnt_turb, density, parms, pnt_w, vol)
     xyz_td  pnt, pnt_w;
     float   *density,*parms;
     vol_td  *vol;
{
 float tmp, dist_sq, density_max, tmp2, offset3;
 extern int frame_number;
 xyz_td direction,cyl, diff;

  /* add some turbulence to the point and calc the path */
  tmp = turbulence(pnt_turb,.26);
  pnt_turb = pnt_turb + 2.0 + tmp; /* for each .x,.y,.z */
  path = calc_path(pnt_turb, frame_number);
  direction = pnt_turb + path; /* for each .x,.y,.z */

  /* evaluate the turbulence function for the point */
  *density = turbulence(direction, parms[9]);
  *density = pow( ((*density *(*density))*parms[1]), (double)parms[2]);

  /****** RAMP IT OFF SPHERICALLY *****/
  XYZ_SUB(diff,vol->shape.center, pnt_wld);
  dist_sq = DOT_XYZ(diff,diff);
  density_max = dist_sq/vol->shape.rad_sq;
  density_max = density_max +parms[3]* variation(pnt_turb);
  if(density_max >= .25)
   { /* get in range 0.0 : 1.0 */
     density_max = (density_max -.25)/.75;
     density_max = (cos(density_max *PI)+1.0)/2.0;
     *density = *density *density_max;
   }

  /**** RAMP IT OFF VERTICALLY   ****/
  tmp2 = pnt_wld.y - vol->shape.center.y;
  if(tmp2 > 0.0)
    {
     tmp2 = (tmp2 +variance(pnt_turb)*.1)/vol->shape.radius;
     if(tmp2 > .15)
       {
        offset3 = (tmp2 -.10)*1.111111;
        offset3 = 1 - (exp(offset3)-1.0)/1.718282;
        *density *= offset3*parms[1];
        }
     }

  /**** RAMP IT OFF CYLINDRICALLY ****/
  diff.y =0.0;
  dist_sq = DOT_XYZ(diff,diff);
  density_max = dist_sq/vol->shape.rad_sq +parms[3]*variance(pnt_turb);
  if(pnt_w.x - vol->shape.center.x  > .7 )
    *density =0.0;
  else if(pnt_w.x - vol->shape.center.x > 0.45)
    {
      tmp2 = 1.0 -((pnt_w.x - vol->shape.center.x) -.45)*4;
      if(*density > tmp2*parms[1])
        *density = tmp2*parms[1];
     }
  if(*density < parms[4])
     *density=0;
 }
```

Figure 2: Function for steam density.

the general shape of the steam. Then the steam density is ramped off cylindrically so that the steam only rises above the cylinder created by the teacup. To create the steam dissipating as it rises, the density is ramped off spherically and vertically.

2.2 Basic Movement of the Gas

There are two obvious ways that the above technique which simulates gaseous phenomena can be animated. The first technique animates the turbulence space. Animating the turbulence space would require the use of functions to move the values in the noise lattice along the specified directional movement of the gas. This would require applying the path movement function to each table element in the lattice (currently 64x64x64 elements) for each functional evaluation of the gas, then interpolating these values to calculate the new noise for each point in the lattice. This would be similar to a particle system approach.

The second technique moves the objects through the turbulence space. The object can be viewed as moving through the turbulence space over time. This is described in detail below. This technique is the approach described here since it is computationally more efficient and still gives effective results.

The first would be better for a true physically-based simulation of turbulent flow. However, since the method described here is not a physically-based simulation of turbulent flow, animating the turbulence space would be artificial and not worth the computational expense. Applying the path movement function to each point in the lattice far exceeds the computational expense of the second approach: evaluating the path function for each functional evaluation of the gas.

The movement of the gas is created by moving the fixed three-dimensional screen space point along a path over time through the turbulence space before evaluating the turbulence function. Each three-dimensional screen space point is inverse mapped back to world space. Then from world space, it is mapped into the gas and turbulence space through the use of simple affine transformations. Finally, it is moved through the turbulence space over time to create the movement of the gas. Therefore, the path direction will have the reverse visual effect. For example, a downward path will show the gas as rising. For each three-dimensional location in screen space, the portion of gas that occupies this location on the screen for this frame is determined. In this way, the techniques presented here can be considered to be an *inverse particle system*. In the case of particle systems (e.g. [55]), the particles are moved over time and each location in three-dimensional screen space is then analyzed to determine which particles occupy this location. In the case of the approach presented here, each point in three-dimensional screen space is moved through the gas space to see which portion of the gas occupies the current location in screen space.

This approach has the advantage over particle systems in that extremely large geometric databases of particles are not required in order to get realistic images. The use of particle systems would require an extremely large number of particles to produce the same images. At least 1 particle would be needed for each volume sample to produce the same quality of image, since this is the sampling resolution for the approach described in this dissertation. So for example, at least $1024 \times 682 =$

139, 673, 600 particles would be necessary to produce an image like the image in Figure 11. This would require a large amount of both memory and disk space. The approach described in this dissertation does not require the storage of any particles to produce these effects.

The approach presented here does have a few disadvantages as compared to particle systems. In the inverse particle system approach, the functions that create the movement of the gas are more complex than for particle systems. The functions for particle systems are given the particle's location and velocity vector. This information can then be used in calculating the new location and velocity of the movement of the particle. This information is not passed to the inverse particle system functions. It must be built into the function. Another disadvantage of the inverse particle system approach is computation time necessary to determine the density of the gas. Functional evaluation is necessary to determine the density of each element in the gas. With particle systems, this additional computation is not necessary. The trade-off becomes the computation time to evaluate the density functions for the gas vs. the computation time to determine which particles occupy which location on the screen.

2.3 Vector Field Tables

The vector field table allows external programs to generate the three-dimensional fields that control the gas movement. The information contained in these tables is described in Table 1.

The direction vector defines the path for the gas movement. The density scaling

Table 1: Vector Field Elements

- direction vector
- density scaling value
- percent of vector to use

factor in the table allows the density of the gas to be decreased. The "percent of vector to use" field specifies how much of the table direction vector to use and how much of the default gas path to use, allowing for a smooth transition between the path defined by the table values and the default path of the gas. This "percent to use" value is a weighting factor for combining the default path vector and the vector field table value. These two values vector path are combined as follows:

$$\begin{aligned} final\_path &= percent\_to\_use \times vector\_field\_table\_direction + \\ &\quad (1 - percent\_to\_use) * default\_path\_direction \end{aligned} \tag{2.1}$$

Vector field tables can generate a wide range of results from choreographed gas movement to scientific visualization of actual gas dynamics calculations. Since these tables have finite resolution, vector field tables seem more suited for controlling general gas movement than intricate gas movement. To control the intricate gas movements, vector field tables can be combined with function field tables, which are described in a later section.

2.3.1 Gas Flow Around Objects

The major use of vector field tables by the author has been the simulation of simple laminar flow of gases around simple objects, such as cylinders and spheres. For incompressible inviscid (zero viscosity) flows, where the flow does not separate from the body, gases flow tangentially around symmetric bodies except for the thin boundary layer [12]. The flow about an arbitrary cylinder in space can be calculated by aligning the cylinder with the Z-axis, calculating the flow about the cylinder aligned with the Z-axis, and then applying the inverse of the transformation to the flow vector to get the flow about the arbitrary cylinder.

The formula below is used for each lattice point to determine the flow about the cylinder. The information needed is the general direction of the gas flow ($gas\_direction$), the specification of the cylinder ($cylinder\_top, cylinder\_radius$), and the lattice point value in world space coordinates (pnt). To calculate the direction flow vector, the following formula is used for each point that is between the top and bottom of the cylinder and within the influence of the cylinder:

$$flow\_vector = (perp\_vector) \times (gas\_direction) \times (perp\_vector) \tag{2.2}$$

where $perp\_vector$ is the vector difference ($pnt - cylinder\_top$) with the Z component set to 0.

The user can specify the radial distance about which the gas flow is affected. Over this distance, the "percent of vector to use" is ramped from 1.0 at the edge of the cylinder to 0.0 at the threshold distance from the cylinder. The "density scaling factor" is used to create an absence of gas within the cylinder. This factor is set to zero for each point that is inside the cylinder. In the flow calculations, flow can be calculated around a cap on the cylinder, or the flow above or below the cylinder can be left unperturbed (open cylinders).

By combining multiple open cylinders, flow about more complex shapes can be approximated. This can also be extended to flow about elliptic cylinders. The only change in the formula is finding the perpendicular vector from the point to the cylinder. This same calculation can be used to simulate the flow around a symmetric Joukowski aerofoil (a class of areofoils used in aerodynamics created from applying a Joukowski transformation to a circular cylinder). The flow about the cylinder is calculated and by using the Joukowski transformation (a conformal mapping), the resulting flow vector can be calculated [12]. To calculate the flow about a sphere the same flow vector formula can also be used where the $perp\_vector$ is the vector difference $(pnt - sphere\_center)$.

Figure 16 shows the vector field table created for flow about a cylinder aligned with the Z axis. The lengths of the vectors show how much of the vector will be used. Figure 17 shows the flow about a circular cylinder. This figure also has a vortex above the cylinder to simulate a more turbulent wake.

The above calculations are merely a "visual simulation" of simple laminar flow about symmetric objects. As mentioned earlier, visual simulation is a very valuable approach for producing animations of gases since the actual physical simulations are computationally extremely complex.

2.4 Functional Flow Field Tables

The other type of tables used to control the gas movement are functional flow field tables. The major use of these tables is for choreographed animation of the gases. These tables define, for each region of the gas, which function to evaluate to control its movement. For each function, a file is specified which contains the type of function and parameters for that function. The functions evaluated by the flow field tables return the information in Table 2.

Table 2: Flow Field Function Values.

Flow Field Function Values
- direction vector
- density scaling value
- percent of vector to use
- velocity

As can be seen, these functions return the same information as that contained in the vector field tables, with the addition of a velocity scaling factor. The advantage of the flow field functions over the vector field tables is that they can provide infinite detail in the motion of the gas; they are evaluated for each point that is volume rendered, not stored at fixed resolution. The disadvantage of the tables is that the functions may be much more expensive to evaluate than simply tri-linearly interpolating values from a table.

The "percent to use" value in the table is again used to provide for a smooth transition between control of the gas movement from the flow field functions, vector field tables and the default path of the gas. This value is also used to allow the smooth transition between control of the gas by different flow field functions. The "percent to use" value will decrease as you move away from the center of control of a given flow field function.

The best way to create these flow field function tables would be with an interactive tool which allows the user to interactively place these functions throughout three-space. For the results presented in this dissertation, however, the flow field function tables were created either by hand or by very simple programs.

2.4.1 Flow field functions

The authors have developed many types of flow field functions that provide interesting animation effects. Additional basic functions are being developed to create a toolbox of functions which can be combined to provide complex choreographed animation. The next several sections describe some of these functions.

Helical paths

A surprisingly large amount of gas animation is convincing when the movement is modeled using simple helical paths. This is the default movement used for all the gases in the results presented in this paper and was the path for the fog movement in "Getting Into Art"[18]. The helical paths seem to add a sufficient amount of variance to an alternative straight path to produce realistic movement. Figure 15 shows a simple helical path movement for steam rising from a teacup. The function in Figure 2.4.1 is a typical helical path function.

```
hel_path(pnt, flow_path,density_scaling, velocity, pixel_size)
     xyz_td  pnt, *flow_path;
     float   *density_scaling, pixel_size;
{
 float tmp;
 extern int frame_num;
 xyz_td cyl;

 /* apply some turbulence to the path
  * so that it appears more random */
  tmp = turbulence(pnt, pixel_size);

 /* determine how to move the point through
  * the space (helical path)  */
 theta =(frame_num%SWIRL_FRAMES)*SWIRL;
 cyl.x = ELLIPSE1 * (float)cos(theta);
 cyl.z = ELLIPSE2 * (float)sin(theta);
 flow_path.x = cyl.x +tmp;
 flow_path.y = -(float)frame_num*DOWN - tmp;
 flow_path.z = cyl.z +tmp;

 *density_scaling=1.0;
 *velocity = NORM(flow_path);
}
```

Figure 3: Simple helical path function.

This function creates upward swirling movement in the gas, which swirls around 360 degrees every SWIRL_FRAMES frames. Turbulence is applied to the path so that it appears more random. The parameters ELLIPSE1 and ELLIPSE2 determine the elliptical shape of the swirling path. DOWN is a parameter that specifies the rate at which the gas moves downward though the gas space.

Attractor and repulsor paths

Attractors and repulsors are primitive functions that can provide a wide range of effects. Figure 18 shows some frames of an attractor's attracting value increasing in strength over time. There is a maximum attraction value for the attractor. This value is simply interpolated from a specified low value to a specified higher value over time. By animating the location and strength of an attractor, many different effects can be achieved. Effects such as a breeze blowing (see Figure 19) and the wake of a moving object can easily be created. Attractors simply create paths radially away from the center of attraction and repulsors create paths radially towards the center of repulsion (as stated previously, path movement needs to be in the opposite direction of the desired visual effect). The function in Figure 4 is an example of a simple attractor function.

The $flow\_func\_td$ structure contains parameters for each instance of the spherical attractor. The parameters include the center of the attractor($ff.center$), the effective distance of attraction ($ff.distance$), and where to begin the falloff from the attractor path to the default path ($ff.falloff\_start$). This function ramps the use of the attractor path from ($ff.falloff\_start$) to ($ff.distance$). A cosine function is used for a smooth transition between the path defined by the attractor and the gas's default path. Variations on this simple attractor include moving attractors, angle limited attractors, attractors with variable maximum attraction, and combinations of all three types.

Spiral vortex functions

Vortex functions can have a variety of uses from simulating actual physical vortices to creating interesting disturbances in flow patterns as an approximation of turbulent flow. The vortex function is based on the simple 2D polar coordinate function:

$$r = \theta \tag{2.3}$$

which translates into three-dimensional coordinates as

$$x = \theta \times cos(\theta) \tag{2.4}$$

$$y = \theta \times sin(\theta) \tag{2.5}$$

The third dimension is normally just linear movement over time along the third axis. To animate this function, θ is based on the frame number. To increase the vortex action, a scalar multiplier for the sine and cosine terms based on the distance from the vortex's axis is added. The effect of the vortex decreases as the distance from the center increases using a linear or quadratic falloff function. This is by no means a true physical simulation of gaseous vortices. Simulating true turbulent flow characteristics, such as those found in Karman vortex streets (turbulent flow induced vortices in the wake of flow about an object) is extremely complex and requires large amounts of supercomputer time for approximation models.

Another type of vortex can be created in a manner similar to the vortex functions used by Sims [55]. For these vortices, the angle of rotation about an axis is determined by both the frame number and the relative distance of the point from the center (or axis) of rotation. The direction vector is then the vector difference of the transformed point and the original point.

Examples of the effects achievable by our vortex functions can be seen in Figures 20 and 21. Both types of vortex functions are very useful for creating swirling wakes in the flow of gas around objects (see Figure 17). Animating the location of these vortices also produces interesting effects, especially when coordinating their movement with the movement of objects in the scene. In this manner, the wake from a moving object can be created. The location of the center of the vortex can be animated to follow the motion of an object.

```
spherical_attractor(point, ff, direction, density_scaling,
                    velocity, percent_to_use)
     xyz_td       point, *direction;
     flow_func_td ff;
     float        *density_scaling, *velocity, *percent_to_use;
{
  float        dist, d2;

  /*calculate distance and direction
   *from center of attractor
   */
  XYZ_SUB(*dir, point, ff.center);

  dist=sqrt(DOT_XYZ(*dir,*dir));

  /* set the density scaling and the
   *  velocity to 1
   */
  *density_scaling=1.0;
  *velocity=1.0;

  /* calculate the falloff factor (cosine)
   */
  if(dist > ff.distance)
    {
      *percent_to_use=0;
    }
  else if (dist < ff.falloff_start)
    {
      *percent_to_use=1.0;
    }
  else
    {
      d2 = (dist - ff.falloff_start)/
           (ff.distance - ff.falloff_start);
      *percent_to_use = (cos(d2*M_PI)+1.0)*.5;
    }
}
```

Figure 4: Spherical attractor function.

CHAPTER III

Conclusions, Results, and Future Extensions

3.1 Conclusions

This dissertation has shown that solid spaces are a unified approach to describing object attributes. All object attributes, including the object's geometry and even the object's motion can be represented in a unified data structure through the use of solid spaces. This unified data structure for representing object attributes allows the creation of a rendering and animation system that incorporates many recent advances in computer graphics and animation in one unified system. This unified system is cohesive, robust, and easily extendable since it is built upon the unified data structure and access functions for solid spaces. This unified system is an extremely powerful system for producing state-of-the-art images.

The rendering system supporting solid spaces shows that volume rendering and scanline a-buffer rendering can be efficiently combined. This system places no restrictions on the volume or surface-defined objects and even allows the volume modeled objects to accurately cast shadows on the other objects in the scene.

This dissertation has shown how solid spaces are a very useful tool for the modeling, rendering, and animation of gaseous phenomena. Vector field tables and functional flow field tables provide a general and powerful method for controlling the animation of gaseous phenomena. A wide range of effects can be created from the use of these tables. The animation of gas flow around symmetric bodies is achievable using the vector flow field tables presented in this dissertation. The functional flow field tables also provide an easy way to choreograph a wide range of animation effects. The combination of the vector field tables and the functional flow field tables provide a system that allows realistic animation of many different gas movement patterns.

3.2 Discussion of Results

This section shows the wide range of results achievable by the use of solid spaces for describing object attributes. The use of solid spaces for describing object color, transparency, shadowing, geometry, and motion are all shown. The computational expense associated with these techniques is also presented. These computational times should be considered an upper limit for computational time to produce the associated images, since the system has not been optimized for speed. The same visual results might also be achieved by using larger step sizes in volume tracing and shadow tracing and by using lower resolution shadow-tables, all of which would decrease the computation time. Empirical testing would be necessary to pin down these threshold values.

3.2.1 Solid space rendering system and gaseous modeling

As mentioned above, the rendering system supporting solid spaces shows that volume rendering and scanline a-buffer rendering can be efficiently combined to produce realistic scenes with gaseous phenomena. The images in Figures 8 through 14 show the results achievable from the rendering system and the method for modeling gaseous phenomena described in this dissertation. These images were calculated on a Hewlett-Packard 9000/370 TSRX workstation with 16 Mb of memory. Table 3 shows rendering times for Figures 9, 10, 11, 12 and 13. Table 4 shows rendering times for each of the images in Figure 14. All rendering times are approximate. The rendering times are were computed using the "/bin/time" Unix function. Figure 9 shows an art gallery scene containing a volume rendered gaseous sphere without volume shadowing. Table 1 provides calculation times for the image in Figure 9 and the same scene without the volume. From these times, it is clear that the addition of a volume rendered object in the scene only slightly increases the computation time.

Figure 10 shows a scene from "Once a Pawn a Foggy Knight..." showing surface based solid textured transparency to create the fog. In this image, three planes were positioned in the scene. The transparency of these planes was then modulated using turbulence-based functions to create the fog. Figure 11 shows the results obtainable by extending the turbulent flow based solid texturing to volume modeling. In this image, the geometry of full a three-dimensional volume is controlled by the turbulence-based volume density functions. This scene is from "Getting Into Art" [18], and has table-based volume shadowing. Figure 12 shows 6 images of a volume-modeled gas rising out of a glass casting shadow-traced shadows on the glass and the wall. The steam, however, does not shadow itself. The motion of the steam is created using a vertical helical path. These images represent every 30th frame of an animated sequence. Figure 13 shows 9 frames of a chess scene similar to Figure 10. In these images, the fog is volume rendered and is "rolling" in from the left of the image. The path for the fog in these images was again a helical path. However, the helical path was horizontal to the left of the image. The front of the fog effect was created using a cut-off plane: any x value greater than (to the right of) the cut-off plane had its transparency ramped-off to zero. The cut-off plane was then moved over time to the right of the image. Notice the shadows of the fog on the rook change as it moves forward. Also notice the shadows of the fog move across the curved base of the rook, the top of the pawn, and the ground plane in the last 3 images.

Table 3: Rendering Times for Figures 8 through 13

Figure	*Resolution*	*Rendering Time (minutes)*	
8	682x1024	62	
9	1024 x 682 1024 x 682	71 59	with volume (actual picture) without volume
10	1024 x 768	105	
11	1024 x 682	260	
12	420 x 400	95	each image
13	320 x 240	165	each image

From this discussion of the results, it is clear that the computational time increases proportionally to the amount of gaseous volume in the image. Self-shadowing of the gases is also very expensive if shadow tracing is used. Figure 14 shows 4 images of a glass with steam rising from it. The first image has no shadowing at all. The second image has shadowing of the volume onto the glass. The third and fourth images have self-shadowing of the volume and shadowing of the volume on the glass. Notice the drastic change in the appearance of the gas. All of these images use the same geometric model for the steam. The difference in the appearance of the shape of the steam in the images in the bottom of the figure is caused by self-shadowing of the steam. Self-shadowing of the gas increases the contrast and definition of the gas. Notice that the images in the bottom of the image are virtually indistinguishable. In the third image, the shadow-table technique was used with shadow table dimensions of 64x64x64. In the fourth image, shadow-tracing was used. The calculation times are given in Table 4. This further illustrates the improvement in computational time with table-based shadowing.

Table 4: Rendering Times for Figure 14. Resolution is 360x250

Figure	*Shadowing Level*	*Rendering Time (minutes)*
a	none	80
b	shadow-traced, no self-shadowing	101
c	shadow-traced, with self-shadowing	1800
d	table-based, with self-shadowing	127

The combination of the volume rendering and scanline A-buffer rendering has wide ranging applications beyond the rendering of gaseous phenomena. This technique allows any volume data to be rendered in scenes with traditional surface based objects. The use of an a-buffer scanline renderer is not necessary if anti-aliasing is not desired. A normal scanline z-buffer could be used instead of the a-buffer; however, transparency will also not be handled easily, which is a normal problem with scanline z-buffer renderers.

Many additional performance improvements are possible. The author found that initially 65% of computation time was being spent in turbulence function evaluations. A 60% performance improvement in the turbulence function evaluation alone has since been achieved. There are certainly more performance improvements possible in the volume rendering section of this new system. Another way to improve performance is to use a stored table of functional values with a lower resolution. This should provide improvements in computation time since volume density functional evaluations are the major computational expense.

Animation using the above techniques is very suitable for distributed processing. The author has used over 150 SUN and Hewlett-Packard workstations in computing the animation "Once a Pawn a Foggy Knight... ." The same network distribution software used for this animation has been changed to handle the new techniques described above. Since the rendering time of a single frame is high, each workstation could also be given a range of scanlines to compute. The network distribution software took approximately 20 minutes to distribute jobs to 150 machines. This time could easily be decreased if detailed information about the performance of machines was not kept. By using a large distributed network like this, high resolution animations could easily be produced at the rate of more than 10 seconds per day.

3.2.2 Animation effects from solid spaces

Solid spaces are a very useful tool for the modeling, rendering, and animation of gaseous phenomena. Solid space animation tables provide a general and powerful method for controlling the animation of gaseous phenomena, allowing a wide range of images to be created. The images in Figures 15 through 21 show the results achievable from the use of the animation techniques described in this dissertation. These results were calculated on a Sun Sparcstation SLC with 8 Mb of memory. Table 5 shows calculation times for Figures 15 through 21. Again, these rendering times are approximate and can be considered an upper limit on the calculation times needed to produce the corresponding effects. The functions used to produce these effects can be optimized to achieve better performance.

Table 5: Rendering Times for Figures 15 through 21

Figure	*Resolution*	*Rendering Time (minutes)*
15	1024 x 682	105
17	320 x 240	34
18	320 x 240	53
19	320 x 240	31
20	320 x 240	52
21	320 x 240	47

The wide range of effects achievable by these techniques can be seen in the images in the figures. Figure 15 shows steam rising out of a teacup using a simple helical path for the movement of the steam. As you can see, even this basic path movement provides a convincing simulation of steam rising. Figure 17 shows the flow of steam around a cylinder created by using the vector flow field tables. The tangential flow patterns around the cylinder can clearly be seen in the figure. A vortex is also present above the cylinder to simulate the turbulent wake of the flow about a cylinder. The realism of the image is increased by the addition of the vortex to simulate the naturally occurring vortex wake of turbulent flow in gases.

The effects achievable by other functional flow field table functions can be seen in Figures 18 through 21. Figures 18 and 19 show effects achievable by attractors. Figure 18 shows the effect of increasing an attractor in strength over time. The movement of the gas slowly changes direction toward the attractor from its default horizontal movement. Figure 19 shows frames of an animation of a breeze blowing the steam from the left of the image. To produce this effect, an attractor was placed to the right of the teacup and the strength of attraction was increased over time. The maximum attraction was only 30%, so it appears as a light breeze. Increasing the maximum attraction would simulate an increase in the strength of the wind.

Figures 17, 20, and 21 show the various effects achievable by the use of vortex functions. The functions provide very interesting movement patterns in the gas. These functions can be used to produce swirling wakes created by moving objects, simulations of turbulent flow wakes (see Figure 17), and many other interesting effects.

3.3 Future Extensions

The techniques described in this dissertation can be extended in many ways. First, the rendering system could be extended to include voxel-based volume rendering. This proposes no new problems in combining the volume rendering with the A-buffer.

Different approaches could also be taken for modeling the gases. Fourier synthesis or fractal models could be used instead of the turbulence model described in this dissertation. Alternatively, the techniques for modeling and animating the gaseous phenomena could be physically-based. One idea is to control the overall fog movement based on a more global turbulent directional field. In this way, for example, the fog would move towards areas of low turbulence. A three-dimensional table containing barometric pressure readings could also be used to create the directional movement of the gas by using the gradients of the values to control the speed and direction of the movement. In this way, you could begin to develop a physically-based model. The next step towards a physically-based model is to develop a physically-based turbulent flow model for various types of gases. This model seems to be computationally complex, but could easily be added into the current system since it is functionally based. The author feels that developing a physically-based turbulent flow function, then simplifying the calculations to be tractable while maintaining the same visual quality is a good approach.

The author is also interested in extending the techniques to animating physically-based gas flow patterns through the use of the solid space animation tables. So far only simple laminar flow about symmetric objects has been modeled. Extending this to handle turbulent gas flow would provide a much more robust gaseous animation system. However, as mentioned earlier, current gas dynamics simulations require very large amounts of supercomputer time and can only accurately compute gas flow around very simple objects. Arbitrary gas flow in an ordinary room is still beyond the capabilities of current gas dynamic simulation techniques [42]. The author is currently investigating models for *buoyant plumes* to produce correct motion patterns for steam rising from a teacup or for smoke rising from a smoke stack.

The functional-based volume modeling and animation techniques described in this dissertation could be extended to include functions for simulating fluid flow. By incorporating simple fluid flow models to generate vector field tables and using a surface-based illumination model instead of a gaseous model, this unified system should be able to produce animations of fluids as easily as animations of gases. Other volume density functions besides fluids and gases, such as hypertextures [47], can be animated by the use of the solid space animation tables. Again, the only difference would be rendering the volume densities as non-gaseous volumes (which is already implemented in the system), instead of the gaseous model used here.

The solid space animation tables could also be used for simulating the effects of wind or water currents on objects. For example, by creating a vector field table containing wind fields, animations of leaves blowing on a windy day could be simulated.

The solid space animation tables could also be combined with a new solid space table that contains the current location within the solid geometry space. The solid space animation tables would then be used for an incremental movement through the solid geometry space. This would increase the frame-to-frame coherence in the movement of the gas. This should also allow for more complex motion of the volumes. This new solid space table would of course need to be calculated incrementally from one frame to the next in order to contain coherent information.

Additional flow function primitives can be created to create a more extensive toolbox of functions for providing complex motion patterns for gases. Functions that simulate physical gas phenomena such as Karman vortices and Karman vortex streets would provide more complex gas movement. Also, by adding more simple mathematical functions such as logarithmic spirals, more interesting and complex gas motion will be achievable.

The development of interactive tools for the creation of the solid space animation tables is another extension to the techniques described in this dissertation. This would allow the animator an easier method for controlling the animation of the gases. This would provide an easy way for the animator to combine various flow functions which control separate areas of the gas movement, providing more complex gas movements.

Figure 5: Volume element creation. Shaded areas are the three volume elements.

Figure 6: Shadow table Calculation.

Figure 7: Three Dimensional table around a volume.

Figure 8: Art Gallery I.

Figure 9: The modern pioneer in an art gallery.

Figure 10: A representative image from “Once a Pawn a Foggy Knight...”.

Figure 11: A representative image from “Getting Into Art.”

Figure 12: Steam rising from a glass. The 6 images are every 30 frames.

Figure 13: Scenes of fog rolling in. The first 6 are every 80 frames. The last 3 are every 40 frames.

Figure 14: Steam rising from a glass. (a) has no shadowing (b) has shadowing of the gas onto the glass. (c) has table-based volume shadowing and self-shadowing. (d) has shadow-traced volume shadowing and self-shadowing.

Figure 15: Helical path for steam rising from a teacup.

Figure 16: Vector field table for circular cylinder (Z axis view). Vector directions represent the direction of flow of the gas.

Figure 17: Flow about a cylinder with vortex wake. Images are every 21 frames.

Figure 18: Effect of an attractor increasing over time. Images are every 21 frames.

Figure 19: A breeze blowing towards the right created by an attractor. Notice the general movement upward and to the right. Images are every 21 frames.

Figure 20: Spiral vortex. Images are every 21 frames.

Figure 21: Cyclical vortex. Images are very 21 frames.

Bibliography

[1] Armstrong, William, and Green, Mark. Articulated Rigid Bodies for Purposes of Animation. Proceedings of Graphics Interface'85. 407–415.

[2] Bartels, Richard H., Beatty, John, C., and Barsky, Brian A. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling.* Morgan Kaufman Publishers, Inc., Los Altos, California, 1987.

[3] Barzel, Ronen, and Barr, Alan. A Modeling System Based on Dynamic Constraints. Proceedings of SIGGRAPH'88 (Atlanta, Georgia, Aug 1-Aug 5,1988). In *Computer Graphics 22,3* (August 1988), 15,16,179–188.

[4] Bier, Eric A., and Sloan, Kenneth R. Jr. Two-Part Texture Mappings. *IEEE Computer Graphics and Applications 6*, 9 (September 1986), 40–53.

[5] Blinn, James. Simulation of Wrinkled Surface. Proceedings of SIGGRAPH'78 (Atlanta, Georgia, August 23-25,1978). In *Computer Graphics 12,3* (August 1978), 286–292.

[6] Blinn, James. Light Reflection Functions for Simulation of Clouds and Dusty Surfaces. Proceedings of SIGGRAPH'82 (Boston, Massachusetts, July 26-30,1982). In *Computer Graphics 16,3* (July 1982), 21–29.

[7] Blinn, James, and Newell, Martin. Texture and Reflection in Computer Generated Images. *Communications of the ACM 19* (1976), 542–547.

[8] Burt, Peter J., and Adelson, Edward H. A Multiresolution Spline With Application to Image Mosaics. *ACM Transactions on Graphics 2*, 4 (October 1983), 217–236.

[9] Carlson, Wayne, Ebert, David, Parent, Richard, and Boyer, Keith. EDGE - Educational and Developmental Graphics Environment. Tech. Rep. OSU-CISRC-2/90-TR3, Department of Computer and Information Science, The Ohio State University, 2036 Neil Ave, Columbus, Ohio 43210-1277, February 1990.

[10] Catmull, Ed. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* PhD thesis, University of Utah, December 1974.

[11] Chadwick, John, Haumann, David, and Parent, Richard. Layered Construction for Deformable Animated Characters. Proceedings of SIGGRAPH'89, (Boston, Massachusetts, July 31-Aug 4,1989). In *Computer Graphics 23,3* (July 1989), 243–252.

[12] Chow, Chuen-Yen. *An Introduction to COMPUTATIONAL FLUID DYNAMICS*. Seminole Publishing Company, Boulder, Colorado, 1983.

[13] Conlisk, A. Terrence Jr. Private Communication, December 1990.

[14] Cook, Robert. Shade Trees. Proceedings of SIGGRAPH'84 (Minneapolis, Minnesota, July 23-27, 1984). In *Computer Graphics 18, 3* (July 1984), 223–231.

[15] Demko, Stephen, Hodges, Laurie, and Naylor, Bruce. Construction of Fractal Objects with Iterated Function Systems. Proceedings of SIGGRAPH'85 (San Francisco, California, July 22-26,1985). In *Computer Graphics 19, 3* (July 1985), 271–278.

[16] Ebert, David. Cover image of annual report. Tech. rep., Department of Computer and Information Science, The Ohio State University, 2036 Neil Ave, Columbus, Ohio 43210-1277, September 1988.

[17] Ebert, David, Boyer, Keith, and Roble, Doug. Once a Pawn a Foggy Knight ... [videotape]. In *SIGGRAPH Video Review 54* (November 1989), ACM SIGGRAPH, New York. segment 3.

[18] Ebert, David, Ebert, Julia, and Boyer, Keith. Getting Into Art. [videotape], Department of Computer and Information Science, The Ohio State University, May 1990.

[19] Ebert, David, and Parent, Richard. Animation of Gaseous Phenomena Using Turbulent Flow Based Solid Texturing. Tech. Rep. OSU-CISRC-7/89-TR36, Department of Computer and Information Science, The Ohio State University, 2036 Neil Ave, Columbus, Ohio 43210-1277, July 1989.

[20] Ebert, David, and Parent, Richard. Rendering and Animation of Gaseous Phenomena by Combining Fast Volume and Scanline A-buffer Techniques. Proceedings of SIGGRAPH'90,(Dallas, Texas, Aug 6-10,1990). In *Computer Graphics 24,4* (August 1990), 357–366.

[21] Forsey, David, and Bartels, Richard. Hierarchical B-Spline Refinement. Proceedings of SIGGRAPH'88 (Atlanta, Georgia, Aug 1-Aug 5,1988). In *Computer Graphics 22,3* (August 1988), 205–212.

[22] Fournier, Alain, and Reeves, William. A Simple Model for Ocean Waves. Proceedings of SIGGRAPH'86 (Dallas, Texas, August 18-22,1986). In *Computer Graphics 20, 4* (August 1986), 75–84.

[23] Fox, Robert W., and McDonald, Alan T. *Introduction to Fluid Mechanics*. John Wiley and Sons, New York, 1985.

[24] Gardner, Geoffrey. Visual Simulation of Clouds. Proceedings of SIGGRAPH'85 (San Francisco, California, July 22-26,1985). In *Computer Graphics 19, 3* (July 1985), 297–303.

[25] Gardner, Geoffrey. Forest Fire Simulation. Proceedings of SIGGRAPH'90, (Dallas,Texas, Aug 6-10,1990). In *Computer Graphics 24,4* (August 1990), 430.

[26] Girard, Michael, and Maciejewski, Anthony. Computational Modeling for the Computer Animation of Legged Figures. Proceedings of SIGGRAPH'85 (San Francisco, California, July 22-26,1985). In *Computer Graphics 19, 3* (July 1985), 263–270.

[27] Gomez, Julian. Twixt: A 3D Animation System. *Computers & Graphics 9*, 3 (1985), 291–298.

[28] Guenter, Brian K. *A Computer System for Simulating Human Facial Expression*. PhD thesis, The Ohio State University, 1989.

[29] Hahn, James K. Realistic Animation of Rigid Bodies. Proceedings of SIGGRAPH'88 (Atlanta, Georgia, Aug 1-Aug 5,1988). In *Computer Graphics 22,3* (August 1988), 299–308.

[30] Haumann, David R. *Using Behavioral Simulation to Animate Complex Processes*. PhD thesis, The Ohio State University, 1989.

[31] Heckbert, Paul S. Survey of Texture Mapping. *IEEE Computer Graphics and Applications 6*, 11 (November 1986), 56–67.

[32] Kajiya, James, and Von Herzen, Brian. Ray Tracing Volume Densities. Proceedings of SIGGRAPH'84 (Minneapolis, Minnesota, July 23-27,1984). In *Computer Graphics 18,3* (July 1984), 165–174.

[33] Kass, Michael, and Miller, Gavin. Rapid, Stable Fluid Dynamics for Computer Graphics. Proceedings of SIGGRAPH'90,(Dallas, Texas, Aug 6-10,1990). In *Computer Graphics 24,4* (August 1990), 49–58.

[34] Klassen, R. Victor. Modeling the Effect of the Atmosphere on Light. *ACM Transaction on Graphics 6*, 3 (July 1987), 215–237.

[35] Magnenat-Thalmann, Nadia, and Thalmann, Daniel. *Computer Animation Theory and Practice*. Springer Verlag, Tokyo, Japan, 1985.

[36] Mandelbrot, B. *The Fractal Geometry of Nature*. Freeman, San Francisco, California, 1982.

[37] Max, Nelson. Vectorized Procedural Models for Natural Terrain: Waves and Islands in the Sunset. Proceedings of SIGGRAPH'81 (Dallas, Texas, August, 1981). In *Computer Graphics 15,3* (August 1981), 317–324.

[38] Max, Nelson. Light Diffusion Through Clouds and Haze. *Computer Vision, Graphics, and Image Processing 33* (1986), 280–292.

[39] Miller, Gavin. The Definition and Rendering of Terrain Maps. Proceedings of SIGGRAPH'86 (Dallas, Texas, August 18-22,1986). In *Computer Graphics 20, 4* (August 1986), 39–48.

[40] Mortenson, Michael E. *Geometric Modeling*. John Wiley & Sons, New York, New York, 1985.

[41] Musgrave, F. Kenton, Kolb, Craig E., and Mace, Robert S. The Synthesis and Rendering of Eroded Fractal Terrains. Proceedings of SIGGRAPH'89, (Boston, Massachusetts, July 31-Aug 4,1989). In *Computer Graphics 23,3* (July 1989), 41–50.

[42] Nakamura, Shoichiro. Private Communication, November 1990.

[43] Nishita, Tomoyuki, Miyawaki, Yasuhiro, and Nakamae, Eihachiro. A Shading Model for Atmospheric Scattering Considering Luminous Intensity Distribution of Light Sources. Proceedings of SIGGRAPH'87 (Anaheim, California, July 27-31,1987). In *Computer Graphics 21,4* (July 1987), 303–310.

[44] Peachey, Darwyn. Solid Texturing of Complex Surfaces. Proceedings of SIGGRAPH'85 (San Francisco, California, July 22-26, 1985). In *Computer Graphics 19,3* (July 1985), 279–286.

[45] Peachey, Darwyn. Modeling Waves and Surf. Proceedings of SIGGRAPH'86 (Dallas, Texas, August 18-22,1986). In *Computer Graphics 20, 4* (August 1986), 65–74.

[46] Perlin, Ken. An Image Synthesizer. Proceedings of SIGGRAPH'85 (San Francisco, California, July 22-26, 1985). In *Computer Graphics 19,3* (July 1985), 287–296.

[47] Perlin, Ken, and Hoffert, Eric. Hypertexture. Proceedings of SIGGRAPH'89,(Boston, Massachusetts, July 31-Aug 4,1989). In *Computer Graphics 23,3* (July 1989), 253–262.

[48] Prusinkiewicz, Przemyslaw, Lindemeyer, Aristid, and Hanan, James. Developmental Models of Herbaceous Plants for Computer Imagery Purposes. Proceedings of SIGGRAPH'88 (Atlanta, Georgia, Aug 1-Aug 5,1988). In *Computer Graphics 22,3* (August 1988), 141–150.

[49] Reeves, William. Particle Systems - A Technique for Modeling a Class of Fuzzy Objects. Proceedings of SIGGRAPH'83 (Detroit, Michigan, July 25-29, 1983). In *Computer Graphics 17,3* (July 1983), ACM SIGGRAPH, 359–376.

[50] Reeves, William, and Blau, Ricki. Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Sytems. Proceedings of SIGGRAPH'85 (San Francisco, California, July 22-26, 1985). In *Computer Graphics 19,3* (July 1985), 313–322.

[51] Reynolds, Craig. Flocks, Herds, and Schools: A Distributed Behavioral Model. Proceedings of SIGGRAPH'87 (Anaheim, California, July 27-31,1987). In *Computer Graphics 21,4* (July 1987), 25–34.

[52] Rogers, David F. *Procedural Elements for Computer Graphics*. McGraw-Hill Book Company, New York, New York, 1985.

[53] Rushmeier, Holly, and Torrance, Ken. The Zonal Method for Calculating Light Intensities in the Presence of a Participating Medium. Proceedings of SIGGRAPH'87 (Anaheim, California, July 27-31,1987). In *Computer Graphics 21,4* (July 1987), 293–302.

[54] Samek, Marcel, Slean, Cheryl, and Weghorst, Hank. Mapping and Distortion in Digital Graphics. *The Visual Computer 2*, 5 (1986), 313–320.

[55] Sims, Karl. Particle Animation and Rendering Using Data Parallel Computation. Proceedings of SIGGRAPH'90,(Dallas, Texas, Aug 6-10,1990). In *Computer Graphics 24,4* (August 1990), 405–413.

[56] T'so, Pauline, and Barsky, Brian. Modeling and Rendering Waves: Wave-Tracing Using Beta-Splines and Reflective and Refractive Texture Mapping. *ACM Transaction on Graphics 6*, 3 (July 1987), 191–214.

[57] Voss, Richard. Fourier Synthesis of Gaussian Fractals: 1/f noises, landscapes, and flakes. In *SIGGRAPH 83:Tutorial on State of the Art Image Synthesis* (1983), vol. 10, ACM SIGGRAPH.

[58] Willis, P.J. Visual Simulation of Atmospheric Haze. *Computer Graphics Forum 6* (1987), 35–42.

[59] Woodward, Paul. Army High Performance Computing Research Center, University of Minnesota. Private Communication, April 1991.

[60] Zeltzer, David. Motor Control Techniques for Figure Animation. *IEEE Computer Graphics and Applications 2*, 9 (1982), 53–59.