

# OBJECT-ORDER RENDERING OF DISCRETE OBJECTS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for  
the Degree Doctor of Philosophy in the  
Graduate School of The Ohio State University

By

J. Edward Swan II, B.S., M.S.

\* \* \* \* \*

The Ohio State University

1998

Dissertation Committee:

Dr. Roni Yagel, Adviser

Dr. Wayne Carlson

Dr. Kikuo Fujimura

Approved by

---

Adviser

Department of Computer &  
Information Science

© Copyright by  
J. Edward Swan II  
1998

## ABSTRACT

This dissertation gives accurate and efficient methods for the object-order rendering of discrete objects. Discrete objects are typically represented with a volume raster and rendered with a volume rendering algorithm. However, current object-order volume rendering algorithms suffer from several problems. First, they require that the volume raster be traversed in a strict visibility order, but existing visibility ordering methods do not always correctly order perspective projections of volume rasters. Second, both perspective and orthographic renderings of volume rasters can contain aliasing artifacts, but current object-order techniques have no method for addressing these artifacts. Third, computer-generated animations suffer from temporal aliasing artifacts, which can be addressed by adding motion blur. But currently the only motion-blur method for object-order techniques is super-sampling, which is very expensive.

This dissertation presents three new techniques for the object-order rendering of discrete objects which address these shortcomings. First, it gives and proves the correctness of a new method for traversing a rectilinear volume raster in an order that guarantees correct visibility under a perspective projection. Second, it introduces an innovative technique for eliminating the aliasing artifacts that occur when rendering a discrete object. Third, it gives a new technique for adding motion blur to renderings of discrete objects which is much more efficient than super-sampling. It uses the common object-order volume rendering method of *splatting* as a testbed for realizing these new techniques.

This dissertation gives examples of the new techniques applied to volume rendering confocal data, texture-mapping discrete polygons, and rendering terrain datasets. In particular, the terrain representation is innovative compared to traditional object-order methods, which typically represent the terrain with triangular surface patches.

A discrete representation has many advantages over more traditional surface-based representations for many objects besides just terrains. These advantages make it likely that discrete representations will be increasingly used in the coming years. To date, however, the algorithms available for rendering discrete objects have been less mature than those for rendering surface-based objects. The techniques presented in this dissertation promise to help close this gap, and thus make the advantages of a discrete representation more widely available.

Dedicated to my family: John, Bonnie Sue, Suzanne,  
Courtney, and Misty.

## ACKNOWLEDGMENTS

I thank my adviser Roni Yagel for many things: his collaboration and direction on the dissertation research; his advice and direction on my program of study, my conference and career activities, and scholarly publications; for finding the right balance between pushing me to accomplish things and yet also giving me the freedom to work on other projects; and for his conviction that I could actually complete a Ph.D. I also thank him for his advice and direction during my job search and career development. Finally, I acknowledge Roni's most outstanding property: his boundless enthusiasm. After many of our meetings I was so on fire with the exciting possibilities of our efforts that I wanted to stay up all night working. This enthusiasm motivated me like no other force could have: the work in this dissertation is definitely of a higher quality and a greater volume than would otherwise have been possible. I hope to embody a similar infectious enthusiasm in my future collaborations with others.

I also thank my other committee members, Wayne Carlson and Kikuo Fujimura, for their advice and insights regarding this dissertation.

I acknowledge Don Stredney, who employed me for three years, gave me the opportunity to learn about research and grow as a researcher, and who taught me many things about building and managing a research program.

I thank Gary Perlman for his early belief in my research potential, for giving me my first research support, for his generosity, his strong sense of ethics, and for teaching me not only about human-computer interaction but also about many other academic issues.

I thank Wayne Carlson for the wonderful opportunity to work and live at ACCAD for the past five years. There could not be a more creative or pleasurable place to think, collaborate, and work. I also thank Wayne for his advice on many issues, and his collaboration on several publications.

I acknowledge Wayne as well as the students and staff of ACCAD for the excellent camaraderie which I have shared in for so many years. At the risk of leaving someone out, let me mention the staff: Eric Alexander, Pete Carswell, Chuck Csuri, Viki Dennis, Carol Gigliotti, Elaine Hamilton, Barb Helfer, Midori Kitagawa-DeLeon, Kris Laszlo, Ruedy Leeman, Matt Lewis, Steve May, Bob McCarthy, Phil Ritzenthaler, Dennis Sessanna, and Stephen Spencer; my fellow CIS students at ACCAD: Kirk Bowers, Kim Chula, Mark Fontana, Meg Geroch, Leslie Hiemenz, Scott King, Nathan Loofbourrow, Dave Reed, Kevin Rodgers, Ferdi Scheepers, Casey (Kevin) Simon, Ed Sindelar, Karan Singh, Sara Susskind, Jim Vaigl, and Lawson Wade; and the ACCAD art students: Gigi Alandt, Paul Badger, Bren Bataclan, Leslie Bishko, Beth Blostein, Agata Bolska , Yina Chang, Alison Colman, Erika Galvao, Kevin Geiger, Pete Hriso, Jean Ippolito, Zil Lilas, Neil McDonald, Terry Monnett, Flip Phillips, Tonya Ramsey, Wen Hwa (Moon) Seun, Traci Temple, Nathania Vishnevsky, Rina Wayanti, and Hisayo Yoshida.

I also acknowledge my fellow graduate students in the volume graphics research group, and thank them for many years of interesting discussions, collaborations, seminars, and fun parties: Yair Kurzion, Asish Law, Raghu Machiraju, Torsten Möller, Klaus Mueller, Alec Rebello, Dave Reed, Naeem Shareef, and Ph-Wen Shih; and our illustrious leader, Roni Yagel. I also acknowledge Roger Crawfis, the new professor in the group — may his tenure here be long and fruitful!

I acknowledge the good fellowship of the CIS students who went through the degree program with me, in particular John Boyd, Paolo Bucci, Kevin Carpenter, Donglai Dai, Steve Edwards, Wayne Heym, Manas Mandal, Bill Pippin, Srini Raghavan, Kevin Rogers, Lynn Snider, and Pete Ware. I also acknowledge the following CIS staff members who provided excellent help and fellowship throughout my years here: Tom Fletcher, Jim Giuliani, James Jobe, George Jones, Marty Marlatt, Elizabeth O'Neill, and Deanna Tavenner. Finally the ever-competent Eleanor Quinlan deserves special mention: she helped me decide to attend OSU, she provided lots of help as we moved to Columbus, she oversaw all my teaching, and she has always been available to listen to my concerns and give good advice.

I acknowledge the many great friends we have made in Columbus, and thank them for many fun evenings of dining, music, entertainment, and just visiting: Harold & Paulene, Lynn & John, Ferdi & Ronél, Jim & Kris, Frank & Kathi, Chris & Monica, Tom & Sara, and Don & Holly.

I acknowledge the fellowship of Saint Mark's Episcopal Church and my rectors Michael Jupin and Melody Williams. I especially thank Saint Mark's Choir for six happy years of companionship and music. I acknowledge Gary Garber and Michael Murry for both spiritual and musical leadership, and for the inspiring example of lives spent in Music and Faith.

I acknowledge the Ohio State University Jazz Department, and especially Tom Carroll and Jeff Ciampa, for teaching me how to play jazz. I thank Don Chilcote and Bill Chappnick for the *Synergetic Trane* adventure and my first real jazz gigs. And I thank Don for the adventurous trip to the Windy City where we lived and breathed jazz for a week — a journey I hope to make again one day.

I thank my parents, John Edward Swan and Bonnie Sue Swan, for a wonderful and happy upbringing that stressed all the really important things in life; for the continuing example of how to live with grace, dignity, and happiness; and for always giving me unconditional love and support. I thank my sister Suzanne for her unconditional love and support, and for sharing the graduate school experience with me. I also thank her husband Bret Kloos for his fellowship and all the good times we have shared.

I thank my in-laws Kenneth Jones and Sandra Jones for letting me take Courtney so far away for so many years, and for always supporting us. I also thank my sister-in-law Kendra for the many years we have known each other and the many good times we have shared.

I thank my wonderful and sweet wife, Courtney, for following me up to Ohio and helping me through this not-always-pleasant journey in so many ways. I acknowledge her as the main reason I am graduating now and not God-only-knows-how-many years from now. I thank her for the deep and unconditional love that we share.

I thank Misty for her sweet and steady little spirit, for her many kisses, and for always being my first child. I acknowledge her quiet companionship during the many long and lonely afternoons that I sat working while she quietly slept nearby.

Last but certainly not least, I acknowledge God and Jesus for the strength to actually finish this endeavor. I especially thank Him for the many mornings when hopelessness set in, and I felt despair and anxiety about *ever* being able to finish, and yet after a prayer I somehow received the strength to go put in yet another day of labor. This honorable accomplishment is the sum of all those mornings.

“The only *good* dissertation is a *done* dissertation.”  
— Dr. Allen McDonald, January 6, 1997

## VITA

January 13, 1965 ..... Born — Bloomington, Indiana

1986–1987 ..... Software Engineer (Co-op Position)  
BellSouth Services  
Birmingham, Alabama

1989 ..... B.S. Computer Science and Engineering  
Auburn University,  
Auburn, Alabama

1988–1990 ..... Software Engineer  
Optimization Technology Incorporated  
Auburn, Alabama

1992 ..... M.S. Computer and Information Science  
The Ohio State University,  
Columbus, Ohio

1990–1996 ..... Graduate Research and Teaching  
Associate, The Ohio State University  
Columbus, Ohio

## PUBLICATIONS

Don Stredney; Wayne Carlson; J. Edward Swan II; Beth Blostein, “The Determination of Environmental Accessibility and ADA Compliance through Virtual Wheelchair Simulation”, *PRESENCE: Teleoperators and Virtual Environments; First Special Issue on The Application of Virtual Environments to Architecture, Building, and Large Structure Design*, Volume 4, Number 3, MIT Press: Summer 1995.

Shu-Chieh Wu, Jack W. Smith, J. Edward Swan II, “Pilot Study on the Effects of a Computer-Based Medical Image System”. *Proceedings of the 1996 AMIA (American Medical Informatics Association) Annual Fall Symposium*, October 26–30, 1996, Washington, D.C.

Raghu Machiraju, Edward Swan, Roni Yagel, “Spatial Domain Characterization and Control of Reconstruction Errors”. *Proceedings of the EuroGraphics Rendering Workshop '95*, June 12–14, 1995, Dublin, Ireland.

Perlman, Gary & Swan, J. Edward II, “Relative Effects of Color-, Texture-, and Density-Coding on Visual Search Performance and Subjective Preference”, *Proceedings of the 38th Annual Meeting of the Human Factors and Ergonomics Society*, Santa Monica, California: HFES, October 1994, pages 343–347.

Swan, J. Edward II; Stredney, Don; Carlson, Wayne; & Blostein, Beth, “The Determination of Wheelchair User Proficiency and Environmental Accessibility Through Virtual Simulation”, *Proceedings of the Second Annual International Conference: “Virtual Reality and Persons with Disabilities”*, California State University, Northridge, California: CENTER ON DISABILITIES, June 1994, pages 156–161.

Perlman, Gary & Swan, J. Edward II, “Color versus Texture Coding to Improve Visual Search Performance”, *Proceedings of the 37th Annual Meeting of the Human Factors and Ergonomics Society*, Santa Monica, California: HFES, October 1993, pages 235–239. Also appears in G. Perlman, G. K. Green, & M. S. Wogalter (Eds.), *Human Factors Perspectives on Human-Computer Interaction: Selections from the Human Factors and Ergonomics Society Annual Meetings 1983–1994*, Santa Monica, CA: HFES, 1995.

Raghu Machiraju, Edward Swan, Roni Yagel, “Error-Bounded and Adaptive Reconstruction”, *Newsletter of SPIE's International Technical Working Group on Electronic Imaging*, Volume 4, Issue 2, 1995.

Gregory J. Wiet, M.S., M.D.; David E. Schuller, M.D.; Joseph Goodman, M.D.; Don Stredney; Charles F. Bender, Ph.D.; Roni Yagel, Ph.D.; J. Edward Swan II, M.S.; Petra Schmallbrock Ph.D.; “Virtual Simulations of Brain and Cranial Base Tumors”, *Proceedings of the 98th Annual Meeting of the American Academy of Otolaryngology—Head and Neck Surgery*, San Diego, California, September 1994.

Stredney, D., McDonald, J., Wiet, G., Yagel, R., Sindelar, E., & Swan, J.E., “Virtual Simulations Through High Performance Computing”, *Proceedings of Medicine Meets Virtual Reality II*, San Diego, California: University of California, San Diego, January 27–30, 1994.

# **FIELDS OF STUDY**

Major Field: Computer and Information Science

Studies in:

Computer Graphics	Dr. Roni Yagel
Human-Computer Interaction	Dr. Gary Perlman
Software Engineering	Dr. Bruce Weide

# TABLE OF CONTENTS

	<b>Page</b>
Abstract . . . . .	ii
Dedication . . . . .	iv
Acknowledgments . . . . .	v
Vita . . . . .	ix
List of Tables . . . . .	xv
List of Figures . . . . .	xvi
Chapters:	
1. INTRODUCTION . . . . .	1
1.1 Volume Rendering Literature Review . . . . .	2
1.1.1 Surface Fitting Algorithms . . . . .	2
1.1.2 Volume Rendering Algorithms . . . . .	3
1.2 The Splatting Algorithm . . . . .	7
1.2.1 The Splatting Pipeline . . . . .	7
1.2.2 Inherent Problems and Solutions . . . . .	12
1.2.3 Reconstruction Kernel . . . . .	26
1.2.4 Advantages and Disadvantages of Splatting . . . . .	26
1.2.5 Splatting Implementation . . . . .	29
2. A VISIBILITY ORDERING ALGORITHM FOR RECTILINEAR GRIDS . . . . .	30
2.1 Introduction . . . . .	30
2.2 Previous Work . . . . .	31
2.2.1 Regular Grids . . . . .	31

2.2.2	Correct Regular Grid Perspective Ordering Methods . . . . .	35
2.2.3	Non-Regular Grids . . . . .	35
2.3	The Perspective Back-to-Front Visibility Ordering . . . . .	36
2.3.1	Definitions and Assumptions . . . . .	37
2.3.2	1D Visibility Ordering . . . . .	42
2.3.3	2D Visibility Ordering . . . . .	47
2.3.4	3D Visibility Ordering . . . . .	55
2.3.5	Discussion . . . . .	69
2.4	Results . . . . .	69
2.5	Summary and Future Work . . . . .	74
3.	AN ANTI-ALIASING TECHNIQUE FOR SPLATTING . . . . .	75
3.1	Introduction . . . . .	75
3.2	Previous Work . . . . .	75
3.2.1	Analytic Techniques . . . . .	76
3.2.2	Point-Sampling Techniques . . . . .	77
3.3	The Anti-Aliasing Technique . . . . .	80
3.3.1	The Different Spaces . . . . .	81
3.3.2	The Need for Anti-Aliasing in Volume Rendering . . . . .	82
3.3.3	Necessary Conditions to Avoid Aliasing . . . . .	85
3.3.4	An Anti-Aliasing Method for Splatting . . . . .	86
3.3.5	Justification for the Method . . . . .	90
3.4	Results . . . . .	91
3.5	Summary and Future Work . . . . .	93
4.	A MOTION BLUR TECHNIQUE FOR OBJECT-ORDER RENDERING OF DISCRETE OBJECTS . . . . .	100
4.1	Introduction . . . . .	100
4.2	Previous Work . . . . .	101
4.2.1	Analytic Methods . . . . .	101
4.2.2	Discrete Methods . . . . .	104
4.3	The Splatting-Based Motion Blur Algorithm . . . . .	105
4.3.1	Motivation . . . . .	105
4.3.2	Method . . . . .	107
4.4	Results and Discussion . . . . .	110
4.5	Summary and Future Work . . . . .	112
5.	APPLICATIONS TO TERRAIN RENDERING . . . . .	114
5.1	Introduction . . . . .	114

5.2	Previous Work . . . . .	115
5.2.1	Ray Casting . . . . .	115
5.2.2	Shear-Warp . . . . .	116
5.2.3	Object Order . . . . .	116
5.3	Terrain Rendering Examples . . . . .	117
6.	CONTRIBUTIONS AND CONCLUSIONS . . . . .	125
	Bibliography . . . . .	127

## LIST OF TABLES

<b>Table</b>		<b>Page</b>
2.1	The relationship between $vp$ and a 1D grid. . . . .	44
2.2	The relationship between $vp$ and a 2D grid. . . . .	48
2.3	The relationship between $vp$ and a 3D grid. . . . .	58

# LIST OF FIGURES

Figure	Page
1.1 Volume ray casting. . . . .	4
1.2 Volume rendering by affine transformation. . . . .	5
1.3 Object-order volume rendering algorithms. . . . .	6
1.4 Data flow diagram of splatting (from [99]). . . . .	8
1.5 A 3D rotationally symmetric filter kernel is integrated to produce a 2D filter kernel. . . . .	10
1.6 The splatting process: reconstruction and resampling with the 2D filter kernel. . . . .	11
1.7 The <i>splat integration problem</i> : each splat kernel is integrated, not composited, along the viewing direction. . . . .	13
1.8 The overlapping splat problem. . . . .	14
1.9 The <i>splat ordering problem</i> : a small change in the viewing parameters causes the traversal ordering to change. . . . .	15
1.10 Summing the splats in a <i>volumetric summation buffer</i> before the visibility calculation. . . . .	16
1.11 Summing the splats in a sheet summation buffer before the visibility calculation. . . . .	17
1.12 The sheet summation buffer suffers from the splat integration problem, and the splat overlap problem in the screen-space $z$ -dimension. . . . .	18

1.13	An animation illustrating the overlapping splats problem. . . . .	20
1.14	The same animation as Figure 1.13, rendered with composited attenuated splats. . . . .	21
1.15	The same animation as Figure 1.13, rendered with a sheet summation buffer. . . . .	23
1.16	A sheet summation buffer showing the <i>splat ordering problem</i> (see Figure 1.9). . . . .	24
1.17	The sheet summation buffer does not calculate illumination properly for perspective projections. . . . .	25
2.1	Pseudocode for the back-to-front visibility ordering [32]. . . . .	32
2.2	A 2D example of the BTF visibility ordering. . . . .	32
2.3	Pseudocode for the Westover back-to-front visibility ordering [98, 99]. . . . .	33
2.4	Pseudocode for the V-BUFFER visibility ordering [94]. . . . .	34
2.5	The V-BUFFER visibility ordering within each slice. . . . .	34
2.6	The obstructs relation. . . . .	38
2.7	An example of three polygons which do not have a visibility ordering. . . . .	39
2.8	The plane $P$ divides the space into two half-spaces $P^+$ and $P^-$ . . . . .	41
2.9	1D, 2D, and 3D grids. . . . .	43
2.10	A 1D grid showing the location of the view point $\mathbf{vp}$ and point objects divided into the sets $G_L$ and $G_R$ . . . . .	44
2.11	A 1D grid with dividing planes added. . . . .	45
2.12	The right-hand points in the set $G_R$ . . . . .	46
2.13	$\mathbf{vp}_u$ either falls before $\mathbf{p}_1$ or after $\mathbf{p}_n$ . . . . .	46

2.14	Naming conventions for a 2D grid. . . . .	49
2.15	Grid layout for proof of 2D visibility ordering. . . . .	50
2.16	The visibility ordering for the set $G_{LD}$ . . . . .	52
2.17	The viewpoint is beyond the “L” edge of the grid. . . . .	54
2.18	The viewpoint is beyond the “LD” corner of the grid. . . . .	55
2.19	Naming conventions for a 3D grid. . . . .	57
2.20	Grid layout for proof of 3D visibility ordering. . . . .	61
2.21	The visibility ordering for the set $G_{LDF}$ . . . . .	64
2.22	The viewpoint is beyond the “F” face of the 3D grid. . . . .	65
2.23	The viewpoint is beyond the “RF” edge of the 3D grid. . . . .	67
2.24	The viewpoint is beyond the “RDF” corner of the 3D grid. . . . .	68
2.25	A cube rendered with the BTF visibility ordering. . . . .	71
2.26	A cube rendered with the WBTF visibility ordering. . . . .	72
2.27	A cube rendered with the PBTF visibility ordering. . . . .	73
3.1	Specification of the viewing frustum. . . . .	82
3.2	Resampling the volume raster onto the integration grid. . . . .	84
3.3	A comparison of the standard splatting method with the anti-aliased method. . . . .	88
3.4	The geometry for scaling splats drawn after $k$ . . . . .	89
3.5	Calculating the integration grid sampling frequency. . . . .	90
3.6	Rendered image of a plane with a checkerboard pattern. . . . .	92
3.7	Rendered image of a terrain dataset. . . . .	94

3.8	Rendered image of a scientific dataset. . . . .	95
3.9	Traditional texture mapping, showing the pixel preimage for direct convolution, a summed area table, and mip-mapping. . . . .	97
3.10	Traditional texture mapping, showing the preimages of two adjacent pixels. . . . .	98
3.11	Texture mapping with the new method discussed in this chapter. . . . .	99
4.1	Motion blur calculated with the accumulation buffer technique. . . . .	106
4.2	Drawing a motion-blurred splat from position $t_{i-1}$ to $t_i$ . . . . .	107
4.3	The construction of a non-blurred and a motion-blurred splat. . . . .	108
4.4	The motion-blurred splat is drawn perpendicular to the ray from the eye point to position $t_i$ . . . . .	109
4.5	One frame from an animation of a rotating box demonstrating the new motion blur method. . . . .	111
5.1	Four frames from a terrain animation with visibility problems denoted by arrows. . . . .	118
5.2	The same animation as Figure 5.1, using the Perspective Back-to-Front visibility ordering from Chapter 2. . . . .	119
5.3	The same animation as Figure 5.1, using the Perspective Back-to-Front visibility ordering from Chapter 2 and the anti-aliasing technique from Chapter 3. . . . .	121
5.4	Four frames from a terrain animation rendered with an orthographic projection. . . . .	122
5.5	The same animation as Figure 5.4, using the motion-blur technique of Chapter 4. . . . .	123
5.6	The same animation as Figure 5.4, using the <i>accumulation buffer</i> motion blur technique. . . . .	124

# CHAPTER 1

## INTRODUCTION

For the past decade volume rendering has emerged as an important technology in the fields of computer graphics and scientific visualization. Initially volume rendering was primarily applied to the application domains of biomedical visualization and scientific visualization (especially computational fluid dynamics), and these remain important applications. However, volume rendering is now being applied to a broader spectrum of areas in computer graphics, including the modeling, rendering, and animation of objects which have formally only been represented by surface-based primitives. Despite this emerging importance, currently volume rendering as a science is not as mature as surface-based computer graphics. There is still a need for basic research in volume rendering methods and theory.

As is discussed in more detail later in this chapter, *splatting* is a popular approach to volume rendering. The work reported in this dissertation extends the splatting method in four main ways:

- it shows how to correctly render a volume when using a perspective projection with splatting (Chapter 2),
- it gives an anti-aliasing technique for splatting which is particularly useful when using a perspective projection (Chapter 3),
- it introduces a new method for adding motion blur to splatting, (Chapter 4), and
- it applies splatting to the new application domain of terrain rendering (Chapter 5).

Although in this dissertation each of these new techniques is given in the context of splatting and volume rendering, they also have applications that go beyond splatting and volume rendering. These extensions are mentioned, but the main focus of this work is the splatting algorithm.

The rest of this chapter is organized as follows. First, Section 1.1 gives a general review of the volume rendering literature, and provides a classification of the current approaches. This places the splatting approach in the broad context of the field. Then Section 1.2 describes the splatting algorithm in some detail. This section also discusses the particular splatting implementation which serves as the testbed for the extensions which are discussed in the rest of this dissertation.

## 1.1 Volume Rendering Literature Review

In volume rendering, we wish to visualize or render a *volume data set*. A volume data set is usually a 3-dimensional *scalar field*, where each 3D point in the field yields the field strength, represented by a single scalar value. Usually the field is available to us in the form of a rectilinear grid of samples. We sometimes refer to this sample grid as a *volume raster*.

There are a number of algorithms for rendering volume data sets. At the coarsest level, the algorithms can be placed into two categories based on the portion of the volume raster set which they render. This categorization separates them into *surface fitting algorithms* and *volume rendering algorithms*.

### 1.1.1 Surface Fitting Algorithms

Surface fitting algorithms first fit geometric primitives to values in the data, and then render these primitives. The data values are usually chosen from an *iso-surface*, which is the set of locations in the data where the scalar field equals some value. In typical data sets from medical or scientific visualization applications, the iso-surface forms a connected surface, such as the air/skin or brain/bone boundary in a CT dataset. Surface

fitting algorithms locate the desired iso-surface and model it with geometric primitives such as triangles, and then render it using standard polygon rendering techniques.

Surface fitting techniques can be classified according to the type of geometric primitive they use. In the *cuberille* method (Herman and Liu [45]; Chen, Herman, Reynolds, and Udupa [11]), voxels which intersect the iso-surface of interest are detected; the voxel faces are rendered as a connected mesh of small squares. A mesh of triangles that approximates the iso-surface can be generated by connecting a stack of planar contours (Fuchs, Kedem, and Uselton [33]), or by finding a linear approximation to the surface inside each voxel (Lorenson and Cline's *marching cubes* technique [60]). In the *dividing cubes* method (Cline, Lorenson, Ludke, and Teeter [12]) a mesh of point primitives is generated from the iso-surface.

### 1.1.2 Volume Rendering Algorithms

Volume rendering algorithms render every voxel in the volume raster directly, without conversion to geometric primitives. They usually include an illumination model which supports semi-transparent voxels; this allows renderings where every voxel in the volume is (potentially) visible. Volume rendering algorithms can be classified according to how the image is generated. This classification groups the algorithms into *ray casting*, *shear-warp*, and *object order* methods.

**Ray Casting:** These algorithms, diagramed in Figure 1.1, cast a ray from each pixel into the volume, sample the volume at regular intervals along the ray, and then composite these samples to produce the final color seen at the pixel. This is an *image-order* computation: the resulting image is calculated pixel by pixel. They are also called *volume ray casting* techniques. They are similar to the common polygon- or surface-rendering procedure of *recursive ray tracing* [30, 96, 36], with the difference that secondary reflection and refraction rays are not spawned.

The first volume ray-casting algorithm is described by Tuy and Tuy [93]. This method only detects the first intersection of the ray with the data set (a binary classification), and so can only show the data's outer surface. It uses a simple depth cueing illumination

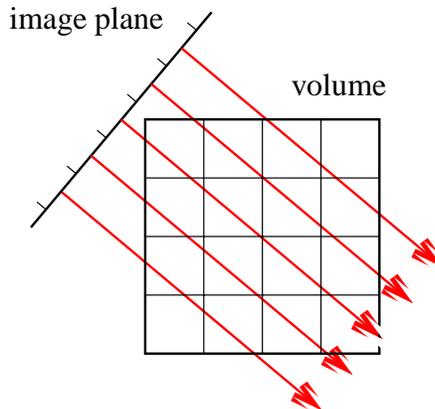


Figure 1.1: Volume ray casting.

model. Another early method from the same year is by Kajiya and Von Herzen [50], which gives a ray-tracing method for volume densities in the context of rendering clouds and other particle systems.

The next several techniques presented here trace rays through the entire volume and sample the volume raster at regular intervals along the ray. The Sabella [87] method introduces the *single-scattering shading model*, a commonly used volume-rendering illumination model. The volume is considered to be composed of many small luminescent particles. The illumination emitted by each particle may scatter off of a neighboring particle, but only one level of this scattering is modeled (otherwise the illumination calculation becomes very complex). Upson and Keeler [94] assume that the volume raster is a set of samples of a trilinearly-varying scalar field. They also introduce the idea of *transfer functions*, where each visible data parameter (such as the red, blue, green, and opacity channels) is an arbitrary function of the scalar data field. By manipulating the transfer functions it is possible to generate many different images of the same data set, and thereby gain a deeper understanding of the underlying data. The Levoy [58] ray-casting method combines many features of the above techniques. In addition, it gives non-binary methods for displaying surfaces from the scalar field.

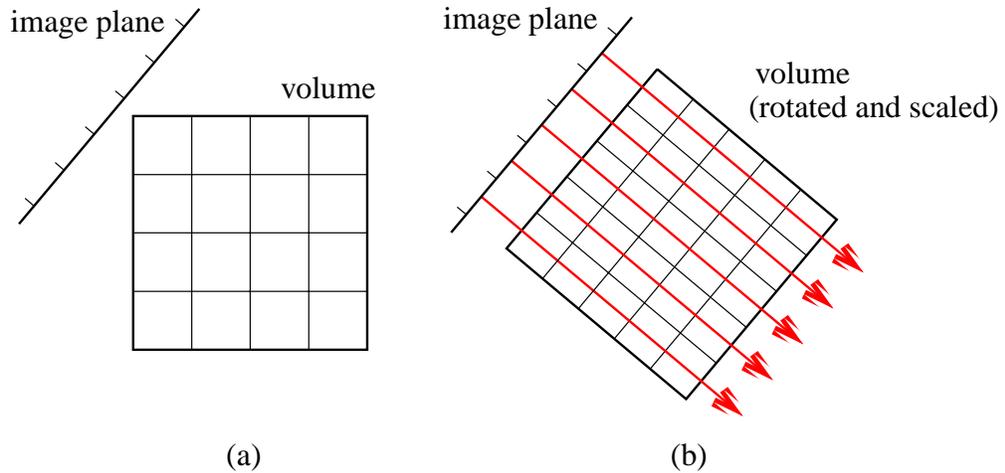


Figure 1.2: Volume rendering by affine transformation. (a) The image plane and volume before transformation. (b) After the volume is rotated and scaled, the image plane samples the volume.

**Shear-Warp:** These algorithms (Figure 1.2) first rotate and scale the volume raster to align with the image pixels (an object-order computation), and then sample the volume raster through each pixel (an image-order computation). They are also called *affine-transformation* algorithms, because the initial volume transformation is affine. They factor the viewing transformation into a sequence of 1D shears possibly separated by transpositions of the volume data. The result is a new volume raster which has been transformed so that rays projected from the image pixels regularly sample the volume in depth-ordered beams.

The technique of Drebin, Carpenter, and Hanrahan [26] composites the transformed volume data sheet-by-sheet to form the final image, where each sheet is parallel to the image plane. Hanrahan [40] gives a technique for breaking any affine volume transformation into at most three 1D shearing passes. Because this technique only works for affine transformations, it can be used for orthographic but not perspective projections. Lacroute and Levoy [54] describe a similar approach with the exception that a sheared 2D image is first produced from the sheared volume data, and then 2D resampled to form the final

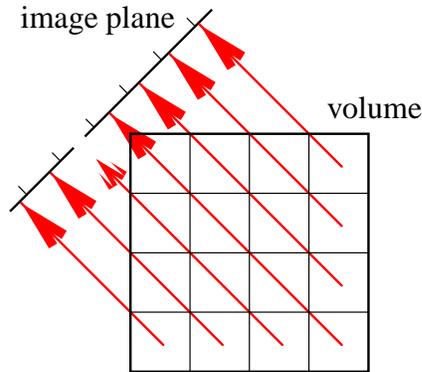


Figure 1.3: Object-order volume rendering algorithms.

image. Because a sheared image is created, the technique only requires one shearing pass through the volume data.

**Object Order:** These algorithms (Figure 1.3) project each voxel to the screen and composite it into an accumulating image. They visit the voxels in an order which ensures that visibility and occlusion between voxels is properly handled. This is an object-order computation: the resulting image is built up voxel-by-voxel. The object order volume rendering algorithms can be broken into two sub-classes depending on the type of primitive which is drawn for each voxel. This classification groups them into *splatting* and *scanline cell drawing* algorithms.

The *splatting* algorithm, given by Westover [97, 98, 99] and extended by Crawfis and Max [21], convolves each voxel with a spherically symmetric reconstruction kernel. Laur and Hanrahan [55] extend the technique to render a hierarchical volume grid stored as an octree. Mao [63] uses spherical and ellipsoidal splats to render curvilinear and irregular grids. The splatting algorithm forms the basis of the work presented in this dissertation; it is described in detail in Section 1.2 below.

*Scanline cell drawing* algorithms treat each voxel as a hexahedron or tetrahedron. They project each voxel to the view plane, decompose the voxel into a set of overlapping

polygons, and then scan-convert the polygons using standard incremental polygon rendering techniques. This last step is typically performed by polygon rendering hardware. The result is that the original voxel is 3D rasterized in a manner which is analogous to 2D rasterizing a polygon.

An early method by Frieder et al. [32] renders opaque squares; it uses binary classification and the cuberille data model [45, 11]. Upson and Keeler [94] render the voxels scanline-by-scanline. Their method creates a polygon perpendicular to the image plane; they split the resulting scanline into spans which properly integrate the voxel in depth. Wilhelms and Van Gelder [101] give an efficient table-driven technique which breaks all the possible projections of a hexahedral cell into sets of overlapping quadrilaterals and triangles. They also discuss the repercussions of scan-line interpolation on depth integration of illumination models.

The algorithms listed so far require rectilinear volumes. The same technique has been applied to irregular volumes, where voxels are typically modeled by tetrahedra, as opposed to hexahedra for regular volumes. Max, Hanrahan and Crawfis [65] give a technique for the proper incremental computation of a single-scattering shading model. Shirley and Tuchman [89] give a table-driven technique similar to Wilhelms and Van Gelder [101], which breaks all the possible projections of a tetrahedral cell into sets of overlapping triangles.

## **1.2 The Splatting Algorithm**

The work in this dissertation is demonstrated in the context of the splatting algorithm; this algorithm is now described in some detail. This description follows the implementation by Westover [97, 98, 99].

### **1.2.1 The Splatting Pipeline**

In this section splatting is described in the context of the technique's rendering pipeline. This description follows Westover's dissertation [99] with some modifications. The pipeline is given in Figure 1.4. This shows how the volume (the input data structure) is

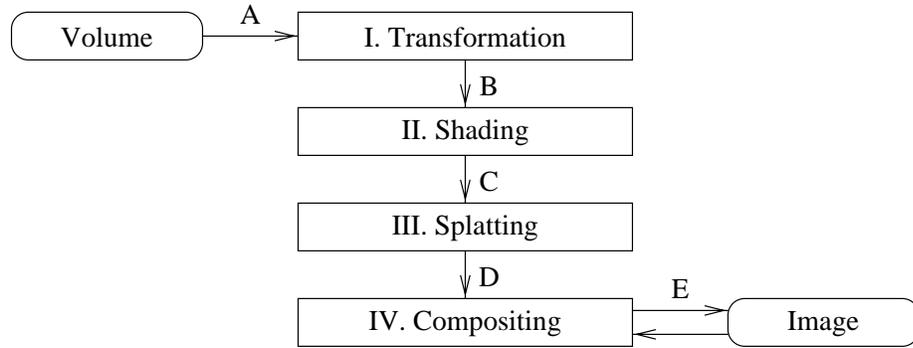


Figure 1.4: Data flow diagram of splatting (from [99]).

transformed through a 4-step rendering process to produce the final image (the output data structure).

**Volume.** The splatting algorithm starts with an input volume. The algorithm traverses the volume in either a *back-to-front* or *front-to-back* order; this ordering determines how the final composition is done in stage IV. Traversal orderings for volumetric grids are discussed in Chapter 2 of this dissertation. The output of the volume is a stream of voxels, A:

$$\begin{array}{ll}
 [d, & \text{density value} \\
 m, & \text{gradient strength} \\
 (\theta, \phi), & \text{gradient direction} \\
 (i, j, k)] & \text{volume grid coordinates}
 \end{array} \tag{1.1}$$

**Transformation (Stage I).** The first stage in the pipeline transforms the voxels from volume grid coordinates  $(i, j, k)$  into image-space screen coordinates  $(x, y, z)$ , where  $(x, y)$  is the voxel’s screen space location and  $z$  is the depth. This transformation is efficiently performed with a *3D digital differential analyzer* algorithm, which is similar to a fast raster line-drawing algorithm [30, page 74]. The first voxel in the volume is transformed by the transformation matrix using a full matrix multiplication, but all subsequent voxels are transformed by adding at most three delta values to the previously transformed value. This method is more clearly described by Machiraju and Yagel [61].

The output of stage I are image-space voxels,  $B$ :

$$\begin{aligned}
 [d, & \quad \text{density value} \\
 m, & \quad \text{gradient strength} \\
 (\theta, \phi), & \quad \text{gradient direction} \\
 (x, y, z)] & \quad \text{image-space coordinates}
 \end{aligned}
 \tag{1.2}$$

**Shading (Stage II).** The next stage shades the voxels using some shading model. Westover uses a table-driven shading process ([99], page 56) that allows a range of shading models to be used, such as Phong shading or single-scattering illumination. The opacity calculations can either emphasize surfaces in a manner similar to Levoy [58], or interior structures in a manner similar to Sabella [87].

The output of stage II are shaded image-space voxels,  $C$ :

$$\begin{aligned}
 [(r, g, b), & \quad \text{color} \\
 \alpha, & \quad \text{opacity} \\
 (x, y, z)] & \quad \text{image-space coordinates}
 \end{aligned}
 \tag{1.3}$$

**Splatting (Stage III).** This stage reconstructs the shaded image-space voxels by convolving them with a filter, and then resamples the filter at the pixels within the filter's extent. The reconstruction occurs in 3D image space, which necessitates a 3D filter kernel. However, the filter is resampled by image pixels, which happen to lie in a 2D plane. Therefore, it is possible to use a 2D filter kernel instead of a 3D filter kernel. As shown in Figure 1.5, Westover starts with a 3D rotationally symmetric filter kernel [99, page 62]. He integrates this kernel along one dimension to produce a 2D filter kernel. Westover uses a Gaussian kernel because it has convenient mathematical properties: it is rotationally symmetric, it is separable, and the integration of a multi-dimensional Gaussian along an axis is another Gaussian of one less dimension.

The splatting process is shown in Figure 1.6. The center of the splat kernel is placed on the view plane at the image space coordinates  $(x, y)$  of the image-space voxel  $C$ . Each pixel in the filter's footprint receives  $C$ 's color and opacity, weighed by the filter's value at the pixel's location. In equation form

$$p_\lambda = s_\lambda w_\lambda, \tag{1.4}$$

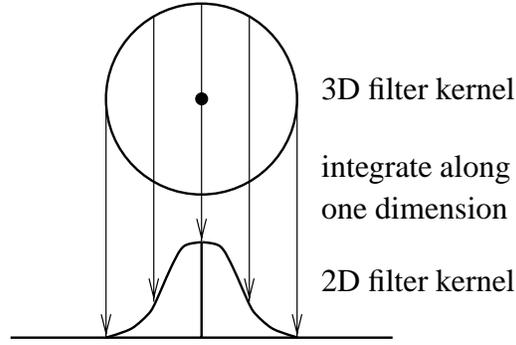


Figure 1.5: A 3D rotationally symmetric filter kernel is integrated to produce a 2D filter kernel.

where  $p$  is the pixel value,  $s$  is the splat value,  $w$  is the height of the filter above the pixel, and  $\lambda$  is one of the color channels  $r, g, b$  or  $\alpha$ .

The output of this stage are tuples  $D$  for all the pixels that lie in the splat kernel's footprint:

$$\begin{aligned} &[(r, g, b), \text{ color} \\ &\alpha, \text{ opacity} \\ &(x', y')] \text{ integer pixel coordinates} \end{aligned} \tag{1.5}$$

**Compositing (Stage IV).** This stage composites each pixel  $D$  from the splatting stage into the final image; each image pixel  $E$  has the same form as  $D$  given in Equation 1.5 above. As shown in Figure 1.4, the compositing stage accesses image pixels, composites them with the incoming splat pixels, and then returns the pixels to the image.

The compositing stage performs two types of compositing, depending on how the volume is traversed. If it is traversed in a back-to-front order, where the voxels are visited in order of decreasing distance to the view plane, then the pixels from stage III occlude those which have already been composited into the image. In this case the pixels are composited using Porter and Duff's **over** operator [81]:

$$D \text{ over } E \equiv \begin{cases} E_\lambda = D_\lambda + E_\lambda(1 - D_\alpha) \\ E_\alpha = D_\alpha + E_\alpha(1 - D_\alpha) \end{cases} \tag{1.6}$$

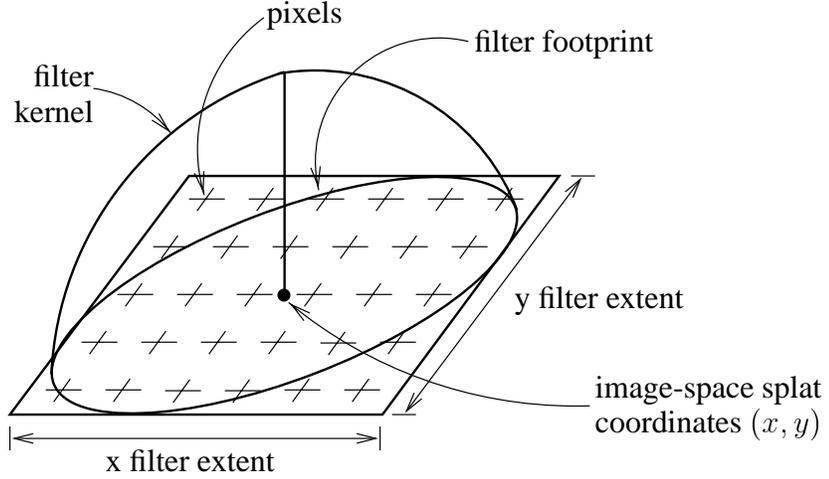


Figure 1.6: The splatting process: reconstruction and resampling with the 2D filter kernel.

where  $\lambda$  is a color channel  $r, g,$  or  $b$ . Here the new value of the image pixel  $E$  is shown to the left of the equal sign, in terms of the old value shown to the right.

If the volume is traversed in a front-to-back order, where the voxels are visited in order of increasing distance from the view plane, then pixels already in the image occlude pixels coming from stage III. In this case the pixels are composited using the **under** operator, which is the inverse of the **over** operator:

$$D \text{ under } E \equiv E \text{ over } D \equiv \begin{cases} E_\lambda = E_\lambda + D_\lambda(1 - E_\alpha) \\ E_\alpha = E_\alpha + D_\alpha(1 - E_\alpha) \end{cases} \quad (1.7)$$

**Image.** The splatting algorithm ends with the rendered image. Westover's implementation allows the user to view partially rendered images as the rendering proceeds [99, page 75]. If the splatting order is back-to-front, then features towards the back of the volume are rendered first and then disappear as they are overwritten by features towards the front of the volume. If the splatting order is front-to-back, then once the features towards the front of the volume are rendered the image does not change very much.

## 1.2.2 Inherent Problems and Solutions

The splatting algorithm as outlined in Section 1.2.1 above is Westover’s basic implementation, as presented in his first publication [97]. This implementation of splatting has several inherent problems, which are described in this section, along with some solutions that Westover proposed in his subsequent publications [98, 99].

### 1.2.2.1 Composition and Integration Problems

All volume rendering algorithms require both a *resampling* and a *visibility* operation: the 3D volume raster is resampled into a new grid (the image pixels), and a visibility operation is applied in the screen-space  $z$ -direction to calculate the correct visibility. The visibility operation is usually implemented as an approximation to a scattering model [50, 65, 87, 101] or as a composition [81].

The splatting algorithm makes several approximations to theoretically correct methods of achieving this resampling and compositing. These approximations cause three pervasive problems in splatting implementations: the *splat integration problem*, the *overlapping splat problem*, and the *splat ordering problem*. These problems and their impact are discussed in this section. Some partial solutions are given in Section 1.2.2.2 and discussed in Section 1.2.2.3.

**The Splat Integration Problem:** The first approximation is that the splatting algorithm integrates the 3D reconstruction filter kernel to produce a 2D filter kernel (Figure 1.5), when instead it should do a composition. This creates the problem shown in Figure 1.7. Consider a sight ray that intersects the 3D reconstruction kernel, and consider three small points  $a$ ,  $b$ , and  $c$  along this ray. The correct visibility ordering is the composition of the points:  $a$  **over**  $b$  **over**  $c$ . However, because the 3D kernel is integrated to a 2D kernel, the actual result is the sum of the points:  $a+b+c$ . This could be avoided if the integration were done for every splat, but for efficiency the integration is pre-computed in a pre-processing step, and the splat is convolved with a 2D and not a 3D filter kernel.

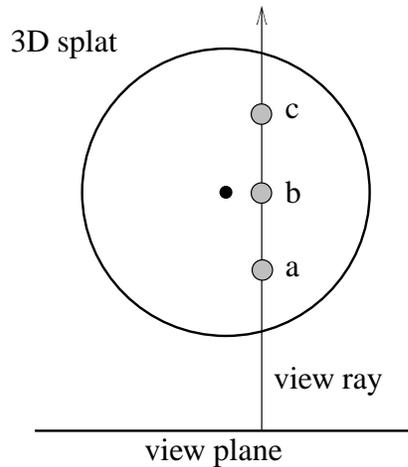


Figure 1.7: The *splat integration problem*: each splat kernel is integrated, not composited, along the viewing direction.

Despite being incorrect, the splat integration problem does not usually produce noticeable artifacts. Instead, it contributes to the common fuzzy appearance of a splatted dataset, since summing tends to wash out details.

**The Overlapping Splat Problem:** The second approximation is that the splatting algorithm does both the reconstruction and the visibility calculation on a per-splat basis. This creates the problem shown in Figure 1.8. Consider two splats, splat 1 and splat 2, and assume that splat 1 is drawn before splat 2. Each splat reconstructs the continuous function within its footprint. In the space where the splats overlap, this reconstructed function should be the *sum* of the functions reconstructed by both splats. This correct result is shown in Figure 1.8a. However, each splat is composited, and not summed, on top of the previously-rendered splats — otherwise, there would be no occlusion between splats, and the algorithm would not correctly solve the hidden surface problem. The result is shown in Figure 1.8b: splat 2 is composited on top of splat 1. This means the reconstructed function is incorrect in the space where the splats overlap.

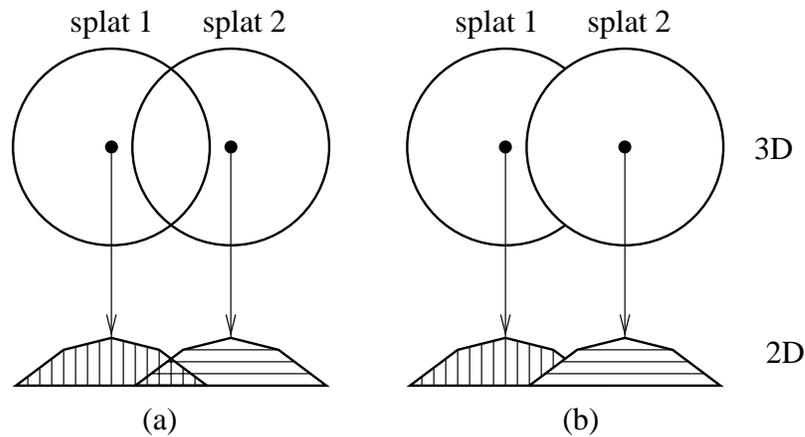


Figure 1.8: The overlapping splat problem. Assume that splat 1 is drawn before splat 2. (a) The desired result: the splat contributions are summed where they overlap. (b) The actual result: splat 2 is composited on top of splat 1, and splat 1 is occluded where the splats overlap.

This problem is widespread in splatting implementations, because in order to avoid the reconstruction artifact of *sample frequency ripple* [64] (e.g. in order for a dataset to be rendered without holes between the splats), it is necessary for the splats to overlap. An example illustrating the overlapping splat problem is given in Figure 1.13 and described in Section 1.2.2.3.

**The Splat Ordering Problem:** The overlapping splat problem interacts with the order in which the volume raster is traversed; this interaction gives rise to the *splat ordering problem*. As discussed in Section 1.2.1 above, the volume raster must be traversed in either a back-to-front or a front-to-back order (traversal orderings are discussed in Chapter 2). As shown in Figure 1.9, this causes a problem when the traversal direction changes because a corner of the volume raster rotates past the viewing plane. In Figure 1.9a the visibility ordering traverses scanlines parallel to face B. In Figure 1.9b the volume has rotated slightly, and the visibility ordering now traverses scanlines parallel to face A. This sudden change in visibility ordering causes the order in which the splats overlap to change

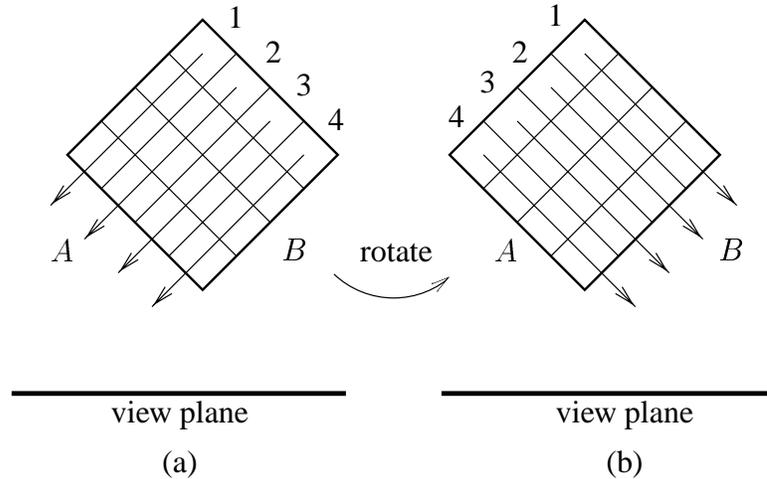


Figure 1.9: The *splat ordering problem*: a small change in the viewing parameters causes the traversal ordering to change.

throughout the entire volume, which can cause various visual artifacts. An example of the splat ordering problem is shown in Figure 1.16 in Section 1.2.2.3.

### 1.2.2.2 Summation Buffers

The source of all the problems mentioned above is that the reconstruction and visibility calculations are done on a per-splat basis. This can be addressed by using either a *volumetric summation buffer* or a *sheet summation buffer*. This section describes these techniques.

**Volumetric Summation Buffer:** The theoretically correct way to render a volume is to first reconstruct the whole volume, and then calculate the visibility of the whole volume. For splatting this can be done by using a data structure called a *volumetric summation buffer* (Figure 1.10). Assume that the volume raster has  $n \times n \times n$  resolution, and the image raster has  $m \times m$  resolution. A volumetric summation buffer is another volume raster with  $m \times m \times n$  resolution: it matches the image raster resolution in width and height and has the same depth resolution as the volume raster. As shown in Figure 1.10,

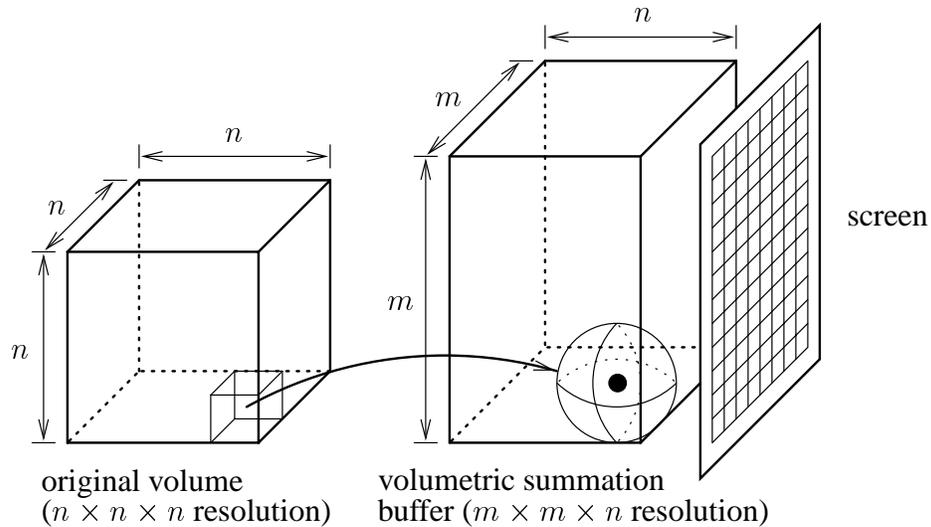


Figure 1.10: Summing the splats in a *volumetric summation buffer* before the visibility calculation.

each voxel is mapped into the summation buffer and convolved with a 3D reconstruction kernel. The kernel spreads the voxel’s energy out into the buffer. This energy is summed, not composited, into every summation buffer voxel within the kernel’s footprint. This process is repeated for every voxel in the volume raster. Next, for every pixel in the image raster the renderer performs a visibility calculation on the depth column of voxels that lie behind the pixel.

This technique calculates the reconstruction and visibility on a per-volume basis instead of a per-voxel basis, and it avoids all the problems mentioned above. Implemented in this manner, the volumetric summation buffer algorithm falls into the category of hybrid order volume rendering algorithms given in Section 1.1.2, with the exception that it uses a more accurate 3D resampling filter instead of the 1D or 2D resampling filters used in other hybrid order algorithms. Although more correct than the standard splatting algorithm, the volumetric summation buffer algorithm has two major disadvantages:

- the buffer requires a large amount of memory, and

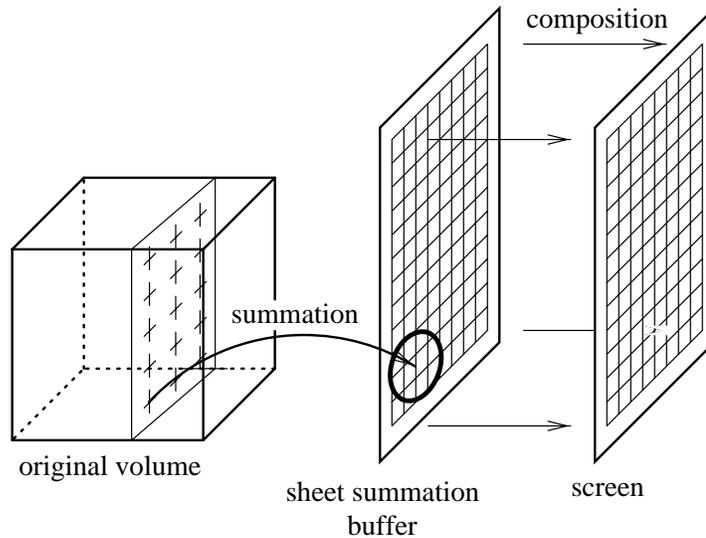


Figure 1.11: Summing the splats in a sheet summation buffer before the visibility calculation.

- the algorithm is hard to parallelize, because the buffer is a shared resource.

Westover proposes this algorithm [99], but does not implement it.

**Sheet Summation Buffer:** Many of the advantages of a volumetric summation buffer are available from reconstructing the volume one sheet at a time, and then compositing these sheets together. This is done with a data structure called a *sheet summation buffer* (Figure 1.11), which Westover proposes and implements [98, 99]. This technique requires accessing the volume in *sheets*, which are slices of the dataset which, after the viewing transformation, are as parallel as possible to the viewing plane. As shown in Figure 1.11, the voxels from each sheet are mapped into the summation buffer and convolved with a 2D reconstruction kernel. The energy from the kernel is summed, not composited, into the sheet buffer. After all the voxels in the sheet are summed, the sheet buffer is composited into the screen.

Although not as theoretically correct as a volumetric summation buffer, the sheet buffer solves the overlapping splat problem within each sheet. Furthermore, it addresses the

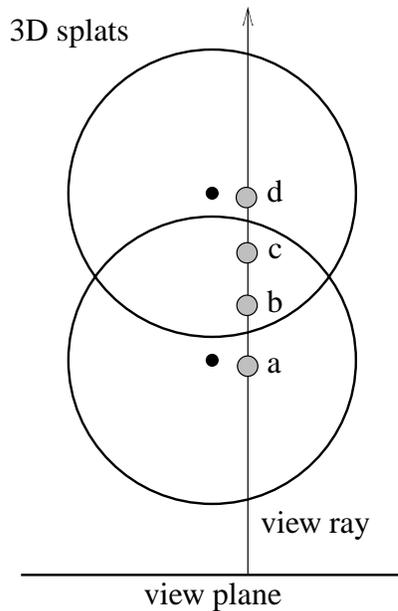


Figure 1.12: The sheet summation buffer suffers from the splat integration problem, and the splat overlap problem in the screen-space  $z$ -dimension.

volumetric buffer's two main disadvantages: it does not require a large amount of memory and it does not make the splatting algorithm difficult to parallelize. However, the sheet buffer still suffers from the splat integration problem, and it suffers from the splat overlap problem in the screen-space  $z$ -dimension.

This is shown in Figure 1.12. Consider a sight ray that intersects the two 3D reconstruction kernels, and consider four small points  $a$ ,  $b$ ,  $c$ , and  $d$  along this ray. The correct visibility ordering is the composition of the points:  $a$  **over**  $b$  **over**  $c$  **over**  $d$ , but because of integrating the splat kernel and then compositing where the kernels overlap, the actual result is  $(a + b_1 + c_1)$  **over**  $(b_2 + c_2 + d)$ .

The summed sheet buffer also suffers from two illumination problems which are described in the next section.

### 1.2.2.3 Results and Discussion

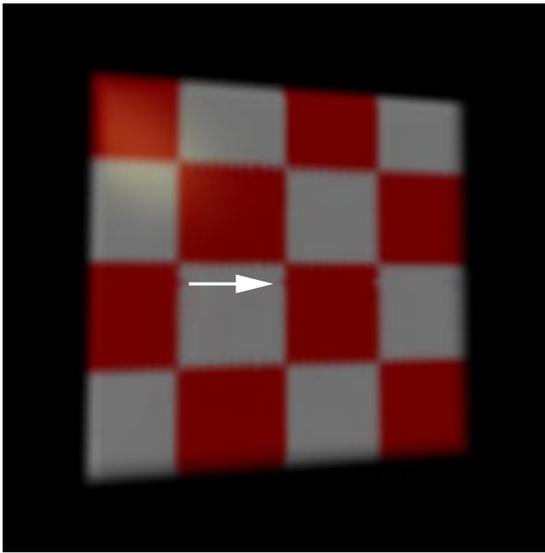
This section gives an example of the overlapping splat problem, and then shows how to mitigate the problem by using attenuated splats or a summed sheet buffer. It then demonstrates two problems with the summed sheet buffer technique.

Figure 1.13 demonstrates an example of the artifacts caused by the overlapping splat problem. It shows four frames from an animation of a rotating cube, rendered with the splatting implementation described in Section 1.2.1 and using the Perspective Back-to-Front ordering algorithm described in Chapter 2. The dataset is a  $40 \times 40 \times 40$  hollow cube texture mapped with alternating  $10 \times 10 \times 10$  red and white cubes.

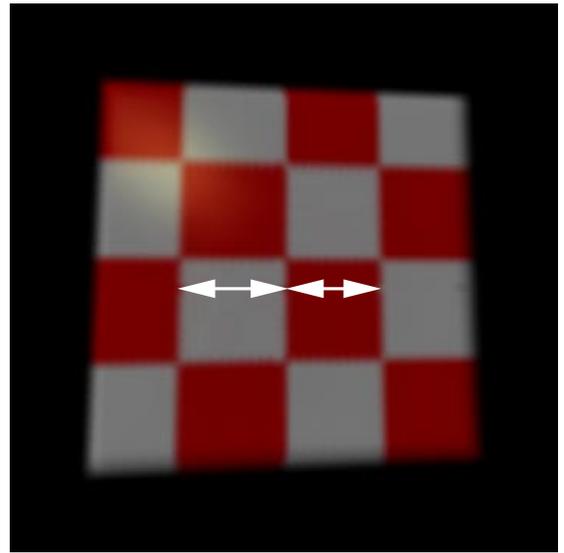
Figure 1.13 illustrates two visual artifacts caused by the overlapping splat problem. Figures 1.13a and 1.13d show a horizontal and vertical scanline artifact caused by the last scanline overlapping the previously drawn scanlines. Figures 1.13b and 1.13c show a second artifact. Note that in Figure 1.13b the marked left-hand stripe appears slightly wider than the right-hand stripe. In Figure 1.13c the situation is reversed: the left-hand stripe appears slightly narrower than the right-hand stripe. This is caused by the splats overlapping where the two stripes meet: in Figure 1.13b the left hand stripe is drawn last and so it overlaps the right-hand stripe, while in Figure 1.13c the opposite happens. Both of these problems are worse when the dataset is animated — when this happens the artifacts appear to swim across the face of the dataset in a distracting manner.

Figure 1.14 shows the same animated sequence as Figure 1.13, except that here the alpha channel of the splats has been attenuated so they appear semi-transparent — note that the sides and back face of the cube are visible through the front face. In Figure 1.14 the artifacts from Figure 1.13 are less noticeable. This is because when the splats are semi-transparent some color from the overlapped splat shows through the overlapping splat. Referring back to Figure 1.8b on page 14, if splat 1 and splat 2 are both semi-transparent then the effect is more like Figure 1.8a, no matter which splat is drawn last.

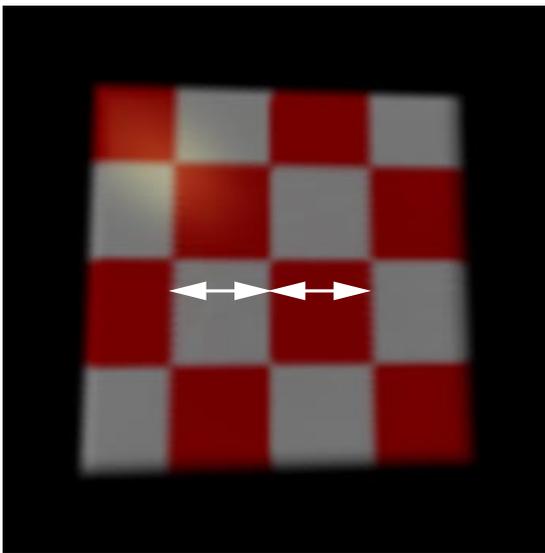
Figure 1.15 also shows the same animated sequence as Figure 1.13, except that here the splats are rendered into a sheet summation buffer. The artifacts from Figure 1.13 are no longer visible. From this it appears that the sheet summation buffer solves the



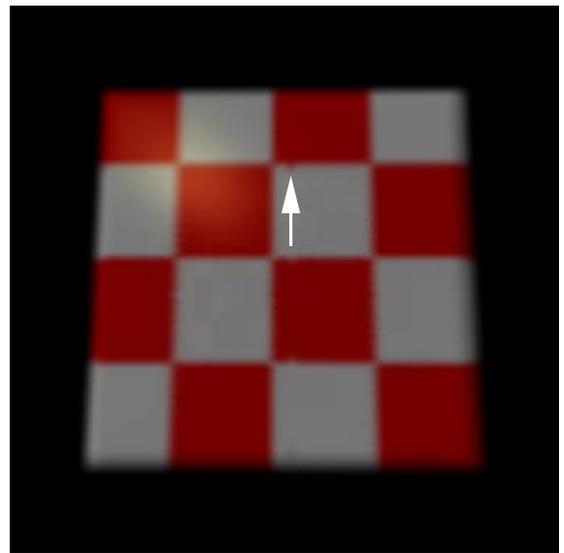
(a)



(b)

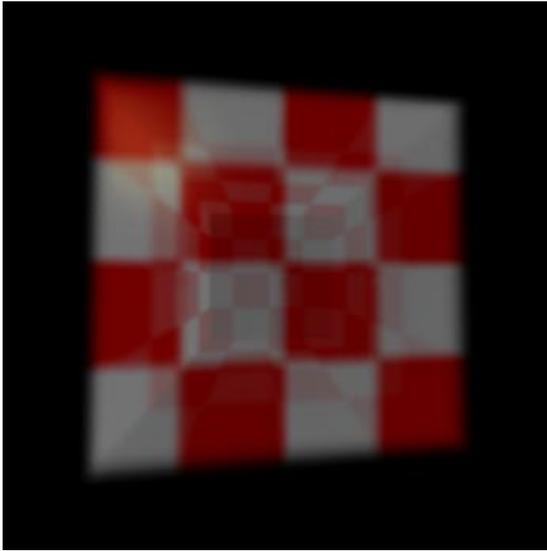


(c)

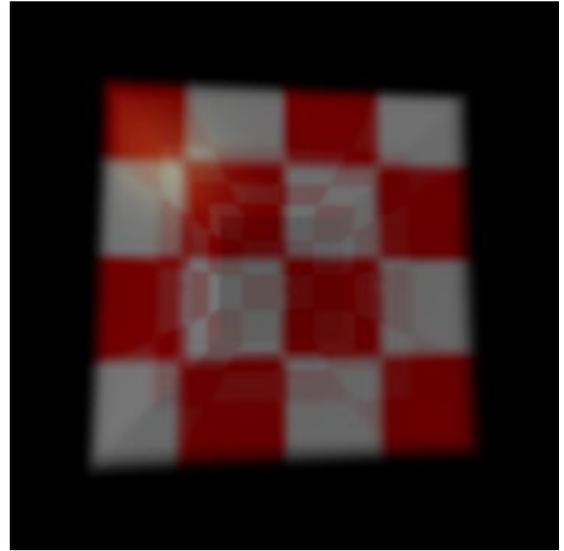


(d)

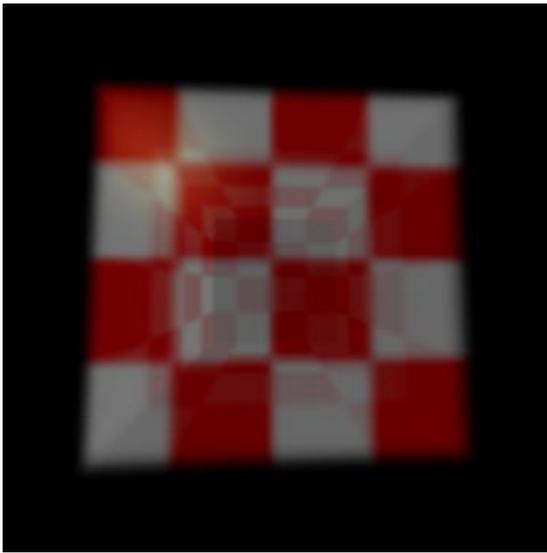
Figure 1.13: An animation illustrating the overlapping splats problem. The animation shows four frames from an animation of a rotating cube, rendered with composited splats. Artifacts from the overlapping splats problem are noted with arrows.



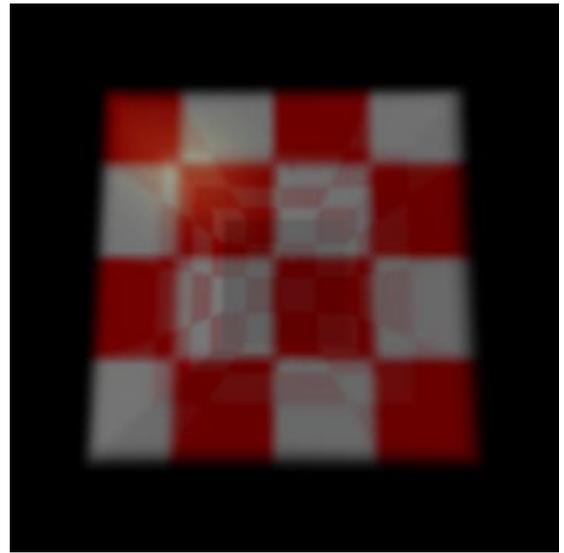
(a)



(b)



(c)



(d)

Figure 1.14: The same animation as Figure 1.13, rendered with composited attenuated splats. The artifacts from Figure 1.13 are less noticeable.

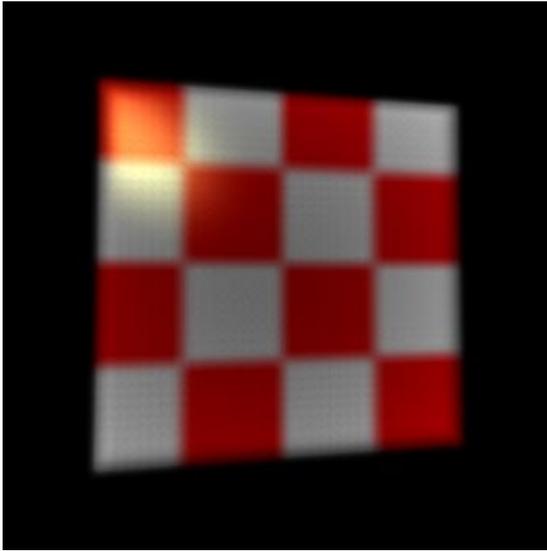
overlapping splat problem, and for the artifacts shown in Figure 1.13 it does. However, the sheet summation buffer has two illumination problems which limit its usefulness.

The first problem is that the summed sheet buffer suffers from the *splat ordering problem*, which causes an apparent illumination error. This is illustrated in Figure 1.16. Here a  $60 \times 60 \times 60$  cube texture-mapped with  $10 \times 10 \times 10$  green and red sub-cubes is shown rendered with an orthographic projection. Between Figure 1.16a and Figure 1.16b the visibility ordering changes, as illustrated in Figure 1.9 back on page 15. In Figure 1.9a the left-hand cube face is bright and the right-hand face is dim, while in Figure 1.9b the situation is reversed. This occurs because in Figure 1.9a the sheet buffer is parallel to the left-hand cube face, while in Figure 1.9b it is parallel to the right-hand cube face. In Figure 1.9a the left-hand face is bright because the splats are all summed together, while in Figure 1.9b the left-hand face becomes dimmer because now rows of splats are composited instead of summed. The converse happens to the right-hand cube face.

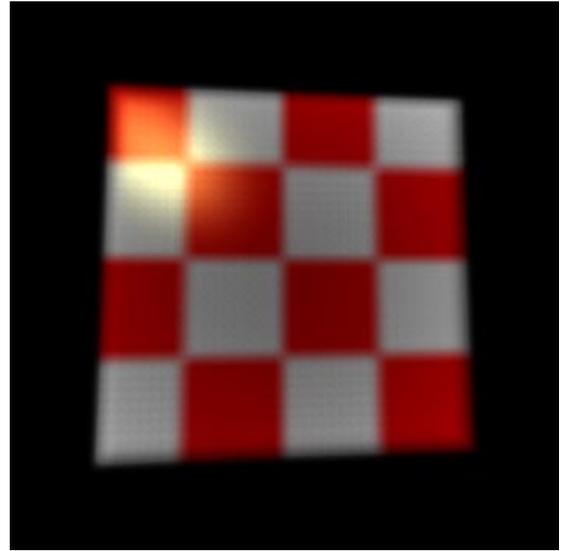
The gist of the problem is that summing the splats in sheets makes the sheets look brighter relative to the composition between the sheets. This causes what appears to be an illumination error that manifests itself as a flashing effect as the cube is rotated.

The second problem is the summed sheet buffer has an apparent illumination error when used with a perspective projection. This problem is illustrated in Figure 1.17. Here a  $40 \times 40 \times 40$  cube texture-mapped with  $10 \times 10 \times 10$  sub-cubes is shown rendered with a perspective projection. Note that the bottom face of the cube brightens as it narrows. This occurs because the perspective distortion packs more and more splats into each unit of screen area as the cube face narrows, and since the splats are summed together they become brighter. As the perspective distortion becomes greater this brightening continues until the pixels are completely saturated.

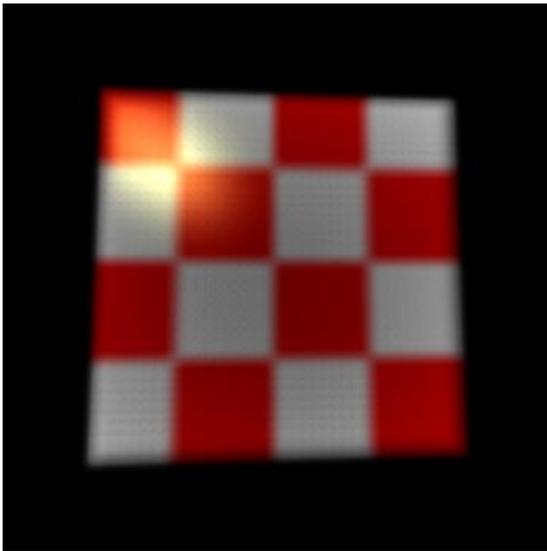
This problem does not occur with an orthographic projection because the number of splats projected onto each unit of screen area remains constant (see Figure 1.16 above). This problem — which has not been previously reported in the literature — shows that the sheet summation buffer does not properly calculate illumination for perspective projections.



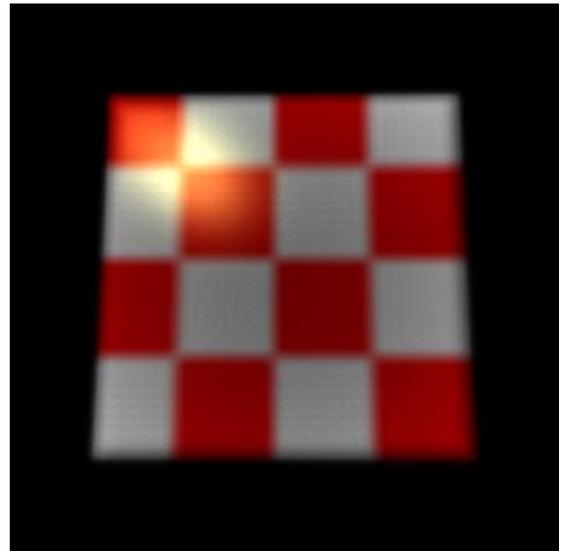
(a)



(b)

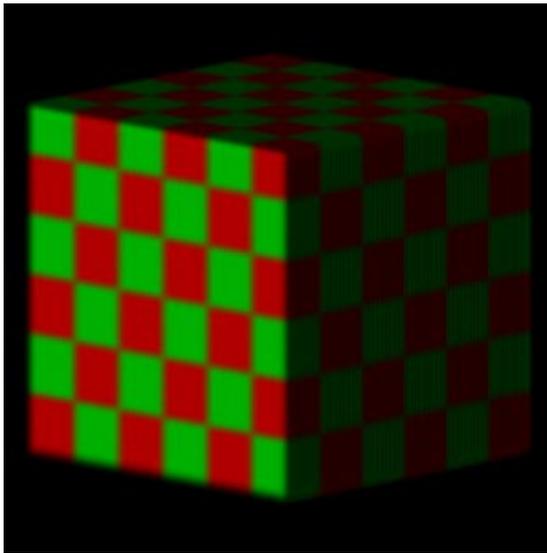


(c)

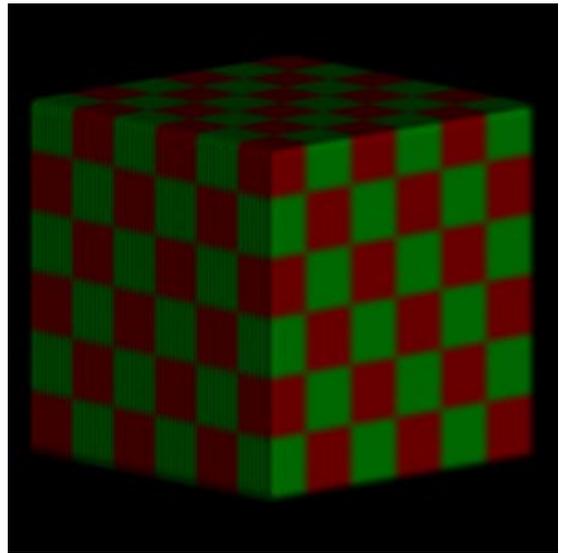


(d)

Figure 1.15: The same animation as Figure 1.13, rendered with a sheet summation buffer. There are no overlapping splat artifacts.



(a)



(b)

Figure 1.16: A sheet summation buffer showing the *splat ordering problem* (see Figure 1.9).

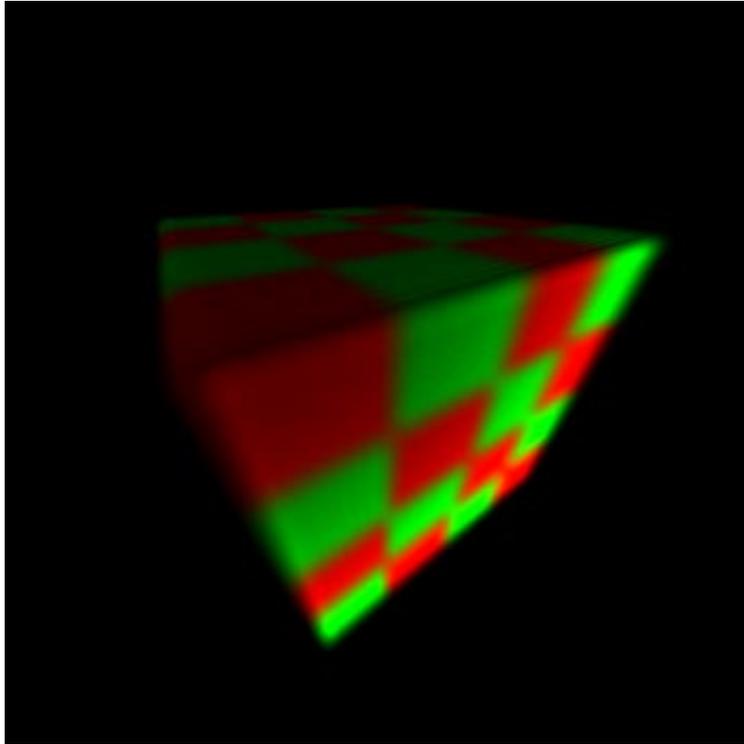


Figure 1.17: The sheet summation buffer does not calculate illumination properly for perspective projections.

### 1.2.3 Reconstruction Kernel

As noted in Section 1.2.1 above, Westover uses a Gaussian reconstruction kernel in part because it has convenient mathematical properties. However, a Gaussian is not a very good reconstruction filter because it has substantial blurring in the passband (see Marschner and Lobb [64]). Crawfis and Max [21] give a better reconstruction filter for splatting, which has the form of a piecewise cubic spline. This filter is optimal in the sense that it reconstructs a rectangular lattice of voxels of constant value with the smallest possible variance. This is the filter used for the images presented in this dissertation.

### 1.2.4 Advantages and Disadvantages of Splatting

This section compares splatting to other volume rendering algorithms. When listing the disadvantages of splatting, it distinguishes between inherent problems and those that are due to implementation inaccuracies.

#### 1.2.4.1 Advantages of Splatting

The main advantage of splatting over ray casting is its inherent speed advantage. In ray casting, reconstruction is performed for each sample point along the ray. At each sample point a  $k^3$  convolution filter is applied. Even if, on the average, each of the  $n^3$  voxels are sampled only once, ray-casting has a complexity of *at least*  $k^3 n^3$ . In splatting, on the other hand, the convolution is precomputed, and every voxel is splatted exactly once. Each splat requires  $k^2$  compositing operations. Therefore, one can expect a complexity of *at most*  $k^2 n^3$ . This gives splatting an inherent speed advantage. As a side product of this advantage, in a splatting implementation one can afford to employ larger reconstruction kernels and improve the accuracy of splatting while incurring an  $O(k^2)$  penalty instead of an  $O(k^3)$  penalty.

Because splatting is an object-order rendering algorithm, it has a trivial parallel implementation (Yagel and Machiraju [107], Westover [99]), where the volume raster is evenly divided among the processors. It is more difficult to distribute the data with ray-driven

approaches, because each ray might need to access many different parts of the volume raster.

Splatting is the preferred volume rendering technique when the desired result is an X-ray projection image instead of the usual composited image [72]. This is because the summation of pre-integrated reconstruction kernels is both faster and more accurate than ray-casting approaches, which require the summation of many discrete sums. Creating X-ray projection images from volumes is an important step in the reconstruction algorithms employed by tomographic medical imaging devices [72], such as CT and PET [7].

Splatting is trivially accelerated by just ignoring empty voxels. It can further be accelerated by extracting and storing just those voxels which contribute to the final image [106], which prevents traversing the entire volume raster. This is equivalent to similar acceleration techniques for volume ray-casting, such as space-leaping (Yagel and Shi [108]) or fitted extents (Sobierajski and Avila [90]), which accelerate ray casting by quickly traversing empty space.

Because the images are calculated in a strict front-to-back or back-to-front order, observing the partially created images can give types of insight into the data which are not available from image-order techniques. In particular, with a back-to-front ordering partial images reveal interior structures, while with a front-to-back ordering it is possible to terminate the rendering early [72].

#### **1.2.4.2 Inherent Disadvantages of Splatting**

An ideal volume renderer first performs the process of reconstruction and then the process of integration (or composition) for the entire volume. Splatting forces both reconstruction and integration to be performed on a per-splat basis. As discussed above in Section 1.2.2.1, the result is incorrect where the splats overlap, and the splats must overlap to ensure a smooth image. This problem is particularly noticeable when the traversal order of the volume raster changes during an animation [98, 99].

For efficiency reasons, in splatting both (transfer-function based) classification and shading are usually applied to the data prior to reconstruction. This is also commonly done in ray-casting [58]. However, this method will produce correct results only if both

classification and shading are linear operators. The result of employing a non-linear transfer function or illumination model may, for example, cause the appearance of pseudo-features that do not exist in the original data. Avoiding these features requires shading models which only model diffuse illumination. While for ray casting methods exist that perform classification and shading only after reconstruction to produce accurate results [71, 80], this is not possible in splatting.

### **1.2.4.3 Implementation-Based Disadvantages of Splatting**

With a ray-casting volume rendering algorithm it is easy to terminate the rays early when using a front-to-back compositing scheme, which can substantially accelerate rendering. Although not reported in the literature, early termination could potentially be implemented for splatting by employing the dynamic screen mechanism [84] (also used by [54] for shear-warp volume rendering). Also, the ray-driven splatting implementation of Mueller and Yagel [72] can support early ray termination.

While ray casting of volumes was originally implemented for both orthographic and perspective viewing, splatting was fully implemented only for orthographic viewing. Although ray casting has to include some mechanism to deal with the nonuniform reconstruction that is necessary with diverging view rays, it seems splatting needs to address several more inaccuracies. For the following discussion, it is useful to adopt the definition given by Crawfis and Max [21], Mueller and Yagel [72], and Yagel et al. [106] that views the footprint table as a polygon in world space centered at the voxel position with the pre-integrated filter kernel function texture-mapped onto it. As is described in [72], when mapping the footprint polygon onto the screen an accurate perspective splatting implementation must: (1) align the footprint polygon perpendicularly with the projector (sight ray) that goes through the polygon center; (2) perspectively project it to the screen to get its screen extent, and (3) ensure that the projector (sight ray) for every pixel that falls within this extent traverses the polygon at a perpendicular angle as well. All three conditions are violated in Westover's splatting algorithm [97]. Mueller and Yagel [72] give a voxel-driven splatting approach that takes care of conditions (1) and (2), and a ray-driven approach that fulfills all three conditions.

### 1.2.5 Splatting Implementation

The work in this dissertation is built on top of a modified version of the 'splat renderer' reported by Yagel et al. [106]. It make use of rendering hardware to quickly draw the splats, in a manner similar to Crawfis and Max [21]. For each splat it draws a polygon in world space centered at the voxel position. The polygon is rotated so it is perpendicular to the ray passing from the eye point through the voxel position. The splat kernel is pre-computed and stored in a  $256 \times 256$  table which is texture mapped onto the polygon by the rendering hardware.

For a given volume the renderer extracts and stores a subset of the voxels. For each voxel it evaluates a transfer function  $t = F(\nabla, \rho)$ , where  $\nabla$  and  $\rho$  are the gradient and density of the voxel; it includes the voxel in the subset if  $t$  exceeds a user-defined threshold. It stores this volume subset as a 2D array of splat rows, where each row contains only the extracted voxels. Each row is implemented as an array of voxels, but the voxels are not necessarily contiguous, and so the renderer must also store each voxel's location and normal vector. In general each row may contain a different numbers of voxels. Despite not storing the empty voxels, this data structure can still be traversed in either a back-to-front or front-to-back order.

## CHAPTER 2

# A VISIBILITY ORDERING ALGORITHM FOR RECTILINEAR GRIDS

### 2.1 Introduction

Among the earliest rendering algorithms were those that rely on sorting the geometric primitives according to their distance to the viewing plane [91]. These algorithms solve the hidden surface problem by visiting the primitives in depth order, from farthest to nearest, and scan-converting each primitive into the screen buffer. Because of the order in which the primitives are visited, closer objects overwrite farther objects, which solves the hidden surface problem (at least as long as the primitives do not form a visibility cycle). Such methods are referred to as *painter's algorithms* and as *list-priority algorithms*.

By definition, any painter's algorithm requires sorting the primitives. In the general case it takes  $O(n \log n)$  time to sort  $n$  primitives. However, because of their structure, volume grids often afford a trivial sort, which simply involves indexing the volume elements in the proper order. Such sorts take only  $O(n)$  time for  $n$  voxels. In this chapter such an indexing scheme is referred to as a *visibility ordering* of a grid.

This chapter gives a visibility ordering for a rectilinear volume grid, termed the *perspective back-to-front (PBTF)* method, which is correct for both orthographic and perspective projections. To date the most widely used visibility orderings for volume rendering are the *back-to-front (BTF)* method and what this chapter shall call the *Westover back-to-front (WBTF)* method (both are defined in Section 2.2 below). While both of

these visibility orderings are correct for orthographic projections of volumes, they are not correct for perspective projections. This fact, which is demonstrated in Section 2.4 below, is not well known in the volume rendering literature.

The main contribution of this chapter is a formal proof of the correctness of the perspective back-to-front method. While the PBTF visibility ordering has been presented in the literature [4, 68], a proof of correctness for the method has not been previously given. Furthermore, the method is not widely known in the volume rendering literature.

This chapter is organized as follows. Section 2.2 reviews previous work in the area of grid visibility orderings. Section 2.3 defines the PBTF visibility ordering, and gives a proof of correctness. Section 2.4 compares the PBTF method with the BTF and the WBTF methods, and shows perspective views which only the PBTF method is able to render correctly. Section 2.5 finishes the chapter by listing some areas of future work.

## 2.2 Previous Work

This section reviews previous work in the area of grid visibility orderings. First previous work in ordering regular grids is presented. This includes two well-known and popular ordering approaches: the *back-to-front (BTF)* method, and another method which has been called various things but which is called the *Westover back-to-front (WBTF)* method in this chapter. Next, two references which give a correct regular grid traversal for perspective projections are discussed. The section concludes with a discussion about visibility orderings for octrees and grids of convex polyhedra.

### 2.2.1 Regular Grids

Frieder et al. give the *back-to-front (BTF)* visibility ordering [32]. Figure 2.1 shows pseudocode for this ordering. It is based on the simple observation that, given a volume grid and a view plane, for each grid axis there is a traversal direction that visits the voxels in order of decreasing distance to the view plane. A 2D example of this is given in Figure 2.2. Here the origin is the farthest point from the view plane, and traversing the grid in order

```

choose  $z_{min}$ ,  $z_{max}$ ,  $y_{min}$ ,  $y_{max}$ ,  $x_{min}$ ,  $x_{max}$  according to view direction
for  $z = z_{min}$  to  $z_{max}$ 
  for  $y = y_{min}$  to  $y_{max}$ 
    for  $x = x_{min}$  to  $x_{max}$ 
      splat( voxel[ $x,y,z$ ] )

```

Figure 2.1: Pseudocode for the back-to-front visibility ordering [32].

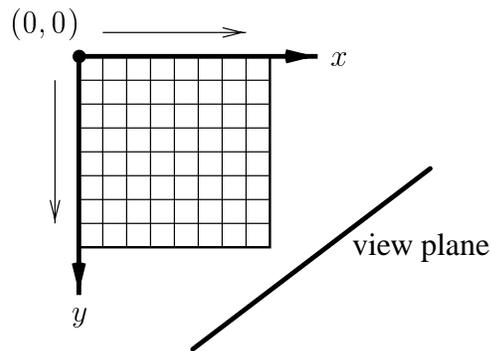


Figure 2.2: A 2D example of the BTF visibility ordering.

of increasing values of  $x$  and  $y$  always visits voxels in order of decreasing distance. This extends naturally to 3D.

The choice of which index changes fastest can be arbitrary — although Figure 2.1 shows the ordering  $z$ ,  $y$  and then  $x$ , any permutation can be used. This means the BTF method can support efficient methods of accessing the volume data by taking into account how the data is stored on disk. It also allows rendering when only some slices but not the entire volume will fit into memory.

Frieder et al. [32] only test the BTF method with orthographic projections, which provides convenient views given their application domain of biomedical visualization. While the BTF ordering correctly renders orthographic projections, it turns out not to work for perspective projections. This is demonstrated in Figure 2.25 in Section 2.4.

```

let  $k$  = axis most perpendicular to view plane
let  $j$  = axis next most perpendicular to view plane
let  $i$  = axis most parallel to view plane
choose  $kmin, kmax, jmin, jmax, imin, imax$  according to view direction

for  $k = kmin$  to  $kmax$ 
  for  $j = jmin$  to  $jmax$ 
    for  $i = imin$  to  $imax$ 
      splat( voxel[ $i,j,k$ ] )

```

Figure 2.3: Pseudocode for the Westover back-to-front visibility ordering [98, 99].

Westover gives the *Westover back-to-front (WBTF)* visibility ordering [98, 99]. Pseudocode for this ordering is given in Figure 2.3. The WBTF ordering is similar to the BTF ordering in that each grid axis is traversed in the direction that visits voxels in order of decreasing distance to the view plane. The algorithm goes farther than the BTF technique in that it also chooses a permutation of the grid axes, such that the slowest changing axis (the one in the outermost loop) is the axis most *perpendicular* to the view plane, while the quickest changing axis is the one most *parallel* to the view plane. The axis in the outermost loop chooses slices of the data set which are perpendicular to one of the grid axes. The WBTF method chooses those slices which, after the view transformation, are closer to being parallel to the view plane than either of the other two slice directions.

In his publications [98, 99] Westover's application domain is also biomedical visualization, and many of his images are rendered using an orthographic perspective. However, Westover also describes an implementation for perspective projections, and while it is true that his method works for a greater set of possible viewpoints than the BTF method, the WBTF ordering turns out not to work for all perspective projections. This is demonstrated in Figure 2.26 in Section 2.4.

Upson and Keeler give the *V-BUFFER* traversal method [94]. Pseudocode for this ordering is given in Figure 2.4. This ordering is similar to the WBTF traversal in that it processes the volume data in slices which are as parallel as possible to the view plane

**let**  $k$  = axis most perpendicular to view plane  
choose  $kmin$ ,  $kmax$  according to view direction

**for**  $k = kmin$  **to**  $kmax$   
  **foreach**  $j, i$  from a concentric sweep about the slice  $k$   
    splat( voxel[ $i,j,k$ ] )

Figure 2.4: Pseudocode for the V-BUFFER visibility ordering [94].

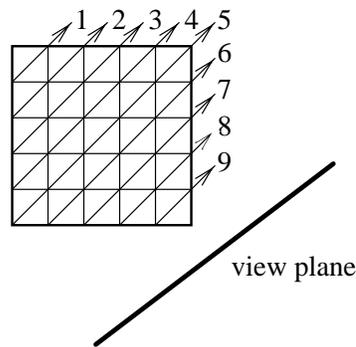


Figure 2.5: The V-BUFFER visibility ordering within each slice.

— hence the orderings have the same outermost loop. The V-BUFFER traversal differs in that it uses a more complicated “concentric sweep” to order the voxels in each slice. Although Upson and Keeler do not give the full details, the sweep strictly orders the voxels according to increasing distance from the view plane. An example of the ordering for one slice is shown in Figure 2.5: when the viewing plane is beyond one corner of the slice, the voxels are visited in the diagonal pattern shown. This differs from the WBTF ordering, which accesses each slice in a scanline fashion, and hence does not visit the voxels in a strict distance ordering.

Although Upson and Keeler [94] do not discuss whether their method supports perspective as well as orthographic projections, the visibility performance of the ordering

should be equivalent to the WBTF ordering. This is because the visibility problems demonstrated in Figure 2.26 for the WBTF method arise from the traversal direction of the outermost loop, which is the same for the two techniques. Hence the V-BUFFER method should suffer the same visibility problem shown in Figure 2.26 as the WBTF method (although the V-BUFFER ordering was not tested as part of this dissertation).

### 2.2.2 Correct Regular Grid Perspective Ordering Methods

There are two references that give a regular grid visibility ordering which is correct for perspective projections. One is by Anderson [4], which describes the visibility ordering in the context of rendering 2D grid surfaces. This ordering is equivalent to the 2D ordering given in Section 2.3.3. The Anderson reference inspired the research reported in this chapter.

The second reference is by Max [68], which describes the visibility ordering in the context of rendering a multi-resolution grid which results from a technique for solving partial differential equations called *adaptive mesh refinement*. This ordering is equivalent to the 3D ordering given in Section 2.3.4.

Although both of these references give the same visibility ordering as the perspective back-to-front (PBTF) ordering that is described in this chapter, this chapter still makes the following unique contributions:

- It gives a rigorous argument for the correctness of the PBTF method for perspective projections.
- It demonstrates that two well-know visibility orderings, the BTF and the WBTF methods, are in fact not correct for perspective projections.
- It presents the PBTF ordering method in the context of splatting, which is a common volume rendering method that requires a visibility ordering.

### 2.2.3 Non-Regular Grids

Aref and Samet [5] give a back-to-front or front-to-back traversal of an octree for a perspective projection. Because a non-leaf octree node can be thought of as a  $2 \times 2 \times 2$

grid, it is easy to enumerate all the possible visibility orderings between the node and an arbitrary view point. The technique is based on a recursive octree traversal along with a table-lookup visibility computation.

Williams [103] gives a linear-time ordering algorithm for grids composed of acyclic convex polyhedra. A similar technique is also reported by Max et al. [65]. Both involve constructing a directed graph which represents the occlusion between adjacent cells, and then topologically sorting the resulting graph.

## 2.3 The Perspective Back-to-Front Visibility Ordering

This section gives the perspective back-to-front (PBTF) visibility ordering algorithm, and a proof of its correctness. The proof is constructive, so it yields a formal description of the PBTF visibility ordering. The ordering is presented and proved in three parts: first for a 1D grid, then a 2D grid, and then finally a 3D grid. In order to simplify the presentation the proof is given in terms of a front-to-back ordering, but it is trivial to generate the equivalent back-to-front ordering by simply reversing the ordering given.

Here is a brief, intuitive overview of the PBTF visibility ordering method:

**1D Visibility Ordering:** Project the view point onto the 1D grid, partitioning it into two halves (Figure 2.10). Traverse each half in order of increasing distance from the view point.

**2D Visibility Ordering:** Project the view point onto the 2D grid, partitioning it into four quadrants (Figure 2.15). Traverse each quadrant in a scanline fashion in order of increasing distance from the view point.

**3D Visibility Ordering:** Project the view point onto the 3D volume, partitioning it into four quadrants (Figure 2.22). Traverse each quadrant in a scanline fashion in order of increasing distance from the view point.<sup>1</sup>

<sup>1</sup>This gives the visibility ordering for the familiar case where the view point is outside of the volume. The proof deals with the more general case where the view point is located inside the volume.

The key idea of the PBTF ordering over the BTF and WBTF orderings is that the projection of the view point onto the grid must be taken into account in order to properly compensate for the perspective distortion. Methods such as the BTF and the WBTF orderings, which try to find a single traversal direction for each grid axis, do not have enough degrees of freedom in this choice to properly compensate for the perspective distortion. This is further demonstrated in Section 2.4, which compares the performance of the three orderings.

The organization of this section is as follows: First, Section 2.3.1 lists various definitions and assumptions which are used during the rest of the proof. Then Section 2.3.2 gives the 1D visibility ordering, followed by the 2D ordering in Section 2.3.3 and the 3D ordering in Section 2.3.4. Finally, Section 2.3.5 discusses the proof.

### 2.3.1 Definitions and Assumptions

We assume that we have a scene composed of a set of *point objects*  $P$  that we wish to render. Point objects are luminous points that have no discernible dimension; mathematically a point object is a tuple  $\mathbf{p} = (\mathbf{c}, \mathbf{x})$ , where  $\mathbf{c}$  is a *color vector* and  $\mathbf{x}$  is a *location vector*. A color vector is a tuple  $\mathbf{c} = (\lambda_1, \lambda_2, \dots, \lambda_n, \alpha)$ , where  $\lambda_1, \lambda_2, \dots, \lambda_n$  are different colors (in practice usually only three colors — red, green, and blue — are specified), and  $\alpha$  is the opacity of the object. A location vector is a tuple  $\mathbf{x} = (x, y, z)$  which gives the three-dimensional location of the point object. This definition of a point object is similar to Reeve’s definition [83] of the particles that make up a particle system.

A *viewpoint*  $\mathbf{vp} = (\mathbf{vp}_x, \mathbf{vp}_y, \mathbf{vp}_z)$  is a location in  $\mathcal{R}^3$  from which the scene is viewed. Relative to  $\mathbf{vp}$  we can define the *obstructs* relation “ $\parallel$ ” as follows. Let  $\mathbf{p}_1, \mathbf{p}_2$ , and  $\mathbf{p}_3$  be point objects. If  $\mathbf{vp}, \mathbf{p}_1$ , and  $\mathbf{p}_2$  lie on a straight line such that  $\mathbf{p}_1$  is closer to  $\mathbf{vp}$  than  $\mathbf{p}_2$ , then  $\mathbf{p}_1 \parallel \mathbf{p}_2$  (Figure 2.6). If a point object does not obstruct another point object we write “ $\not\parallel$ ”. Note that in Figure 2.6  $\mathbf{p}_2 \not\parallel \mathbf{p}_1$ , and because they do not lie on a straight line with  $\mathbf{vp}$  both  $\mathbf{p}_1 \not\parallel \mathbf{p}_3$  and  $\mathbf{p}_3 \not\parallel \mathbf{p}_1$ .

A set of point objects can be put into a *visibility ordering*, such that object  $\mathbf{p}_1$  is rendered in front of  $\mathbf{p}_2$  iff  $\mathbf{p}_2 \not\parallel \mathbf{p}_1$ . The phrase “is rendered in front of” can either mean that object  $\mathbf{p}_1$  or object  $\mathbf{p}_2$  is rendered first, depending on how each point object is composited

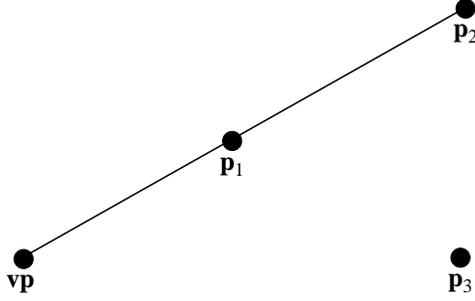


Figure 2.6: The obstructs relation.  $\mathbf{p}_1 \parallel \mathbf{p}_2$ , but  $\mathbf{p}_1 \not\parallel \mathbf{p}_3$ .

into the accumulating image. We refer to this order as *front-to-back* or *back-to-front*. In a front-to-back visibility ordering, the objects closest to  $\mathbf{vp}$  are rendered first, and so once a point object has been composited into the image it occludes all subsequent objects. In a back-to-front visibility ordering, the objects farthest from  $\mathbf{vp}$  are rendered first, and so each point object occludes those which have already been composited into the image. For a back-to-front traversal each object is composited using Porter and Duff’s **over** operator [81], while a front-to-back traversal uses the **under** operator. If the object  $\mathbf{p}_j$  is composited into the pixel  $\mathbf{c} = (\lambda, \alpha)$ , then

$$\mathbf{p}_j \text{ over } \mathbf{c} \equiv \begin{cases} \lambda_{\text{new}} = \lambda_j + \lambda_{\text{old}}(1 - \alpha_j) \\ \alpha_{\text{new}} = \alpha_j + \alpha_{\text{old}}(1 - \alpha_j) \end{cases}, \quad (2.1)$$

and

$$\mathbf{p}_j \text{ under } \mathbf{c} \equiv \mathbf{c} \text{ over } \mathbf{p}_j \equiv \begin{cases} \lambda_{\text{new}} = \lambda_{\text{old}} + \lambda_j(1 - \alpha_{\text{old}}) \\ \alpha_{\text{new}} = \alpha_{\text{old}} + \alpha_j(1 - \alpha_{\text{old}}) \end{cases}, \quad (2.2)$$

where  $(\lambda_{\text{new}}, \alpha_{\text{new}})$  are the new values of  $\mathbf{c}$  after compositing  $\mathbf{p}_j$ , defined in terms of the old values  $(\lambda_{\text{old}}, \alpha_{\text{old}})$ . In this section, unless otherwise specified a front-to-back traversal is assumed.

If the objects  $\mathbf{p}_j$  and  $\mathbf{p}_k$  are point objects, then an absolute visibility ordering (with respect to some viewpoint) exists between them, because point objects either wholly occlude or wholly do not occlude each other. This is not the case with larger objects —

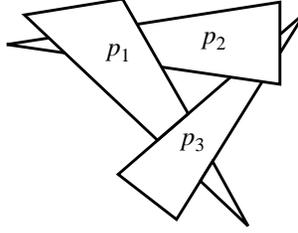


Figure 2.7: An example of three polygons which do not have a visibility ordering.

Figure 2.7 shows an example of three polygons  $p_1$ ,  $p_2$ , and  $p_3$ , where part of each polygon occludes a part of another polygon. From the viewpoint of this figure, no visibility ordering exists for these three polygons.

If a visibility ordering exists between objects  $\mathbf{p}_1$  and  $\mathbf{p}_2$ , we can express this ordering with a *visibility ordering operator*:

$$\mathbf{p}_1 \Rightarrow \mathbf{p}_2, \quad (2.3)$$

where “ $\Rightarrow$ ” (read: “visited before”) is a relation which indicates that object  $\mathbf{p}_1$  is visited before object  $\mathbf{p}_2$  during a visibility ordering. For a back-to-front ordering this means  $\mathbf{p}_1$  is farther than  $\mathbf{p}_2$  from  $\mathbf{vp}$ , while for a front-to-back ordering  $\mathbf{p}_1$  is closer than  $\mathbf{p}_2$  to  $\mathbf{vp}$ .

We also use the *visibility ordering delimiter*  $\langle \rangle$  to indicate a visibility ordering among sequential objects. For example,

$$\langle \mathbf{p}_j : 1 \leq j \leq 3 \rangle \quad (2.4)$$

indicates

$$\mathbf{p}_1 \Rightarrow \mathbf{p}_2 \Rightarrow \mathbf{p}_3. \quad (2.5)$$

Visibility ordering delimiters can be nested, so

$$\langle \langle \mathbf{p}_{i,j} : 1 \leq i \leq 3 \rangle : 1 \leq j \leq 2 \rangle \quad (2.6)$$

indicates

$$\mathbf{p}_{1,1} \Rightarrow \mathbf{p}_{2,1} \Rightarrow \mathbf{p}_{3,1} \Rightarrow \mathbf{p}_{1,2} \Rightarrow \mathbf{p}_{2,2} \Rightarrow \mathbf{p}_{3,2}. \quad (2.7)$$

The visibility delimiter may also be applied to a set. If set  $A = \{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3\}$ , then  $\langle A \rangle$  indicates the contents of  $A$  placed in a visibility order, e.g.:  $\langle A \rangle = \langle \mathbf{p}_j : 1 \leq j \leq 3 \rangle = \mathbf{p}_1 \Rightarrow \mathbf{p}_2 \Rightarrow \mathbf{p}_3$ .

When dealing with nested visibility ordering delimiters, it will be common that the order of the iteration variables can be permuted without changing the overall visibility ordering. For example, for the set of point objects referenced in Equation 2.6 it could be the case that either of the following orderings is correct:

$$\begin{aligned} & \langle \langle \mathbf{p}_{i,j} : 1 \leq i \leq 3 \rangle : 1 \leq j \leq 2 \rangle \\ \text{or} & \langle \langle \mathbf{p}_{i,j} : 1 \leq j \leq 2 \rangle : 1 \leq i \leq 3 \rangle. \end{aligned} \quad (2.8)$$

In such situations, as a notational convenience we shall list just one of the visibility orderings and then refer to the other as the first ordering's *dual*, formed by a permutation of the iteration variables. For the example above we would write:

$$\langle \langle \mathbf{p}_{i,j} : 1 \leq i \leq 3 \rangle : 1 \leq j \leq 2 \rangle \text{ or } \textit{dual}, \quad (2.9)$$

where the term “*dual*” is just a notation for indicating the same visibility ordering with a permutation of the iteration variables.

Visibility ordering operators nested three deep will also be common. In such cases a statement like

$$\langle \langle \langle \mathbf{p}_{i,j,k} : 1 \leq i \leq 3 \rangle : 1 \leq j \leq 2 \rangle : 1 \leq k \leq 3 \rangle \text{ or } \textit{duals} \quad (2.10)$$

is shorthand for

$$\begin{aligned} & \langle \langle \langle \mathbf{p}_{i,j,k} : 1 \leq i \leq 3 \rangle : 1 \leq j \leq 2 \rangle : 1 \leq k \leq 3 \rangle \\ \text{or} & \langle \langle \langle \mathbf{p}_{i,j,k} : 1 \leq i \leq 3 \rangle : 1 \leq k \leq 3 \rangle : 1 \leq j \leq 2 \rangle \\ \text{or} & \langle \langle \langle \mathbf{p}_{i,j,k} : 1 \leq j \leq 2 \rangle : 1 \leq i \leq 3 \rangle : 1 \leq k \leq 3 \rangle \\ \text{or} & \langle \langle \langle \mathbf{p}_{i,j,k} : 1 \leq j \leq 2 \rangle : 1 \leq k \leq 3 \rangle : 1 \leq i \leq 3 \rangle \\ \text{or} & \langle \langle \langle \mathbf{p}_{i,j,k} : 1 \leq k \leq 3 \rangle : 1 \leq i \leq 3 \rangle : 1 \leq j \leq 2 \rangle \\ \text{or} & \langle \langle \langle \mathbf{p}_{i,j,k} : 1 \leq k \leq 3 \rangle : 1 \leq j \leq 2 \rangle : 1 \leq i \leq 3 \rangle. \end{aligned} \quad (2.11)$$

### 2.3.1.1 Dividing Property of a Plane

In the sections that follow, we make use of the *dividing property* of a plane (see Figure 2.8). Consider a plane  $P$ . It divides space into two half-spaces  $P^+$  and  $P^-$ . Assume

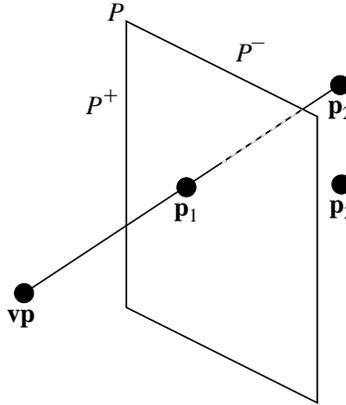


Figure 2.8: The plane  $P$  divides the space into two half-spaces  $P^+$  and  $P^-$ .  $\{vp, p_1\} \in P^+$ , and  $\{p_2, p_3\} \in P^-$ .

without loss of generality that the view point  $vp$  is in  $P^+$ , and also assume without loss of generality that compositing is front-to-back. Consider any point objects  $p_1$  in  $P^+$  and  $p_2, p_3$  in  $P^-$  such that  $p_1 \parallel p_2$  and  $p_1 \not\parallel p_3$ . Because these are point objects, they must lie entirely in one of the half-spaces  $P^+$  or  $P^-$ ; they cannot straddle the plane and lie in both. Clearly  $p_1 \Rightarrow p_2$  and  $p_1 \Rightarrow p_3$  both hold. Even though  $p_3 \Rightarrow p_1$  also holds (because  $p_1$  and  $p_3$  do not obstruct each other), as this example shows we can define the visibility ordering operator to always order objects according to increasing distance from  $vp$ . Because all objects in  $P^+$  are closer to  $vp$  than all objects in  $P^-$ , we may state

$$P^+ \Rightarrow P^-, \quad (2.12)$$

or that all point objects in  $P^+$  are rendered before all point objects in  $P^-$ . This can be extended to more elaborate shapes through the intersection and union of similar half-spaces.

### 2.3.1.2 1D, 2D, and 3D Grids

In this chapter we are most interested in 1D, 2D, and 3D *rectilinear grids* of point objects. A 1D rectilinear grid of point objects (Figure 2.9a) is a set  $\{P, U\}$ .  $P$  is a sequence of

point objects  $(\mathbf{p}_1, \dots, \mathbf{p}_n)$  that lie along a straight line which we call the  $\mathbf{u}$  axis.  $\mathbf{U}$  is a spacing vector  $(u_1, \dots, u_{n-1})$ , where  $u_i$  gives the distance between  $\mathbf{p}_i$  and  $\mathbf{p}_{i+1}$ .

A 2D rectilinear grid of point objects (Figure 2.9b) is a set  $\{\mathbf{P}, \mathbf{U}, \mathbf{V}\}$ .  $\mathbf{P}$  is a 2D sequence of point objects  $(\mathbf{p}_{i,j} : 1 \leq i \leq n, 1 \leq j \leq m)$  which lie on a two-dimensional rectilinear grid  $\mathbf{uv}$ , which we call the  $\mathbf{uv}$  plane.  $\mathbf{U}$  is a spacing vector  $(u_1, \dots, u_{n-1})$ , where  $u_i$  gives the distance between the grid lines  $\mathbf{u}_i$  and  $\mathbf{u}_{i+1}$ , and similarly  $\mathbf{V}$  is a spacing vector  $(v_1, \dots, v_{m-1})$ , where  $v_j$  gives the distance between the grid lines  $\mathbf{v}_j$  and  $\mathbf{v}_{j+1}$ .

And similarly, a 3D rectilinear grid of point objects (Figure 2.9c) is a set  $\{\mathbf{P}, \mathbf{U}, \mathbf{V}, \mathbf{W}\}$ .  $\mathbf{P}$  is a 3D sequence of point objects  $(\mathbf{p}_{i,j,k} : 1 \leq i \leq n, 1 \leq j \leq m, 1 \leq k \leq p)$  which lie on a three-dimensional grid  $\mathbf{uvw}$ , which we call the  $\mathbf{uvw}$  space.  $\mathbf{U}$  is a spacing vector  $(u_1, \dots, u_{n-1})$ , where  $u_i$  gives the distance between the grid planes  $\mathbf{u}_i$  and  $\mathbf{u}_{i+1}$ ,  $\mathbf{V}$  is a spacing vector  $(v_1, \dots, v_{m-1})$ , where  $v_j$  gives the distance between the grid planes  $\mathbf{v}_j$  and  $\mathbf{v}_{j+1}$ , and similarly  $\mathbf{W}$  is a spacing vector  $(w_1, \dots, w_{p-1})$ , where  $w_k$  gives the distance between the grid planes  $\mathbf{w}_k$  and  $\mathbf{w}_{k+1}$ .

Note that although each grid point  $\mathbf{p}$  still consists of a color vector  $\mathbf{c}$  and a location vector  $\mathbf{x}$ , in an implementation the location vector is not stored with each grid point, since the location is quickly computed from the point's array indices.

## 2.3.2 1D Visibility Ordering

This section gives the perspective back-to-front (PBTF) visibility ordering for a 1D grid. First, Section 2.3.2.1 discusses the possible relationships between a view point and a 1D grid. These relationships yield two cases: *edge-on* and *corner-on* views of a grid. The edge-on case is the most general; Theorem 1 gives a visibility ordering for this case in Section 2.3.2.2. Corollary 1 then gives the visibility ordering for the corner-on case in Section 2.3.2.3.

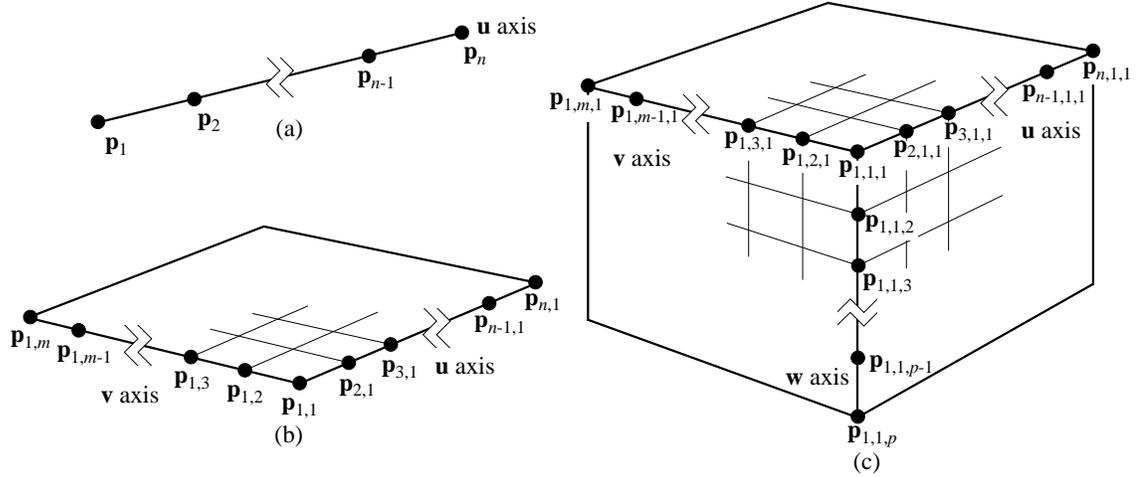


Figure 2.9: 1D, 2D, and 3D grids.

### 2.3.2.1 Views of a 1D Grid

Assume that we have a 1D grid as shown in Figure 2.9a and a viewpoint  $\mathbf{vp}$ , and let the projection of  $\mathbf{vp}$  on the  $\mathbf{u}$  axis be  $\mathbf{vp}_u$ . Then there are two possible *views* or relationships between the grid and  $\mathbf{vp}_u$ ; these are summarized in Table 2.1.

We call the situation where the projection of  $\mathbf{vp}$  is contained within the grid boundary (e.g.  $\mathbf{u}_1 \leq \mathbf{vp}_u \leq \mathbf{u}_n$ ) the *edge-on* view — by this we mean that the view point is looking at the grid surface “edge-on” (see Figure 2.10). A visibility ordering for this view is given in this section in Theorem 1. The only other possible views are when  $\mathbf{vp}_u$  falls before  $\mathbf{u}_1$  or after  $\mathbf{u}_n$ ; we refer to these as *corner-on* views (see Figure 2.13). Visibility orderings for these views are given in Corollary 1.

### 2.3.2.2 1D Edge-On View

Assume that we have a 1D grid as shown in Figure 2.10. Assume without loss of generality that  $\mathbf{p}_q \leq \mathbf{vp}_u < \mathbf{p}_{q+1}$  (i.e. that the projection of  $\mathbf{vp}$  on the  $\mathbf{u}$  axis can lie directly on  $\mathbf{p}_q$ , but must be strictly less than  $\mathbf{p}_{q+1}$ ). Then we can define a visibility ordering for the 1D grid:

1D Grid				
view	number of cases	case	direction of vp from grid	relationship of vp to grid
Edge On:	1 case (Theorem 1)	1*	within	$u_1 < vp_u < u_n$
Corner On:	2 cases (Corollary 1)	1*	L	$vp_u \leq u_1$
		2*	R	$u_n \leq vp_u$

Table 2.1: The relationship between vp and a 1D grid. The cases marked “\*” are covered in Theorem 1 or Corollary 1.

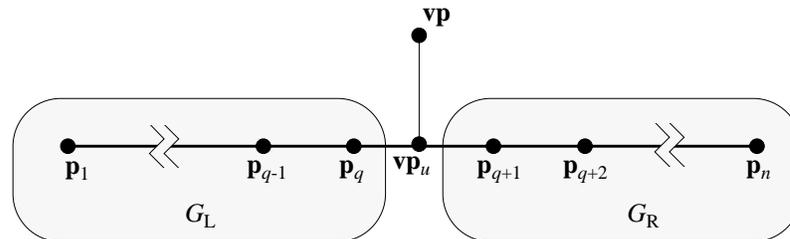


Figure 2.10: A 1D grid showing the location of the view point  $vp$  and point objects divided into the sets  $G_L$  and  $G_R$ .

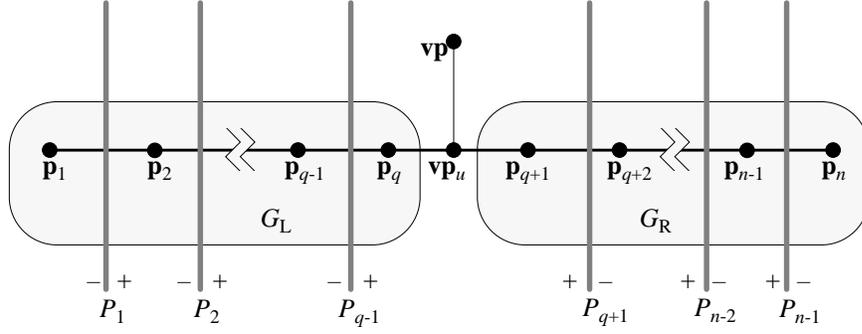


Figure 2.11: A 1D grid with dividing planes added.

**Theorem 1** A visibility ordering for a 1D grid with  $\mathbf{p}_q \leq \mathbf{vp}_u < \mathbf{p}_{q+1}$  is as follows:

First, divide the grid into the sets:

$$\langle G_L \rangle = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_q\} \quad (2.13)$$

$$\langle G_R \rangle = \{\mathbf{p}_{q+1}, \mathbf{p}_{q+2}, \dots, \mathbf{p}_n\}. \quad (2.14)$$

Then a visibility ordering for each set is:

$$\langle G_L \rangle = \mathbf{p}_q \Rightarrow \mathbf{p}_{q-1} \Rightarrow \dots \Rightarrow \mathbf{p}_2 \Rightarrow \mathbf{p}_1 \quad (2.15)$$

$$\langle G_R \rangle = \mathbf{p}_{q+1} \Rightarrow \mathbf{p}_{q+2} \Rightarrow \dots \Rightarrow \mathbf{p}_{n-1} \Rightarrow \mathbf{p}_n. \quad (2.16)$$

**Proof:** Insert dividing planes into the grid as shown in Figure 2.11, where planes  $P_i : 1 \leq i \leq q-1, q+1 \leq i \leq n-1$  are placed anywhere between points  $\mathbf{p}_i$  and  $\mathbf{p}_{i+1}$ . Define the half-spaces  $P_i^+$  to be the side of each plane facing  $\mathbf{vp}$ , and the half-spaces  $P_i^-$  to be the side facing away from  $\mathbf{vp}$ .

Now consider the points in the set  $G_R$ :  $\mathbf{p}_{q+1}, \mathbf{p}_{q+2}, \dots, \mathbf{p}_{i-1}, \mathbf{p}_i, \mathbf{p}_{i+1}, \dots, \mathbf{p}_{n-1}, \mathbf{p}_n$  (Figure 2.12). For any  $\mathbf{p}_i : q+1 \leq i < n$ , by the dividing property of planes  $\mathbf{p}_i \Rightarrow \{\mathbf{p}_{i+1}, \mathbf{p}_{i+2}, \dots, \mathbf{p}_n\}$  since  $\mathbf{p}_i \in P_i^+$  and  $\{\mathbf{p}_{i+1}, \mathbf{p}_{i+2}, \dots, \mathbf{p}_n\} \in P_i^-$ . Hence  $\mathbf{p}_{q+1} \Rightarrow \mathbf{p}_{q+2} \Rightarrow \dots \Rightarrow \mathbf{p}_{n-1} \Rightarrow \mathbf{p}_n$ .

And consider the points  $\mathbf{p}_q, \mathbf{p}_{q-1}, \dots, \mathbf{p}_2, \mathbf{p}_1$ . A symmetric argument proves that  $\mathbf{p}_q \Rightarrow \mathbf{p}_{q-1} \Rightarrow \dots \Rightarrow \mathbf{p}_2 \Rightarrow \mathbf{p}_1$ . **Q.E.D.**

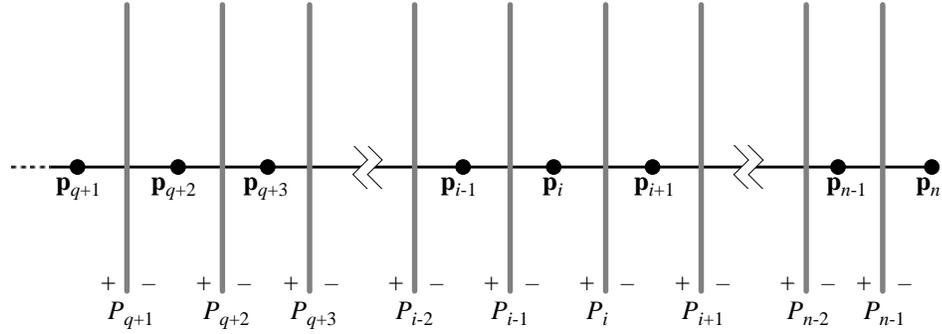


Figure 2.12: The right-hand points in the set  $G_R$ .

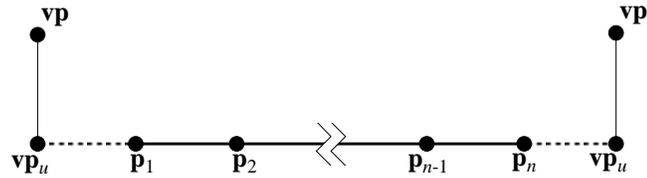


Figure 2.13:  $vp_u$  either falls before  $p_1$  or after  $p_n$ .

Note that Theorem 1 and its proof do not specify a visibility ordering between sets  $G_L$  and  $G_R$  — there is no such visibility ordering because  $G_L \not\parallel G_R$  and  $G_R \not\parallel G_L$ . Hence either  $G_L \Rightarrow G_R$  or  $G_R \Rightarrow G_L$  without effecting visibility.

### 2.3.2.3 1D Corner-On View

The visibility ordering for the 1D corner-on cases leads to the following simple corollary:

**Corollary 1** *A visibility ordering for a 1D grid when  $vp_u \leq p_1$  is:*

$$p_1 \Rightarrow p_2 \Rightarrow \dots \Rightarrow p_{n-1} \Rightarrow p_n. \quad (2.17)$$

*A visibility ordering for a 1D grid when  $vp_u \geq p_n$  is:*

$$p_n \Rightarrow p_{n-1} \Rightarrow \dots \Rightarrow p_2 \Rightarrow p_1. \quad (2.18)$$

**Proof:** The corollary follows immediately from Theorem 1. **Q.E.D.**

### 2.3.3 2D Visibility Ordering

This section gives the perspective back-to-front (PBTF) visibility ordering for a 2D grid. First, Section 2.3.3.1 gives some nomenclature which is helpful when discussing 2D grids. Then Section 2.3.3.2 discusses the possible relationships between a view point and a 2D grid. These relationships yield three cases: *face-on*, *edge-on*, and *corner-on* views of a grid. The face-on case is the most general; Theorem 2 gives a visibility ordering for this case in Section 2.3.3.3. Corollaries 2 and 3 then give visibility orderings for the edge-on and corner-on cases in Sections 2.3.3.4 and 2.3.3.5, respectively.

#### 2.3.3.1 Nomenclature

We use the following nomenclature to refer to the different parts of a 2D grid, as shown in Figure 2.14. The grid is defined by the unit vectors  $\mathbf{u}$  and  $\mathbf{v}$ , where the grid origin is in the lower-left corner. Within the grid we use the four directions shown in Figure 2.14; these are called the *edge directions* because they give the direction to each edge from the point of view of the center of the grid. The edge directions are L, R, U, and D, where

- L and R stand for (L)eft and (R)ight and give a direction relative to the  $\mathbf{u}$  axis, and
- U and D stand for (U)p and (D)own and give a direction relative to the  $\mathbf{v}$  axis.

Combining these edge directions into pairs we obtain the four *corner directions* LU, LD, RU, and RD, which give the direction towards the four corners of the grid.

In general we will deal with a grid of size  $n \times m$ , where there are  $n$  point objects along the  $\mathbf{u}$  axis and  $m$  point objects along the  $\mathbf{v}$  axis. We will usually be interested in a point object in the interior of the grid; we will refer to this as the point  $\mathbf{p}_{q,r}$ .

#### 2.3.3.2 Views of a 2D Grid

Assume that we have a 2D grid as shown in Figure 2.14 and a viewpoint  $\mathbf{vp}$ , and let the projection of  $\mathbf{vp}$  on the  $\mathbf{uv}$  plane be  $(\mathbf{vp}_u, \mathbf{vp}_v)$ . Then there are three possible views or relationships of the grid from  $(\mathbf{vp}_u, \mathbf{vp}_v)$ ; these are summarized in Table 2.2.

<b>2D Grid</b>				
<b>view</b>	<b>number of cases</b>	<b>case</b>	<b>direction of vp from grid</b>	<b>relationship of vp to grid</b>
Face On:	1 case (Theorem 2)	1*		$u_1 < vp_u < u_n$ and $v_1 < vp_v < v_m$
Edge On:	4 cases (Corollary 2)	1*	L	$vp_u \leq u_1$ and $v_1 < vp_v < v_m$
		2	R	$u_n \leq vp_u$ and $v_1 < vp_v < v_m$
		3	U	$u_1 < vp_u < u_n$ and $v_m \leq vp_v$
		4	D	$u_1 < vp_u < u_n$ and $vp_v \leq v_1$
Corner On:	4 cases (Corollary 3)	1	LU	$vp_u \leq u_1$ and $v_m \leq vp_v$
		2*	LD	$vp_u \leq u_1$ and $vp_v \leq v_1$
		3	RU	$u_n \leq vp_u$ and $v_m \leq vp_v$
		4	RD	$u_n \leq vp_u$ and $vp_v \leq v_1$

Table 2.2: The relationship between  $vp$  and a 2D grid. The cases marked “\*” are proved in Theorem 2 and Corollaries 2 and 3.

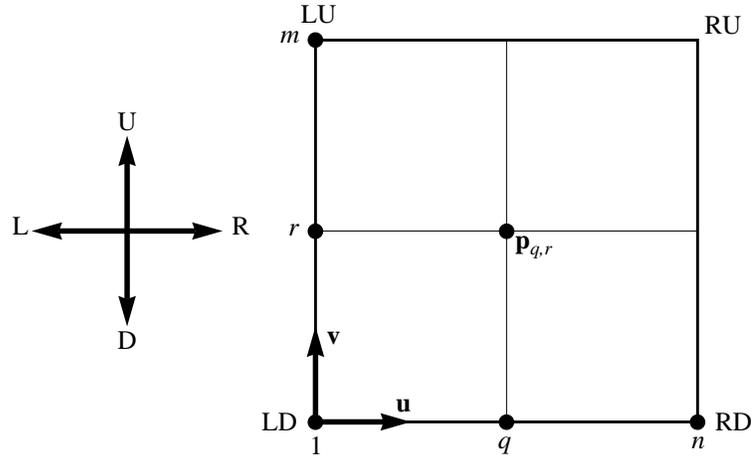


Figure 2.14: Naming conventions for a 2D grid.

Borrowing terminology from Anderson [4], we classify the views as face-on, edge-on, and corner-on, where

- a **face-on** view is when  $(\mathbf{vp}_u, \mathbf{vp}_v)$  is contained within the grid. A visibility ordering for this view is given in Theorem 2.
- An **edge-on** view is when one coordinate (either  $\mathbf{vp}_u$  or  $\mathbf{vp}_v$ ) lies outside the grid and the other coordinate lies within the grid. As listed in Table 2.2 there are four possible edge-on cases, depending on the direction of the viewpoint from the grid. A visibility ordering for one of these cases is given in Corollary 2. The other cases are provable by symmetric corollaries. And
- a **corner-on** view is when both the coordinates  $\mathbf{vp}_u$  and  $\mathbf{vp}_v$  lie outside the grid. As listed in Table 2.2 there are four possible corner-on cases, one for each corner beyond which the viewpoint could lie. A visibility ordering for one of these cases is given in Corollary 3. The other cases are provable by symmetric corollaries.



$$\begin{aligned}
\langle G_{RD} \rangle &= \{ \mathbf{p}_{i,j} : q+1 \leq i \leq n, 1 \leq j \leq r \} \\
\langle G_{RU} \rangle &= \{ \mathbf{p}_{i,j} : q+1 \leq i \leq n, r+1 \leq j \leq m \} \\
\langle G_{LU} \rangle &= \{ \mathbf{p}_{i,j} : 1 \leq i \leq q, r+1 \leq j \leq m \}
\end{aligned}$$

Then a visibility ordering for each set is:

$$\begin{aligned}
\langle G_{LD} \rangle &= \langle \langle \mathbf{p}_{i,j} : q \geq i \geq 1 \rangle : r \geq j \geq 1 \rangle \text{ or dual} \\
\langle G_{RD} \rangle &= \langle \langle \mathbf{p}_{i,j} : q+1 \leq i \leq n \rangle : r \geq j \geq 1 \rangle \text{ or dual} \\
\langle G_{RU} \rangle &= \langle \langle \mathbf{p}_{i,j} : q+1 \leq i \leq n \rangle : r+1 \leq j \leq m \rangle \text{ or dual} \\
\langle G_{LU} \rangle &= \langle \langle \mathbf{p}_{i,j} : q \geq i \geq 1 \rangle : r+1 \leq j \leq m \rangle \text{ or dual}
\end{aligned}$$

**Proof:** We shall show the visibility ordering for set  $G_{LD}$ ; symmetric arguments demonstrate the visibility ordering for sets  $G_{RD}$ ,  $G_{RU}$ , and  $G_{LU}$ .

Consider the set  $G_{LD}$  as shown in Figure 2.16. Let  $G_{LD}$  be broken into subsets  $G_1, G_2, \dots, G_{r-1}, G_r$  where subset  $G_j$  is composed of all the grid points on row  $j$ :

$$G_j = \bigcup_{i=1}^q \mathbf{p}_{i,j}. \quad (2.19)$$

Place horizontal dividing planes  $P_1, P_2, \dots, P_{r-2}, P_{r-1}$  between each row of grid points so that plane  $P_j$  divides set  $G_j$  from set  $G_{j+1}$ . Let the half spaces  $P_j^+$  be the side of each plane facing  $\mathbf{vp}$ , and the half spaces  $P_j^-$  be the side facing away from  $\mathbf{vp}$ .

For any subset  $G_j : 1 \leq j \leq r$ , we have  $G_j \subset P_{j-1}^+$  and  $\{G_{j-1}, G_{j-2}, \dots, G_1\} \subset P_{j-1}^-$ . Hence  $G_j \Rightarrow \{G_{j-1}, G_{j-2}, \dots, G_1\}$ , and thus we have

$$G_r \Rightarrow G_{r-1} \Rightarrow \dots \Rightarrow G_2 \Rightarrow G_1. \quad (2.20)$$

Now consider each subset  $G_j$ . Corollary 1 gives the visibility ordering

$$\mathbf{p}_{q,j} \Rightarrow \mathbf{p}_{q-1,j} \Rightarrow \dots \Rightarrow \mathbf{p}_{2,j} \Rightarrow \mathbf{p}_{1,j}. \quad (2.21)$$

This combined with Equation 2.20 gives us the overall visibility ordering for set  $G_{LD}$ :

$$\langle \langle \mathbf{p}_{i,j} : q \geq i \geq 1 \rangle : r \geq j \geq 1 \rangle. \quad (2.22)$$

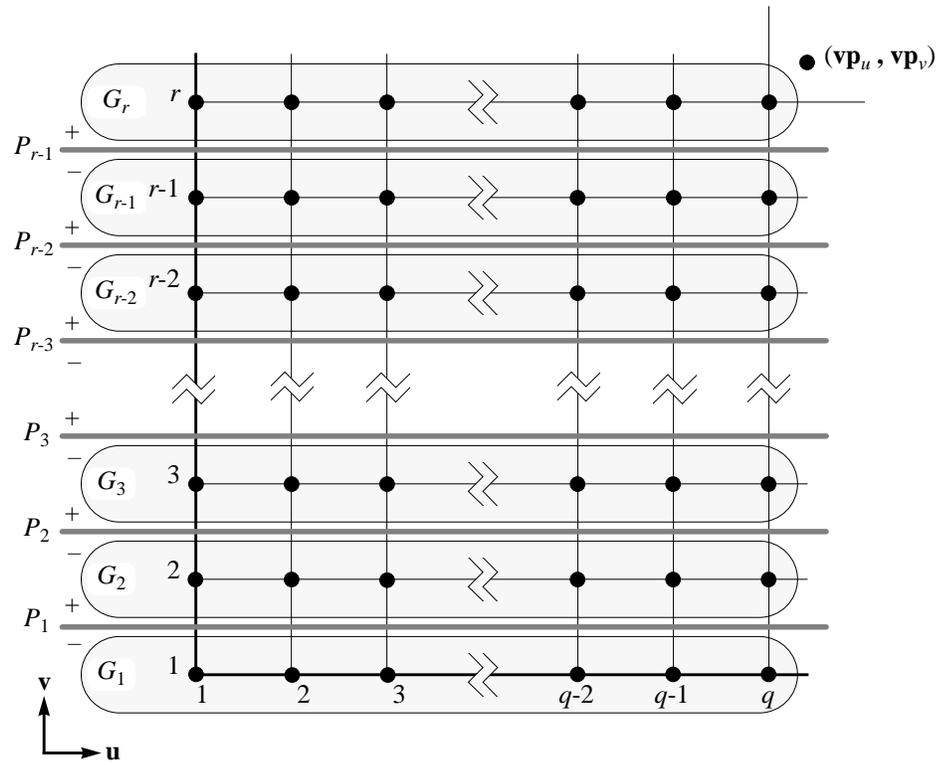


Figure 2.16: The visibility ordering for the set  $G_{LD}$ .

This argument has proved the visibility ordering for set  $G_{LD}$ . Symmetric arguments prove the ordering for sets  $G_{RD}$ ,  $G_{RU}$ , and  $G_{LU}$ . **Q.E.D.**

Note that by placing the cutting planes vertically instead of horizontally we can obtain the *dual* visibility ordering

$$\langle\langle \mathbf{p}_{i,j} : r \geq j \geq 1 \rangle : q \geq i \geq 1 \rangle. \quad (2.23)$$

Also note that similar to Theorem 1, Theorem 2 and its proof do not specify a visibility ordering between the sets  $G_{LD}$ ,  $G_{RD}$ ,  $G_{RU}$ , and  $G_{LU}$  — there is no such visibility ordering because the sets do not obscure each other with respect to  $\mathbf{vp}$ .

### 2.3.3.4 2D Edge-On View

Assume that we have a 2D grid, and assume that  $(\mathbf{vp}_u, \mathbf{vp}_v)$  is to the left of the grid, as shown in Figure 2.17. Furthermore assume that  $\mathbf{vp}_v$  is closer to  $\mathbf{p}_{1,r}$  than any other grid point. Then we can define a visibility ordering for the 2D grid:

**Corollary 2** *A visibility ordering for a 2D grid when  $\mathbf{vp}_u \leq \mathbf{u}_1$  and for each  $j$  such that  $1 \leq j < r, r < j \leq m$  we have  $|\mathbf{vp}_v - \mathbf{p}_{1,r}| \leq |\mathbf{vp}_v - \mathbf{p}_{1,j}|$ , is as follows.*

*First, divide the grid into the sets:*

$$\langle G_U \rangle = \{ \mathbf{p}_{i,j} : 1 \leq i \leq n, r \leq j \leq m \} \quad (2.24)$$

$$\langle G_D \rangle = \{ \mathbf{p}_{i,j} : 1 \leq i \leq n, 1 \leq j \leq r - 1 \} \quad (2.25)$$

*Then a visibility ordering for each set is:*

$$\langle G_U \rangle = \langle\langle \mathbf{p}_{i,j} : 1 \leq i \leq n \rangle : r \leq j \leq m \rangle \text{ or dual} \quad (2.26)$$

$$\langle G_D \rangle = \langle\langle \mathbf{p}_{i,j} : 1 \leq i \leq n \rangle : r - 1 \geq j \geq 1 \rangle \text{ or dual} \quad (2.27)$$

**Proof:** Corollary 2 follows immediately from the visibility ordering of the sets  $G_{RU}$  and  $G_{RD}$  given in Theorem 2. **Q.E.D.**

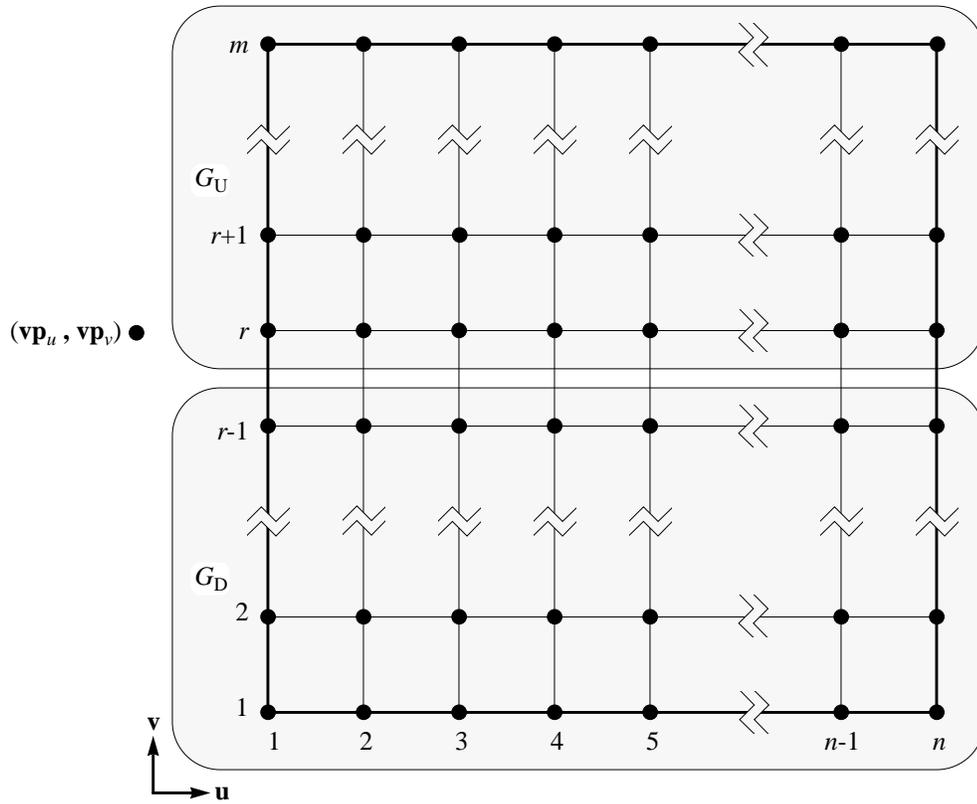


Figure 2.17: The viewpoint is beyond the “L” edge of the grid. This is case 1 of the edge-on view from Table 2.2.

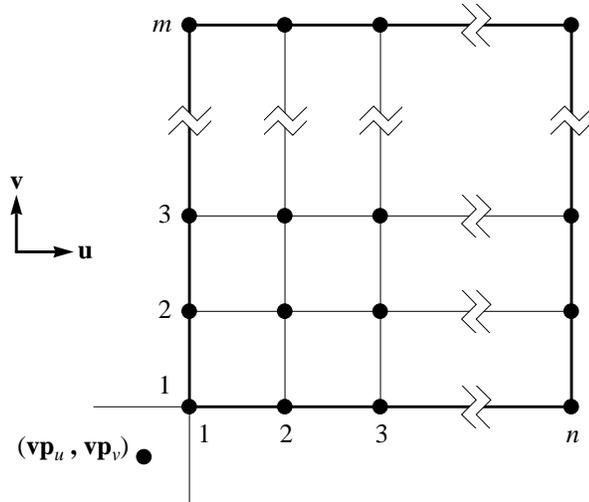


Figure 2.18: The viewpoint is beyond the “LD” corner of the grid. This is case 2 of the corner-on view from Table 2.2.

### 2.3.3.5 2D Corner-On View

Assume that we have a 2D grid, and assume that  $(\mathbf{vp}_u, \mathbf{vp}_v)$  is beyond the LD corner, as shown in Figure 2.17. Then we can define a visibility ordering for the 2D grid:

**Corollary 3** *A visibility ordering for a 2D grid when  $\mathbf{vp}_u \leq \mathbf{u}_1$  and  $\mathbf{vp}_v \leq \mathbf{v}_1$  is:*

$$\langle \langle \mathbf{p}_{i,j} : 1 \leq i \leq n \rangle : 1 \leq j \leq m \rangle \text{ or dual} \quad (2.28)$$

**Proof:** Corollary 3 follows immediately from the visibility ordering of the set  $G_{RU}$  given in Theorem 2. **Q.E.D.**

### 2.3.4 3D Visibility Ordering

This section gives the perspective back-to-front (PBTF) visibility ordering for a 3D grid. First, Section 2.3.4.1 gives some nomenclature which is helpful when discussing 3D grids. Then Section 2.3.4.2 discusses the possible relationships between a view point and a 3D grid. These relationships yield four cases: *volume-on*, *face-on*, *edge-on*, and *corner-on*

views of a grid. The volume-on case is the most general; Theorem 3 gives a visibility ordering for this case in Section 2.3.4.3. Corollaries 4, 5, and 6 then give visibility orderings for the face-on, edge-on, and corner-on cases in Sections 2.3.4.4, 2.3.4.5, and 2.3.4.6, respectively.

### 2.3.4.1 Nomenclature

We use the following nomenclature to refer to the different parts of a 3D grid; in general this nomenclature follows that of Samet [88]. A 3D grid is shown in Figure 2.19. The grid is defined by the unit vectors  $\mathbf{u}$ ,  $\mathbf{v}$ , and  $\mathbf{w}$  where the grid origin is the lower-left corner; this is similar to taking the 2D grid from Figure 2.14 and extruding it out in the  $\mathbf{w}$  direction. Within the grid we indicate a direction by using the six “primitive” directions shown in Figure 2.19; these are called the *face directions* because they give the direction of each face from the point of view of the center of the cube. The face directions are L, R, U, D, F, and B, where

- L and R stand for (L)eft and (R)ight and give a direction relative to the  $\mathbf{u}$  axis,
- U and D stand for (U)p and (D)own and give a direction relative to the  $\mathbf{v}$  axis, and
- F and B stand for (F)ront and (B)ack and give a direction relative to the  $\mathbf{w}$  axis.

Combining these face directions into pairs we obtain the 12 *edge directions* LU, LD, LF, LB, RU, RD, RF, RB, UF, UB, DF, and DB, which give the direction towards the 12 edges of the grid. And combining the face directions into triples we obtain the eight *corner directions* LUF, LUB, LDF, LDB, RUF, RUB, RDF, and RDB, which give the direction towards the eight corners of the grid.

In general we will deal with a grid of size  $n \times m \times p$ , where there are  $n$  point objects along the  $\mathbf{u}$  axis,  $m$  point objects along the  $\mathbf{v}$  axis, and  $p$  point objects along the  $\mathbf{w}$  axis. We will usually be interested in a point object in the interior of the grid; we will refer to this as the point  $\mathbf{p}_{q,r,s}$ .

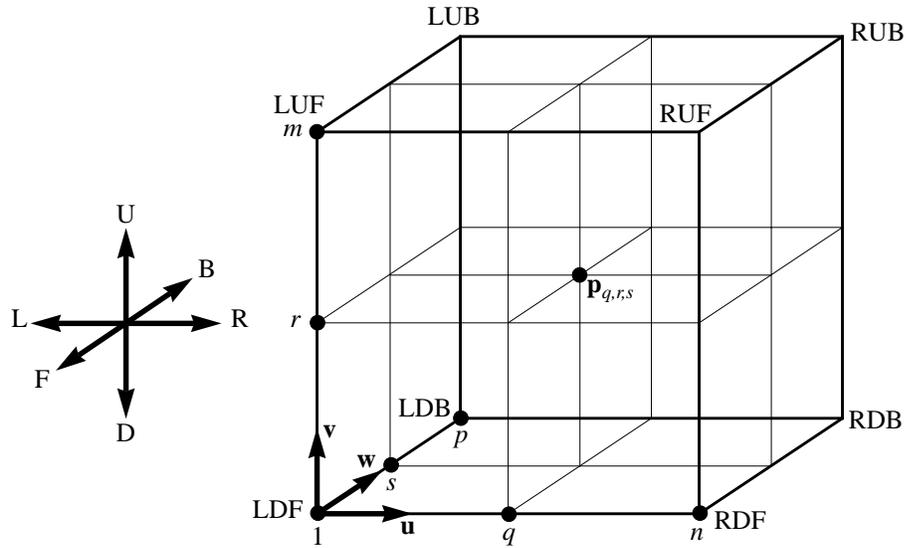


Figure 2.19: Naming conventions for a 3D grid.

### 2.3.4.2 Views of a 3D Grid

Assume that we have a 3D grid as shown in Figure 2.19 and a viewpoint  $\mathbf{vp}$ , and let the location of  $\mathbf{vp}$  expressed the  $uvw$  coordinate system be  $(\mathbf{vp}_u, \mathbf{vp}_v, \mathbf{vp}_w)$ . Then there are four possible views or relationships of the grid from  $(\mathbf{vp}_u, \mathbf{vp}_v, \mathbf{vp}_w)$ ; these are summarized in Table 2.3.

Extending the terminology from our 2D grid, we classify the views as volume-on, face-on, edge-on, and corner-on, where

- a **volume-on** view is when  $(\mathbf{vp}_u, \mathbf{vp}_v, \mathbf{vp}_w)$  is contained within the grid. A visibility ordering for this view is given in Theorem 3.
- A **face-on** view is when one coordinate of  $\mathbf{vp}$  lies outside the grid and the other two still lie inside the grid; in this situation the viewpoint is looking straight at a face of the grid. As listed in Table 2.3 there are six possible face-on cases, depending on the direction of the viewpoint from the grid. A visibility ordering for one of these cases is given in Corollary 4; the other cases are provable by symmetric corollaries.

<b>3D Grid</b>				
<b>view</b>	<b>number of cases</b>	<b>case</b>	<b>direction of vp from grid</b>	<b>relationship of vp to grid</b>
Volume On:	1 case (Theorem 3)	1*		$\mathbf{u}_1 < \mathbf{vp}_u < \mathbf{u}_n$ and $\mathbf{v}_1 < \mathbf{vp}_v < \mathbf{v}_m$ and $\mathbf{w}_1 < \mathbf{vp}_w < \mathbf{w}_p$
Face On:	6 cases (Corollary 4)	1	L	$\mathbf{vp}_u \leq \mathbf{u}_1$ and $\mathbf{v}_1 < \mathbf{vp}_v < \mathbf{v}_m$ and $\mathbf{w}_1 < \mathbf{vp}_w < \mathbf{w}_p$
		2	R	$\mathbf{u}_n \leq \mathbf{vp}_u$ and $\mathbf{v}_1 < \mathbf{vp}_v < \mathbf{v}_m$ and $\mathbf{w}_1 < \mathbf{vp}_w < \mathbf{w}_p$
		3	U	$\mathbf{u}_1 < \mathbf{vp}_u < \mathbf{u}_n$ and $\mathbf{v}_m \leq \mathbf{vp}_v$ and $\mathbf{w}_1 < \mathbf{vp}_w < \mathbf{w}_p$
		4	D	$\mathbf{u}_1 < \mathbf{vp}_u < \mathbf{u}_n$ and $\mathbf{vp}_v \leq \mathbf{v}_1$ and $\mathbf{w}_1 < \mathbf{vp}_w < \mathbf{w}_p$
		5*	F	$\mathbf{u}_1 < \mathbf{vp}_u < \mathbf{u}_n$ and $\mathbf{v}_1 < \mathbf{vp}_v < \mathbf{v}_m$ and $\mathbf{vp}_w \leq \mathbf{w}_1$
		6	B	$\mathbf{u}_1 < \mathbf{vp}_u < \mathbf{u}_n$ and $\mathbf{v}_1 < \mathbf{vp}_v < \mathbf{v}_m$ and $\mathbf{w}_p \leq \mathbf{vp}_w$
Edge On:	12 cases (Corollary 5)	1	LU	$\mathbf{vp}_u \leq \mathbf{u}_1$ and $\mathbf{v}_m \leq \mathbf{vp}_v$ and $\mathbf{w}_1 < \mathbf{vp}_w < \mathbf{w}_p$

Table 2.3 continued on next page

Table 2.3: The relationship between vp and a 3D grid. The cases marked “\*” are covered in Theorem 3 and Corollaries 4, 5 and 6.

Table 2.3 continued from previous page

view	number of cases	case	direction of vp from grid	relationship of vp to grid
		2	LD	$vp_u \leq u_1$ and $vp_v \leq v_1$ and $w_1 < vp_w < w_p$
		3	LF	$vp_u \leq u_1$ and $v_1 < vp_v < v_m$ and $vp_w \leq w_1$
		4	LB	$vp_u \leq u_1$ and $v_1 < vp_v < v_m$ and $w_p \leq vp_w$
		5	RU	$u_n \leq vp_u$ and $v_m \leq vp_v$ and $w_1 < vp_w < w_p$
		6	RD	$u_n \leq vp_u$ and $vp_v \leq v_1$ and $w_1 < vp_w < w_p$
		7*	RF	$u_n \leq vp_u$ and $v_1 < vp_v < v_m$ and $vp_w \leq w_1$
		8	RB	$u_n \leq vp_u$ and $v_1 < vp_v < v_m$ and $w_p \leq vp_w$
		9	UF	$u_1 < vp_u < u_n$ and $v_m \leq vp_v$ and $vp_w \leq w_1$
		10	UB	$u_1 < vp_u < u_n$ and $v_m \leq vp_v$ and $w_p \leq vp_w$

Table 2.3 continued on next page

Table 2.3 continued from previous page

view	number of cases	case	direction of vp from grid	relationship of vp to grid
		11	DF	$u_1 < vp_u < u_n$ and $vp_v \leq v_1$ and $vp_w \leq w_1$
		12	DB	$u_1 < vp_u < u_n$ and $vp_v \leq v_1$ and $w_p \leq vp_w$
Corner On:	8 cases (Corollary 6)	1	LUF	$vp_u \leq u_1$ and $v_m \leq vp_v$ and $vp_w \leq w_1$
		2	LUB	$vp_u \leq u_1$ and $v_m \leq vp_v$ and $w_p \leq vp_w$
		3	LDF	$vp_u \leq u_1$ and $vp_v \leq v_1$ and $vp_w \leq w_1$
		4	LDB	$vp_u \leq u_1$ and $vp_v \leq v_1$ and $w_p \leq vp_w$
		5	RUF	$u_n \leq vp_u$ and $v_m \leq vp_v$ and $vp_w \leq w_1$
		6	RUB	$u_n \leq vp_u$ and $v_m \leq vp_v$ and $w_p \leq vp_w$
		7*	RDF	$u_n \leq vp_u$ and $vp_v \leq v_1$ and $vp_w \leq w_1$
		8	RDB	$u_n \leq vp_u$ and $vp_v \leq v_1$ and $w_p \leq vp_w$

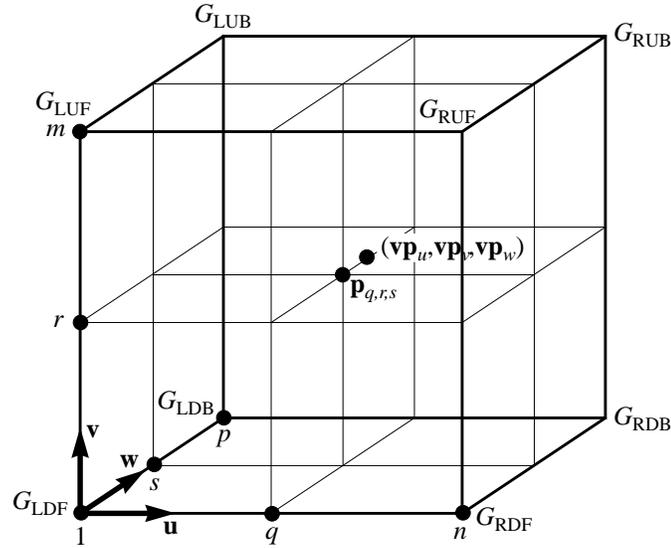


Figure 2.20: Grid layout for proof of 3D visibility ordering.

- An *edge-on* view is when two coordinates of  $\mathbf{vp}$  lie outside the grid and the third coordinate lies inside the grid; in this case  $\mathbf{vp}$  is located beyond a grid edge. As listed in Table 2.3 there are 12 possible edge-on cases. A visibility ordering for one of these cases is given in Corollary 5; the other cases are provable by symmetric corollaries. And
- a *corner-on* view is when all three coordinates of  $\mathbf{vp}$  lie outside the grid; for this view  $\mathbf{vp}$  lies beyond a grid corner. As listed in Table 2.3 there are eight possible corner-on cases. A visibility ordering for one of these cases is given in Corollary 6; the other cases are provable by symmetric corollaries.

### 2.3.4.3 3D Volume-On View

Assume that we have a 3D grid as shown in Figure 2.20. Assume without loss of generality that  $\mathbf{vp} = (\mathbf{vp}_u, \mathbf{vp}_v, \mathbf{vp}_w)$  is closer to  $\mathbf{p}_{q,r,s}$  than any other point. Then we can define a visibility ordering for the 3D grid:

**Theorem 3** A visibility ordering for a 3D grid when, for each  $1 \leq i < q, q < i \leq n; 1 \leq j < r, r < j \leq m; 1 \leq k < s, s < k \leq p$  we have  $|(\mathbf{vp}_u, \mathbf{vp}_v, \mathbf{vp}_w) - \mathbf{p}_{q,r,s}| \leq |(\mathbf{vp}_u, \mathbf{vp}_v, \mathbf{vp}_w) - \mathbf{p}_{i,j,k}|$ , is as follows.

First, divide the grid into the sets:

$$\langle G_{\text{LDF}} \rangle = \{\mathbf{p}_{i,j,k} : 1 \leq i \leq q, 1 \leq j \leq r, 1 \leq k \leq s\} \quad (2.29)$$

$$\langle G_{\text{RDF}} \rangle = \{\mathbf{p}_{i,j,k} : q+1 \leq i \leq n, 1 \leq j \leq r, 1 \leq k \leq s\} \quad (2.30)$$

$$\langle G_{\text{LUF}} \rangle = \{\mathbf{p}_{i,j,k} : 1 \leq i \leq q, r+1 \leq j \leq m, 1 \leq k \leq s\} \quad (2.31)$$

$$\langle G_{\text{RUF}} \rangle = \{\mathbf{p}_{i,j,k} : q+1 \leq i \leq n, r+1 \leq j \leq m, 1 \leq k \leq s\} \quad (2.32)$$

$$\langle G_{\text{LDB}} \rangle = \{\mathbf{p}_{i,j,k} : 1 \leq i \leq q, 1 \leq j \leq r, s+1 \leq k \leq p\} \quad (2.33)$$

$$\langle G_{\text{RDB}} \rangle = \{\mathbf{p}_{i,j,k} : q+1 \leq i \leq n, 1 \leq j \leq r, s+1 \leq k \leq p\} \quad (2.34)$$

$$\langle G_{\text{LUB}} \rangle = \{\mathbf{p}_{i,j,k} : 1 \leq i \leq q, r+1 \leq j \leq m, s+1 \leq k \leq p\} \quad (2.35)$$

$$\langle G_{\text{RUB}} \rangle = \{\mathbf{p}_{i,j,k} : q+1 \leq i \leq n, r+1 \leq j \leq m, s+1 \leq k \leq p\} \quad (2.36)$$

Then a visibility ordering for each set is:

$$\begin{aligned} \langle G_{\text{LDF}} \rangle &= \langle \langle \langle \mathbf{p}_{i,j,k} : q \geq i \geq 1 \rangle : r \geq j \geq 1 \rangle : s \geq k \geq 1 \rangle \\ &\text{or duals} \end{aligned} \quad (2.37)$$

$$\begin{aligned} \langle G_{\text{RDF}} \rangle &= \langle \langle \langle \mathbf{p}_{i,j,k} : q+1 \leq i \leq n \rangle : r \geq j \geq 1 \rangle : s \geq k \geq 1 \rangle \\ &\text{or duals} \end{aligned} \quad (2.38)$$

$$\begin{aligned} \langle G_{\text{LUF}} \rangle &= \langle \langle \langle \mathbf{p}_{i,j,k} : q \geq i \geq 1 \rangle : r+1 \leq j \leq m \rangle : s \geq k \geq 1 \rangle \\ &\text{or duals} \end{aligned} \quad (2.39)$$

$$\begin{aligned} \langle G_{\text{RUF}} \rangle &= \langle \langle \langle \mathbf{p}_{i,j,k} : q+1 \leq i \leq n \rangle : r+1 \leq j \leq m \rangle : s \geq k \geq 1 \rangle \\ &\text{or duals} \end{aligned} \quad (2.40)$$

$$\begin{aligned} \langle G_{\text{LDB}} \rangle &= \langle \langle \langle \mathbf{p}_{i,j,k} : q \geq i \geq 1 \rangle : r \geq j \geq 1 \rangle : s+1 \leq k \leq p \rangle \\ &\text{or duals} \end{aligned} \quad (2.41)$$

$$\begin{aligned} \langle G_{\text{RDB}} \rangle &= \langle \langle \langle \mathbf{p}_{i,j,k} : q+1 \leq i \leq n \rangle : r \geq j \geq 1 \rangle : s+1 \leq k \leq p \rangle \\ &\text{or duals} \end{aligned} \quad (2.42)$$

$$\langle G_{\text{LUB}} \rangle = \langle \langle \langle \mathbf{p}_{i,j,k} : q \geq i \geq 1 \rangle : r+1 \leq j \leq m \rangle : s+1 \leq k \leq p \rangle$$

or duals (2.43)

$$\langle G_{\text{RUB}} \rangle = \langle \langle \langle \mathbf{p}_{i,j,k} : q+1 \leq i \leq n \rangle : r+1 \leq j \leq m \rangle : s+1 \leq k \leq p \rangle$$

or duals (2.44)

**Proof:** We shall show the visibility ordering for set  $G_{\text{LDF}}$ ; symmetric arguments demonstrate the visibility ordering for the sets  $G_{\text{RDF}}$ ,  $G_{\text{LUF}}$ ,  $G_{\text{RUF}}$ ,  $G_{\text{LDB}}$ ,  $G_{\text{RDB}}$ ,  $G_{\text{LUB}}$ , and  $G_{\text{RUB}}$ .

Consider the set  $G_{\text{LDF}}$  as shown in Figure 2.21. Let  $G_{\text{LDF}}$  be broken into subsets  $G_1, G_2, \dots, G_{r-1}, G_r$ , where subset  $G_j$  is composed of all the grid points on slice  $j$ :

$$G_j = \bigcup_{i=1}^q \bigcup_{k=1}^s \mathbf{p}_{i,j,k}. \quad (2.45)$$

Place horizontal dividing planes  $P_1, P_2, \dots, P_{r-2}, P_{r-1}$  between each slice of grid points so that plane  $P_j$  divides set  $G_j$  from set  $G_{j+1}$ . Let the half spaces  $P_j^+$  be the side of each plane facing  $\mathbf{vp}$ , and the half spaces  $P_j^-$  be the side facing away from  $\mathbf{vp}$ .

For any subset  $G_j : 1 \leq j \leq r$ , we have  $G_j \subset P_{j-1}^+$  and  $\{G_{j-1}, G_{j-2}, \dots, G_1\} \subset P_{j-1}^-$ . Hence  $G_j \Rightarrow \{G_{j-1}, G_{j-2}, \dots, G_1\}$ , and thus we have

$$G_r \Rightarrow G_{r-1} \Rightarrow \dots \Rightarrow G_2 \Rightarrow G_1. \quad (2.46)$$

Now consider each subset  $G_j$ . Corollary 3 gives the visibility ordering

$$\langle \langle \langle \mathbf{p}_{i,j,k} : q \geq i \geq 1 \rangle : s \geq k \geq 1 \rangle. \quad (2.47)$$

This combined with Equation 2.46 gives us the overall visibility ordering for set  $G_{\text{LDF}}$ :

$$\langle \langle \langle \langle \mathbf{p}_{i,j,k} : q \geq i \geq 1 \rangle : s \geq k \geq 1 \rangle : r \geq j \geq 1 \rangle. \quad (2.48)$$

This argument has proved the visibility ordering for set  $G_{\text{LDF}}$ . Symmetric arguments prove the ordering for sets  $G_{\text{RDF}}$ ,  $G_{\text{LUF}}$ ,  $G_{\text{RUF}}$ ,  $G_{\text{LDB}}$ ,  $G_{\text{RDB}}$ ,  $G_{\text{LUB}}$ , and  $G_{\text{RUB}}$ . **Q.E.D.**

Note that by placing the cutting planes in a different orientation, as well as by changing the orientation of the slice-by-slice cutting planes from Corollary 3, we could end up with

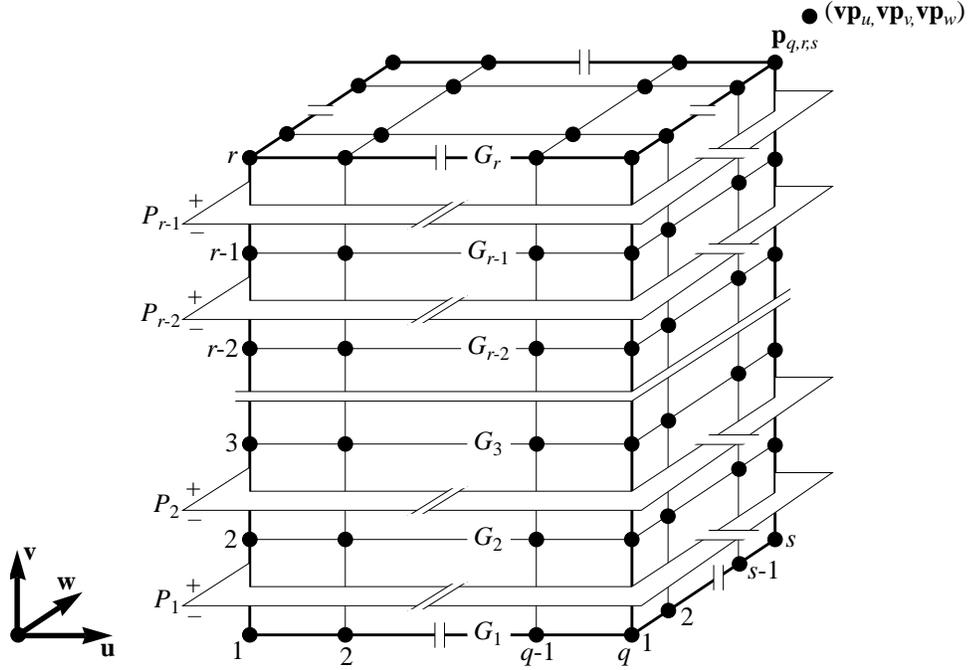


Figure 2.21: The visibility ordering for the set  $G_{LDF}$ .

any of the *dual* visibility orderings for  $G_{LDF}$ :

$$\begin{aligned}
& \langle \langle \langle \mathbf{p}_{i,j,k} : q \geq i \geq 1 \rangle : s \geq k \geq 1 \rangle : r \geq j \geq 1 \rangle \\
\text{or } & \langle \langle \langle \mathbf{p}_{i,j,k} : q \geq i \geq 1 \rangle : r \geq j \geq 1 \rangle : s \geq k \geq 1 \rangle \\
\text{or } & \langle \langle \langle \mathbf{p}_{i,j,k} : r \geq j \geq 1 \rangle : q \geq i \geq 1 \rangle : s \geq k \geq 1 \rangle \\
\text{or } & \langle \langle \langle \mathbf{p}_{i,j,k} : r \geq j \geq 1 \rangle : s \geq k \geq 1 \rangle : q \geq i \geq 1 \rangle \\
\text{or } & \langle \langle \langle \mathbf{p}_{i,j,k} : s \geq k \geq 1 \rangle : q \geq i \geq 1 \rangle : r \geq j \geq 1 \rangle \\
\text{or } & \langle \langle \langle \mathbf{p}_{i,j,k} : s \geq k \geq 1 \rangle : r \geq j \geq 1 \rangle : q \geq i \geq 1 \rangle
\end{aligned} \tag{2.49}$$

Furthermore, as Corollary 3 itself has two visibility orderings, the overall ordering could even change within each slice.

Also note that like Theorem 1 and Theorem 2, Theorem 3 does not specify a visibility ordering between the octant sets  $G_{LDF}$ ,  $G_{RDF}$ ,  $G_{LUF}$ ,  $G_{RUF}$ ,  $G_{LDB}$ ,  $G_{RDB}$ ,  $G_{LUB}$ , and  $G_{RUB}$ — there is no such visibility ordering because the sets do not obscure each other with respect to  $\mathbf{vp}$ .



$$\langle G_{RU} \rangle = \{\mathbf{p}_{i,j,k} : q + 1 \leq i \leq n, r + 1 \leq j \leq m, 1 \leq k \leq p\} \quad (2.52)$$

$$\langle G_{RD} \rangle = \{\mathbf{p}_{i,j,k} : q + 1 \leq i \leq n, 1 \leq j \leq r, 1 \leq k \leq p\} \quad (2.53)$$

Then a visibility ordering for each set is:

$$\begin{aligned} \langle G_{LU} \rangle &= \langle \langle \langle \mathbf{p}_{i,j,k} : q \geq i \geq 1 \rangle : r + 1 \leq j \leq m \rangle : 1 \leq k \leq p \rangle \\ &\text{or duals} \end{aligned} \quad (2.54)$$

$$\begin{aligned} \langle G_{LD} \rangle &= \langle \langle \langle \mathbf{p}_{i,j,k} : q \geq i \geq 1 \rangle : r \geq j \geq 1 \rangle : 1 \leq k \leq p \rangle \\ &\text{or duals} \end{aligned} \quad (2.55)$$

$$\begin{aligned} \langle G_{RU} \rangle &= \langle \langle \langle \mathbf{p}_{i,j,k} : q + 1 \leq i \leq n \rangle : r + 1 \leq j \leq m \rangle : 1 \leq k \leq p \rangle \\ &\text{or duals} \end{aligned} \quad (2.56)$$

$$\begin{aligned} \langle G_{RD} \rangle &= \langle \langle \langle \mathbf{p}_{i,j,k} : q + 1 \leq i \leq n \rangle : r \geq j \geq 1 \rangle : 1 \leq k \leq p \rangle \\ &\text{or duals} \end{aligned} \quad (2.57)$$

**Proof:** Corollary 4 follows immediately from the visibility ordering of the sets  $G_{LUB}$ ,  $G_{LDB}$ ,  $G_{RUB}$ , and  $G_{RDB}$  given in Theorem 3. **Q.E.D.**

### 2.3.4.5 3D Edge-On View

In an edge-on view, two coordinates of  $\mathbf{vp}$  lie outside the 3D grid and one coordinate lies inside the grid, so  $\mathbf{vp}$  projects onto an edge of the grid. For Corollary 5, assume that we have a 3D grid, and assume that  $\mathbf{vp}$  is to the right and in front of the grid (so that it projects onto the RF grid edge) as shown in Figure 2.23. Furthermore assume that the projection of  $\mathbf{vp}$  onto the RF edge  $(n, \mathbf{vp}_v, 1)$  is closer to  $\mathbf{p}_{n,r,1}$  than any other grid point. Then we can define a visibility ordering for the 3D grid:

**Corollary 5** *A visibility ordering for a 3D grid when  $\mathbf{u}_n \leq \mathbf{vp}_w$ ,  $\mathbf{vp}_w \leq \mathbf{w}_1$ , and for each  $1 \leq j < r, r < j \leq m$  we have  $|(n, \mathbf{vp}_v, 1) - \mathbf{p}_{n,r,1}| \leq |(n, \mathbf{vp}_v, 1) - \mathbf{p}_{n,j,1}|$  is as follows.*

*First, divide the grid into the sets:*

$$\langle G_U \rangle = \{\mathbf{p}_{i,j,k} : 1 \leq i \leq n, r + 1 \leq j \leq m, 1 \leq k \leq p\} \quad (2.58)$$

$$\langle G_D \rangle = \{\mathbf{p}_{i,j,k} : 1 \leq i \leq n, 1 \leq j \leq r, 1 \leq k \leq p\} \quad (2.59)$$

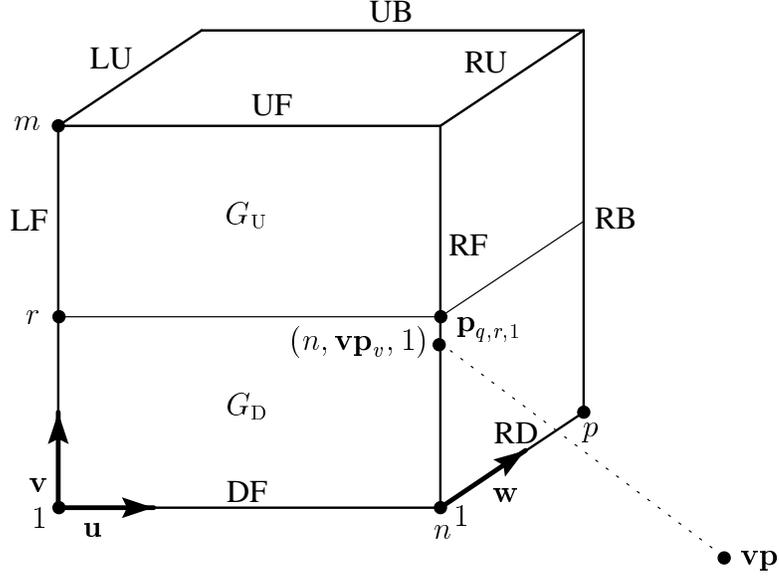


Figure 2.23: The viewpoint is beyond the “RF” edge of the 3D grid. This is case 7 of the edge-on view from Table 2.3. The visible grid edges are labeled.

Then a visibility ordering for each set is:

$$\langle G_U \rangle = \langle \langle \langle \mathbf{p}_{i,j,k} : n \geq i \geq 1 \rangle : r+1 \leq j \leq m \rangle : 1 \leq k \leq p \rangle$$

or duals

(2.60)

$$\langle G_D \rangle = \langle \langle \langle \mathbf{p}_{i,j,k} : n \geq i \geq 1 \rangle : r \geq j \geq 1 \rangle : 1 \leq k \leq p \rangle$$

or duals

(2.61)

**Proof:** Corollary 5 follows immediately from the visibility ordering of the sets  $G_{LUB}$  and  $G_{LDB}$  given in Theorem 3. **Q.E.D.**

### 2.3.4.6 3D Corner-On View

In a corner-on view, all three coordinates of  $\mathbf{vp}$  lie outside the 3D grid, and so  $\mathbf{vp}$  is beyond a grid corner. For Corollary 6, assume that we have a 3D grid, and assume that  $\mathbf{vp}$  is beyond the RDF corner, as shown in Figure 2.24. Then we can define a visibility ordering for the 3D grid:

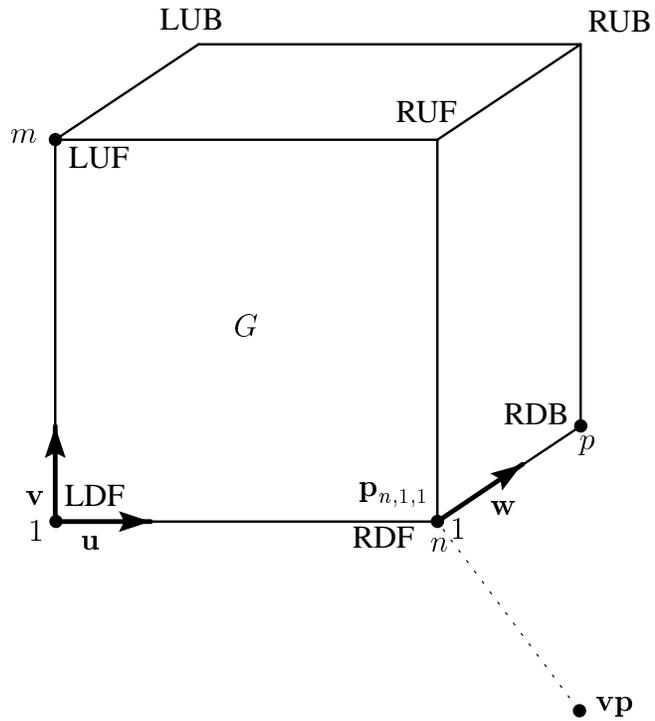


Figure 2.24: The viewpoint is beyond the “RDF” corner of the 3D grid. This is case 7 of the corner-on view from Table 2.3. The visible grid corners are labeled.

**Corollary 6** A visibility ordering for a 3D grid when  $\mathbf{u}_n \leq \mathbf{vp}_w$ ,  $\mathbf{vp}_v \leq \mathbf{v}_1$ , and  $\mathbf{vp}_w \leq \mathbf{w}_1$  is

$$\langle\langle \mathbf{p}_{i,j,k} : n \geq i \geq 1 \rangle : 1 \leq j \leq m \rangle : 1 \leq k \leq p \rangle \text{ or duals.} \quad (2.62)$$

**Proof:** Corollary 6 follows immediately from the visibility ordering of the set  $G_{\text{LUB}}$  given in Theorem 3. **Q.E.D.**

### 2.3.5 Discussion

This section concludes the proof of the PBTF visibility ordering.

The proof is given in an undefined space, which we can assume is equivalent to world space. However, many rendering algorithms operate in either eye space or perspective space. As discussed in Section 3.3.1, eye space is a *rigid body transformation* of world space, and thus preserves parallelism of lines, length of lines, and angles between lines. The splatting algorithm operates in perspective space. As discussed in [74, page 361], the perspective transformation maps straight lines into straight lines and planes into planes. Because planes and lines are preserved under both transformations, the cutting planes on which the proof is based have the same function in both eye space and perspective space as they do in world space. Thus, the proof holds in world space, eye space, and perspective space.

As given, the proof yields a front-to-back ordering of the grid. As discussed in Section 2.3.1, splatting and other list-priority or painter’s algorithms can operate in either a front-to-back or back-to-front order, depending on how the objects are composited into the image plane. The given front-to-back ordering can be trivially reversed to yield an equivalent back-to-front ordering.

## 2.4 Results

Figures 2.25–2.27 show a volumetric dataset rendered with the splatting algorithm discussed in Chapter 1. The dataset is a  $60 \times 60 \times 60$  hollow cube which contains 19K splats. Alternate  $10 \times 10 \times 10$  sub-cubes are colored either red or white to create a volumetric

checkerboard effect. The dataset is viewed with a perspective projection; the viewing parameters are identical for each figure.

In Figure 2.25 the voxels are visited with the BTF visibility ordering. This shows a large visibility error in the upper part of the cube. The reason for this error is that the outermost loop of the visibility ordering (see Figure 2.1) renders from the bottom of the cube towards the top. This can be seen in the animation of the rendering in the lower part of Figure 2.25. While the bottom half of the cube is being rendered, the scanlines from the back face are rendered *above* the scanlines from the front face, but while the top half of the cube is being rendered, the scanlines from the back face are rendered *below* the scanlines from the front face. The lower row of the animation shows this error — the scanlines from the back face are overwriting the already-drawn scanlines from the front face. This is the opposite of what is desired, and it is caused by the perspective distortion. Figure 2.25 shows that the BTF visibility ordering is incorrect for this perspective view.

In Figure 2.26 the voxels are visited using the Westover BTF visibility ordering. This shows a visibility error in the right-hand side of the figure. This error is similar to the BTF error shown in Figure 2.25. As the animation shows, it occurs because the WBTF ordering chooses to render from left to right in the outermost loop of the visibility ordering (see Figure 2.3). This results in incorrect visibility in the right-hand side of the cube, again because the scanlines from the back face are overwriting the already-drawn scanlines from the front face. Figure 2.26 shows that the WBTF visibility ordering is incorrect for this perspective view.

In Figure 2.27 the voxels are visited using the Perspective BTF visibility ordering. This does not show a visibility error. The animated sequence shows why — the algorithm chooses what is essentially a Westover-type visibility ordering, but it chooses this ordering separately for the two halves of the cube that lie to either side of the projection of the eye point. Thus the back scanlines never overwrite the front scanlines, and no visibility error occurs.

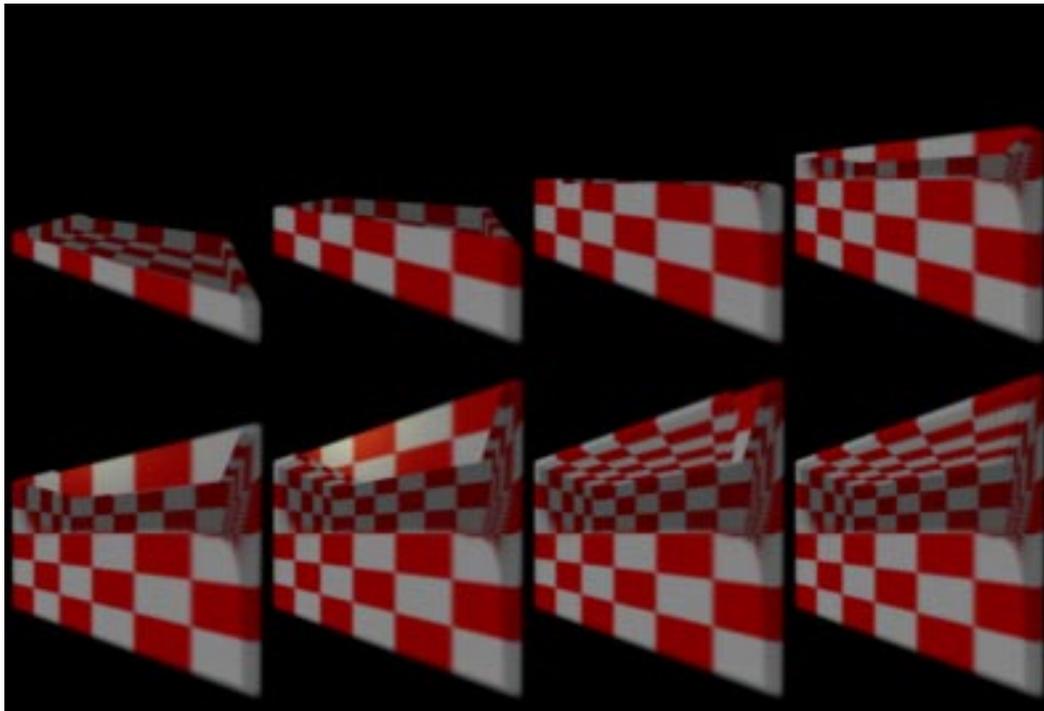
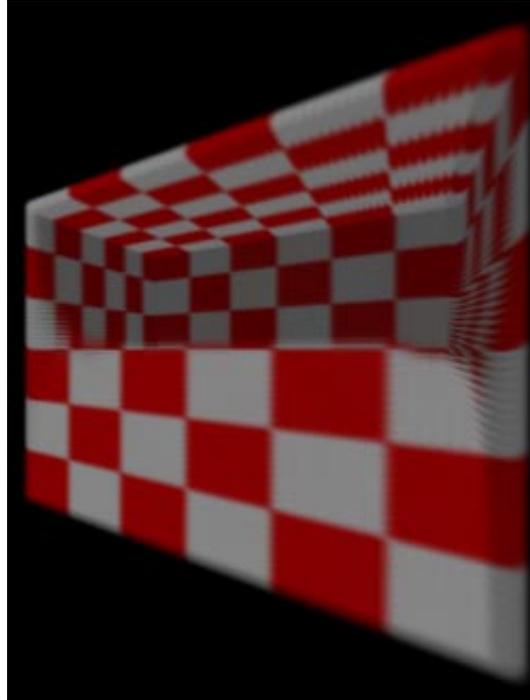


Figure 2.25: (upper) A cube rendered with the BTF visibility ordering. Note the visibility error along the top of the cube. (lower) An animation of the cube being rendered.

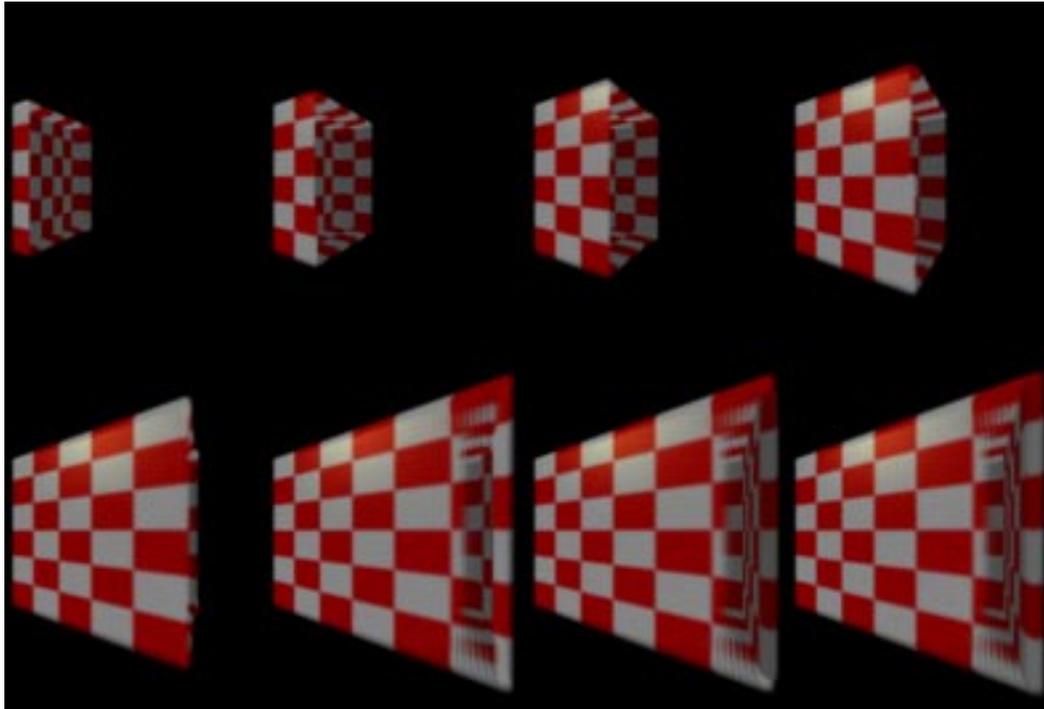
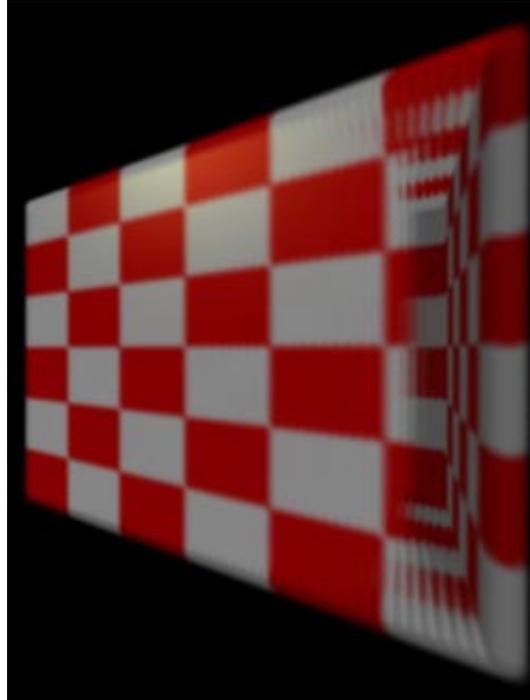


Figure 2.26: (upper) A cube rendered with the WBTF visibility ordering. Note the visibility error along the side of the cube. (lower) An animation of the cube being rendered.

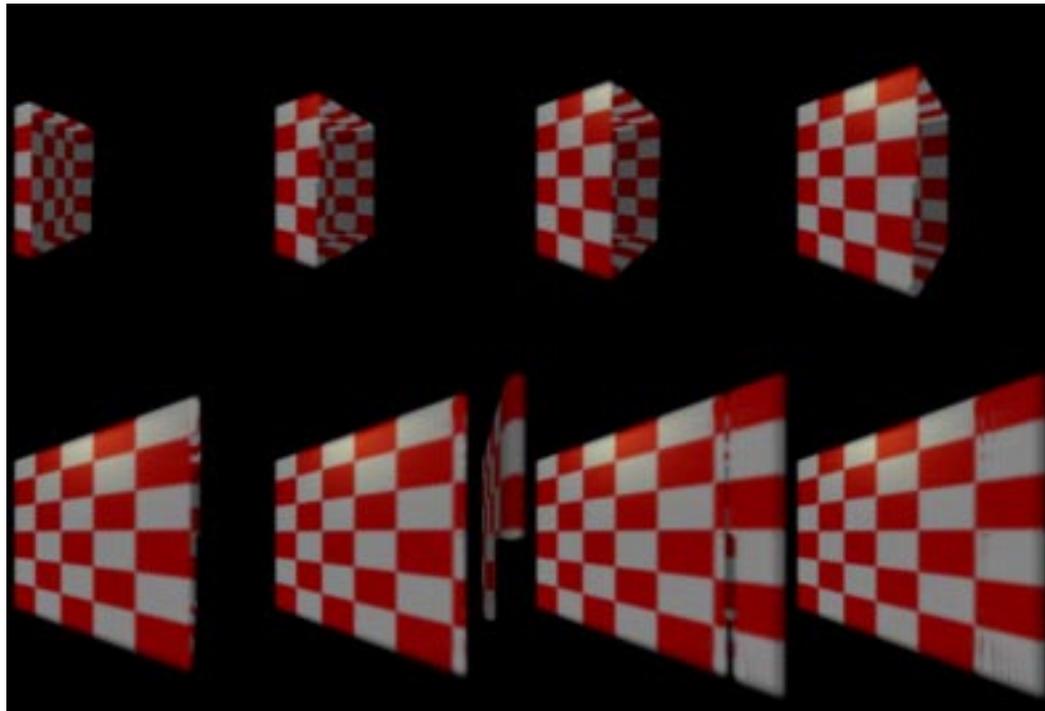
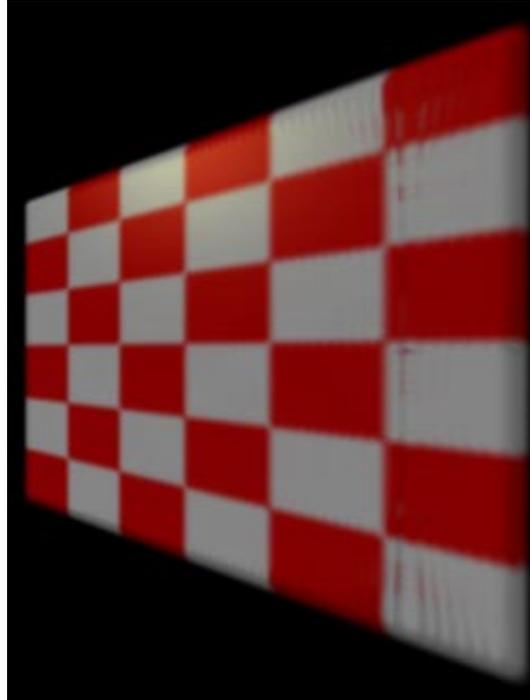


Figure 2.27: (upper) A cube rendered with the PBTF visibility ordering. (lower) An animation of the cube being rendered.

## 2.5 Summary and Future Work

This chapter has presented the Perspective BTF visibility ordering, and given a proof of correctness for the technique. It has compared the PBTF algorithm to two well-known visibility ordering methods commonly used with splatting: the Back-to-Front and the West-over Back-to-Front methods. The chapter has demonstrated that there exist perspective views which the PBTF visibility ordering is able to render correctly, but the BTF and the WBTF visibility orderings are not able to render correctly.

When viewing a scene with a perspective projection, the standard BTF visibility ordering easily shows visibility errors similar to Figure 2.25. Because it has more ways to choose the visibility ordering, the WBTF method is more robust, and does not show visibility errors for many viewpoints which the BTF method is not able to render correctly. However, as Figures 2.25 and 2.26 demonstrate, there are views which neither method can render correctly. The PBTF visibility ordering generates correct visibility because it takes into account the projection of the view point onto the grid, and for each axis visits the grid in a different order on either side of the projected view point. This gives the PBTF ordering enough freedom in choosing traversal directions that it is able to compensate for the perspective distortion.

There are a number of ways in which work presented in this chapter can be extended:

- The PBTF technique could be applied recursively to order a nested hierarchical grid structure like that discussed by Max [68]. It would be interesting to see if the proof can be similarly extended.
- Similar proof techniques might yield more insight into why the BTF and WBTF methods work for orthographic projections but not for perspective projections.

## CHAPTER 3

# AN ANTI-ALIASING TECHNIQUE FOR SPLATTING

### 3.1 Introduction

As currently implemented (Section 1.2), the splatting algorithm does not correctly render cases where the volume sampling rate is higher than the image sampling rate (e.g. more than one voxel maps into a pixel). This situation arises with orthographic projections of high-resolution volumes, as well as with perspective projections of volumes of any resolution. The result is potentially severe spatial and temporal aliasing artifacts. Volume ray casting and shear-warp algorithms avoid these artifacts by employing reconstruction kernels which vary in width [54, 76, 80]. Unlike ray casting and shear-warp algorithms, existing splatting algorithms do not have an equivalent mechanism for avoiding these artifacts. This chapter proposes such a mechanism, which delivers high-quality splatted images.

To put the technique in perspective, first Section 3.2 discusses previous work in the area of anti-aliasing. The method itself is given in Section 3.3, followed by results in Section 3.4 and a discussion of future work in Section 3.5.

### 3.2 Previous Work

Aliasing is a fundamental problem in computer graphics, but its basis in the field of signal processing is well understood. Because the problem is so pervasive, aliasing phenomena have motivated an enormous quantity of research. This section reviews some of the most

important previous work in anti-aliasing. Because of the amount of previous work in this area, no attempt is made at any sort of complete cataloging. Instead, this section identifies the broad research categories in the area, and points out a few seminal papers in each category.

All anti-aliasing methods attempt to solve the *anti-aliasing integral*. This integral, which solves the aliasing problem, is:

$$s(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} i(u + x, v + y) f(u, v) du dv, \quad (3.1)$$

where  $i(x, y)$  is the original image,  $f(u, v)$  is the anti-aliasing *filter kernel* or *point-spread function*, and  $s(x, y)$  is the resulting anti-aliased, filtered image. Anti-aliasing techniques attempt to find accurate and efficient solutions to this integral.

Existing anti-aliasing methods can be broken into two broad categories, depending on how they try to solve Equation 3.1: *analytic techniques*, and *point-sampling techniques*.

### 3.2.1 Analytic Techniques

Analytic techniques solve the anti-aliasing integral using numerical methods. They are commonly implemented as part of continuous visible surface algorithms, because these algorithms provide a description of the scene geometry in the required form — usually at the precision of the machine’s floating-point representation.

An analytic anti-aliasing technique is presented by Crow [22] in one of the earliest anti-aliasing papers. He describes the aliasing problem in terms of signal processing theory, and suggests several anti-aliasing filter kernels. He also gives techniques to integrate anti-aliasing with hidden surface computation, and how to only apply the filtering where necessary (such as along surface edges, silhouette edges, and when rendering a polygon which is smaller than a pixel). In another early paper, Catmull [9] performs polygon clipping for all the polygons that lie under a pixel, and then numerically calculates the areas of the visible polygons. His technique is equivalent to filtering the scene geometry with a box kernel.

Feibush et al. [28] give a method which exploits the symmetry inherent in a radially symmetric filter to efficiently implement the convolution in Equation 3.1 with a 2D lookup

table. This technique allows arbitrarily complicated kernels without an increase in rendering time. Catmull [10] gives an algorithm which is similar to his earlier technique [9], but uses the Feibush et al. method to efficiently utilize a higher-quality Gaussian filter. Turkowski [92] gives an even more efficient method that only requires a 1D lookup table. And Abram et al. [1] give a similar table-lookup technique which also re-orders the convolution so that the filtered pixels are computed in object-order instead of image-order.

Because numerically solving Equation 3.1 is complicated, most analytical anti-aliasing techniques limit scene geometry to simple primitives like polygons and line segments. Furthermore, the polygons are usually flat-shaded or at most have some form of interpolated shading. In general, analytically filtering the patterns from texture mapping is too complex, although Norton et al. [75] give just such an algorithm that operates in the frequency domain. The usefulness of this technique is limited by the need to describe the texture patterns by a truncated Fourier series, however.

Some methods calculate the analytical integral where it is easy to do so (such as along a silhouette edge) and approximate the integral where it is difficult to calculate analytically (such as when there are many overlapping polygon fragments under one pixel). One such method is the *a-buffer* technique, first described by Carpenter [8]. This method replaces the complicated pixel fragment visibility calculations of Catmull [9]. Instead discrete approximations of the pixel fragments are stored in bit masks, which afford very efficient computations by bitwise operations. Fiume et al. [29] give a similar algorithm, which uses a lookup table of subpixel coverage masks to calculate an approximate box filtering of polygon edges. However, the method does not always calculate the correct subpixel visibility, and so the filtering quality varies.

### 3.2.2 Point-Sampling Techniques

Many popular visible surface algorithms, such as ray-tracing or z-buffering, only provide point samples of the projected image, and therefore do not provide enough information to attempt a numerical solution to the anti-aliasing integral. Also, when texture mapping from a discrete texture, by definition discrete texture samples are all that are available.

However, Equation 3.1 can still be estimated by point samples. This is the basis of point-sampling anti-aliasing techniques.

### 3.2.2.1 Texture Mapping

The aliasing problem is very acute in texture mapping, especially on polygons viewed with a perspective projection that disappear into the horizon. Anti-aliasing is considered from the very first work on texture mapping [47, page 305]. Because aliasing is so prevalent, almost all texture mapping implementations include anti-aliasing considerations. Texture-mapping anti-aliasing techniques can be broken into two categories: *direct convolution*, and *prefiltering*.

**Direct Convolution:** These techniques center an anti-aliasing filter kernel on each pixel and then convolve the kernel with those texture samples that lie under the kernel's footprint. This results in very accurate but expensive filtering. Feibush et al. [28] give the general technique. Texture points are transformed into screen space and filtered with an arbitrary radially symmetric kernel. The technique of Perny et al. [79] is similar. They treat each inverse-transformed pixel as an ellipse in texture space, and use a windowed sinc function as a kernel.

Greene and Heckbert [38] and Heckbert [44] give the *elliptical weighted average* filter technique. This also treats inverse-transformed pixels as ellipses in texture space, but unlike Feibush et al. and Perny et al. [28, 79] transforms the whole filter kernel to texture space and does the convolution there. They recommend a Gaussian kernel, but because the filter is implemented with a look-up table any kernel shape is supported.

**Prefiltering:** The direct convolution techniques given above are accurate, but they are expensive. For perspective images, particularly a texture mapped polygon that disappears into the horizon, the number of texture samples that must be filtered to calculate the color of one screen pixel can number in the thousands [43]. When a texture map is used many times in an image (either because it is mapped to many items or because it is tiled across

a surface), it makes sense to prefilter the texture map, so that fewer texture samples are required to shade a pixel.

Williams [102] prefilters the texture into a pyramid data structure called a *mip-map*, where successive pyramid levels are 1/4th the size of previous levels. This allows constant-time integrations across square regions of the texture map. This was generalized by Crow [24], who prefilters the texture into a *summed area table*, where each table entry is the sum of the rectangular texture region below and to the left of the entry. This allows constant-time integrations across rectangular regions of the texture map. Glassner [35] extends summed-area tables in two ways. First, he proposes using additional tables which hold the sums of other shapes besides rectangles. This allows tighter bounding shapes than axially-aligned rectangles at the cost of the additional storage for multiple summed area tables. Second, he gives a method of approximating the pixel's preimage to an arbitrary degree of precision. This improves the quality of the filtering, but no longer texture maps each pixel in constant time.

With this exception all the above techniques texture each pixel in constant time. However, they are all equivalent to a box filter kernel, so the filtering quality is not as high as the more expensive direct convolution methods.

Heckbert [42] gives another extension of summed-area tables: he shows how the technique can be generalized, through repeated integration of the texture, to yield all the b-spline kernels (box, triangle, quadratic, cubic, and so forth). This results in higher-quality constant-time filtering, but is still limited to rectangular regions of the texture map.

### 3.2.2.2 Supersampling

The *supersampling* techniques address the aliasing problem by calculating an intermediate image at a higher resolution than the final image, and then forming the final image by some sort of averaging operation on the intermediate image. Unlike the other anti-aliasing techniques discussed above, supersampling can address all forms of aliasing in an image: both jagged edges and aliased textures. There have been several supersampling techniques:

**Regular Supersampling:** Simply increasing the sampling rate reduces but does not solve the aliasing problem, and it greatly increases the expense of rendering an image. In a comparison study Crow [23] found that analytic prefiltering algorithms [9, 22] had better performance than several different types of supersampling.

**Adaptive Supersampling:** Supersampling is not only expensive, but for typical images it is only really needed for a small fraction of the pixels. Adaptive supersampling techniques attempt to detect where supersampling is needed and only take additional samples at these locations. Whitted [100] first described the technique in the context of ray tracing.

**Stochastic Sampling:** Instead of taking the additional samples on a regular grid, these techniques take them on a grid which has been stochastically distributed. This turns what would otherwise be high-frequency aliasing patterns into incoherent noise, while not having much of an effect on the low-frequency image content. Because incoherent noise is not very noticeable to the human visual system and because supersampling techniques address all forms of aliasing, to date stochastic sampling has been the most generally successful anti-aliasing technique. It was first described by Cook [18] in the context of distributed ray-tracing, as well as by Dippé and Wold [25]. Lee et al. [56] introduce the concept of *stratified sampling*, where sample positions are calculated and the samples are weighed on the basis of a local estimate of scene variance. And Cook et al. [19] describe a system which uses stochastic sampling with a z-buffer.

### 3.3 The Anti-Aliasing Technique

This section describes the splatting-based anti-aliasing method. Section 3.3.1 describes the different spaces used in the rest of the chapter, and Section 3.3.2 describes why anti-aliasing is needed for volume rendering algorithms. Section 3.3.3 develops an expression (Equation 3.7) which, if satisfied by a given volume rendering algorithm, indicates that the algorithm will not contain the aliasing artifacts that arise from the resampling phase of the rendering process. The anti-aliasing method itself is described in Section 3.3.4.

Section 3.3.5 shows that the method satisfies the equation developed in Section 3.3.3, and consequently argues that the method is correct.

### 3.3.1 The Different Spaces

**World space**  $(x_w, y_w, z_w)$  is the space in which the volume raster is defined. In a viewing calculation, world space is first transformed into **eye space**  $(x_e, y_e, z_e)$ . This is accomplished by concatenating a transformation matrix with a rotation matrix; the result is that the eye point is at  $(0, 0, 0)$  and is looking down the positive z-axis. Such a transformation can be specified as

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ t_1 & t_2 & t_3 & 1 \end{bmatrix}, \quad (3.2)$$

where  $r_{11}-r_{33}$  specifies the rotation and  $t_1-t_3$  specifies the translation. Note that because the upper  $3 \times 3$  Jacobian submatrix is orthogonal, this is a **rigid body transformation** and therefore preserves parallelism of lines, length of lines, and angles between lines [30, page 207].

For the purpose of this chapter the most important property of this transformation is that, when going from world to eye space, the sample spacing of the volume raster is preserved. This means that if we give an argument that depends upon the uniformity of the sample spacing, this argument can be made in either world or eye space.

**Perspective space**  $(x_p, y_p, z_p)$  is the eye space after a perspective transform. It can be written as

$$x_p = \left(\frac{D}{h}\right) \cdot \frac{x_e}{z_e} \quad (3.3)$$

$$y_p = \left(\frac{D}{h}\right) \cdot \frac{y_e}{z_e} \quad (3.4)$$

$$z_p = F \left(\frac{1 - D/z_e}{F - D}\right), \quad (3.5)$$

where  $D$  is the distance to the near clipping plane,  $F$  is the distance to the far clipping plane, and  $h$  is one-half the dimension of the viewing plane (see Figure 3.1). Here we are assuming a square viewport, and that the near clipping plane and the viewing plane are coincident. This notation follows Watt and Watt [96].

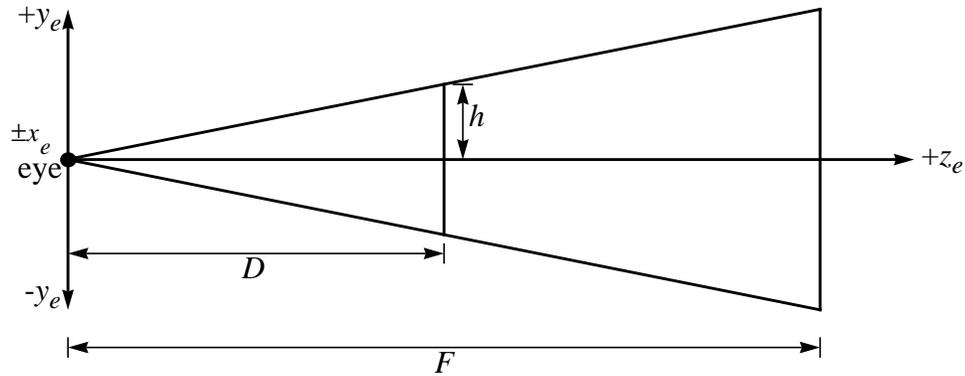


Figure 3.1: Specification of the viewing frustum. The x-axis goes into and out of the page.

Note that the perspective transformation is non-linear. When implemented, it is usually factored into a linear part (implemented with a matrix multiplication as part of the rendering pipeline) and a non-linear part (implemented by the perspective division). This space is also called *screen space* or *device space*.

### 3.3.2 The Need for Anti-Aliasing in Volume Rendering

The process of volume rendering is based on the integration (or composition), along an *integration grid*, of the volume raster (Figure 3.2). This integration grid is composed of *sight projectors* (or *rays*) which pass from the eye point, through the view plane, and into the volume raster. Before this integration can occur, the volume raster has to be reconstructed and then resampled along the integration grid. This is illustrated in Figure 3.2 for a perspective view of the volume, where the volume raster is shown as a lattice of dots, and the integration grid is shown as a series of rays, cast through pixels, which traverse the volume raster. Figure 3.2a shows the scene in eye space, where the eye is located at point  $(0, 0, 0)$  and is looking down the positive z-axis (denoted  $+z_e$ ). The perspective projection means the integration grid diverges as it traverses the volume. Figure 3.2b shows the same scene in perspective space, after perspective transformation and perspective division. Here the volume raster is distorted according to the perspective transformation,

and the integration grid lines are parallel. Because of this the eye is no longer located at a point, but can be considered the plane  $z_p = 0$ .

The reconstruction and resampling of the volume raster onto the integration grid has to be done properly to avoid aliasing artifacts. Preferably aliasing is avoided by sampling above the Nyquist limit, but if this is not possible then aliasing can also be avoided by low-pass filtering the volume to reduce its frequency content. For an orthographic view this low-pass filtering must be applied to the entire volume, but for a perspective view low-pass filtering may only be required for a portion of the volume. This can most easily be seen in perspective space (Figure 3.2b). Note that there is a distance along the  $+z_p$  axis, denoted  $k_p$ , where the sampling rate of the volume raster and the sampling rate of the integration grid are the same. Before this point there is less than one voxel per pixel, and after this point there is more than one voxel per pixel. When there is more than one voxel per pixel the volume raster can contain higher frequency information than the integration grid can represent, and so aliasing is possible. In the next section this concept is developed into an equation.

The same thing is true in eye space (Figure 3.2a). Here there is an equivalent distance along the  $+z_e$  axis, denoted  $k$ , where the sampling rates of the volume raster and the integration grid are the same (note that in general  $k \neq k_p$ , but the two distances are related by the perspective transformation). Aliasing artifacts can occur after  $k$ , when the distance between adjacent rays is greater than one voxel.

Volume ray casting algorithms generally perform the reconstruction in eye space. They avoid aliasing by employing reconstruction kernels which become larger as the rays diverge [3, 41, 76, 80]. This provides an amount of low-pass filtering which is proportional to the distance between the rays. Splatting algorithms generally perform the reconstruction in perspective space. Unlike ray casting algorithms, existing splatting algorithms do not have an equivalent mechanism to avoid aliasing. This chapter proposes such a mechanism.

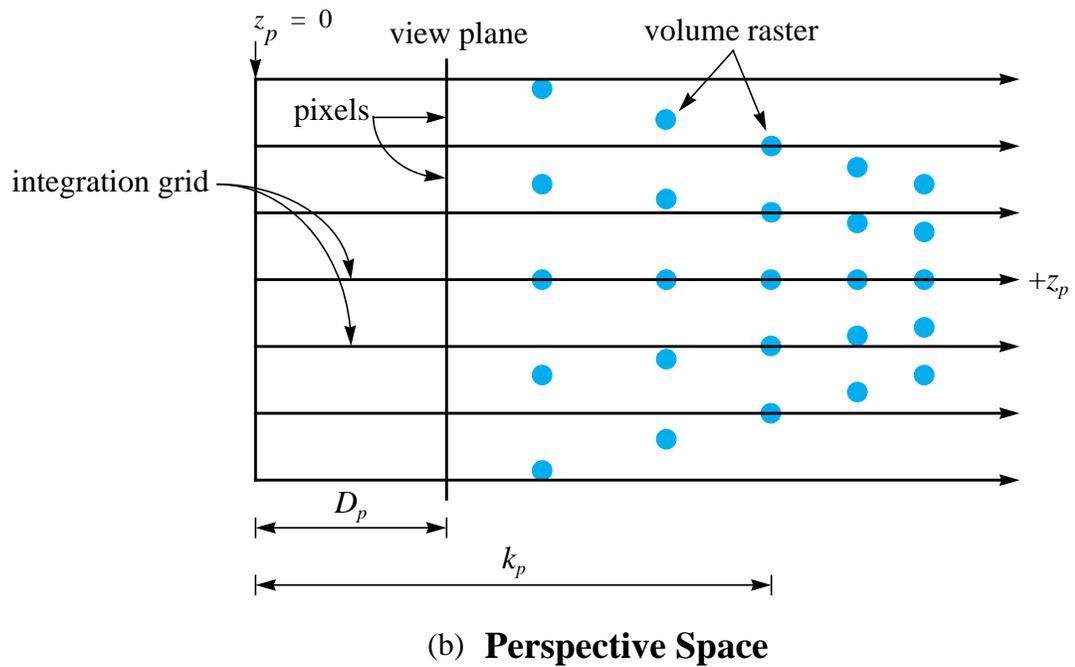
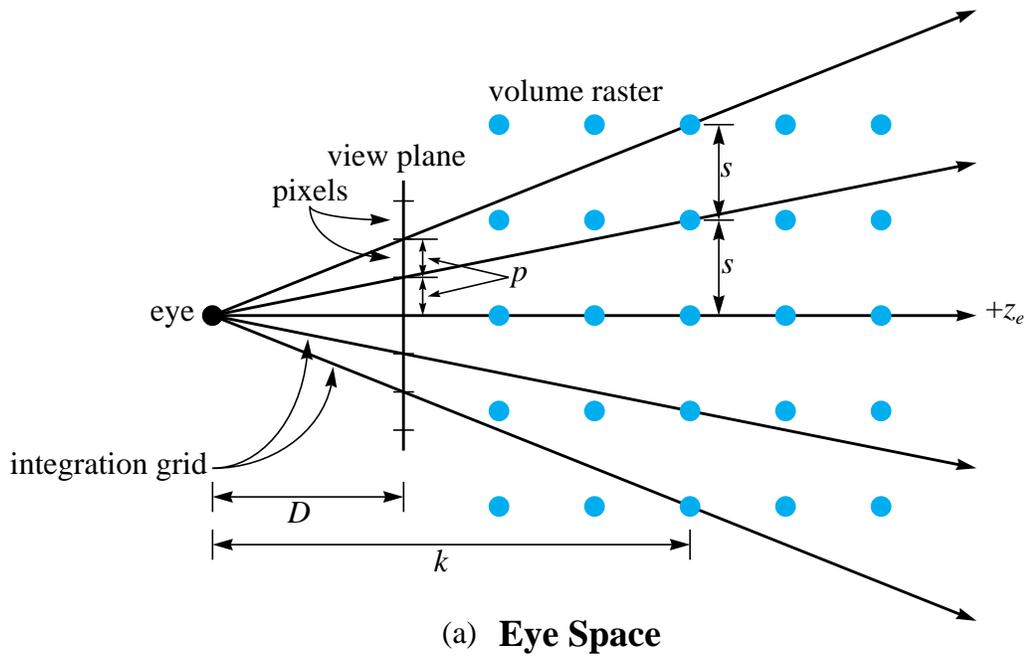


Figure 3.2: Resampling the volume raster onto the integration grid. (a) In eye space. (b) In perspective space.

### 3.3.3 Necessary Conditions to Avoid Aliasing

This section gives the conditions which are necessary for the volume rendering resampling process to avoid introducing aliasing artifacts into the integration grid samples. Let  $s$  be the volume raster grid spacing (Figure 3.2a), and  $\rho = 1/s$  be the volume raster sampling rate. The volume raster contains aliasing when either

1. the sampled function is not bandlimited, or
2. the function is bandlimited at frequency  $w$  but the sampling rate  $\rho$  is below the Nyquist limit:  $\rho < 2w$ .

If the first condition holds then aliasing will be present no matter how large  $\rho$  becomes. However, if the function is bandlimited at  $w$ , then as long as

$$\rho \geq 2w \tag{3.6}$$

there is no aliasing in the volume raster. Assuming that Equation 3.6 is true, the resampling process needs to resample the volume raster onto the integration grid in a manner that guarantees that no aliasing is introduced.

Let  $\phi$  represent the sampling frequency of the integration grid. For a perspective projection the integration grid diverges (Figure 3.2a) and therefore  $\phi$  is a function of distance along the  $z_e$  axis:  $\phi = \phi(d)$ , where  $d$  is the distance. For an orthographic projection  $\phi$  can still be expressed as a function, but it will have a constant value. As illustrated in Figure 3.2, at  $k$  the sampling rates of the volume raster and the integration grid are the same:  $\phi(k) = \rho$ .

The distance  $k$  means there are two cases to consider:

**Case 1:**  $d < k$ . This is the case for the portion of the grid in Figure 3.2a before  $k$ . Here

$\phi(d) > \rho$ , and if Equation 3.6 holds then  $\phi(d) > 2w$  and there is no aliasing.

**Case 2:**  $d \geq k$ . This is the case for the portion of the grid in Figure 3.2a after  $k$ . Here

$\phi(d) \leq \rho$ , and so it may be that  $\phi(d) < 2w$ . If this is the case, then the integration grid will contain an aliased signal once  $d$  is large enough.

This argument shows that, assuming Equation 3.6, volume rendering algorithms do not have to perform anti-aliasing as long as Case 1 holds (before the distance  $k$  is reached). However, once Case 2 holds (after  $k$  is reached), to avoid aliasing it is necessary to low-pass filter the volume raster. Ideally the amount of this filtering is a function of  $d^2$ , and reduces the highest frequency in the volume raster from  $w$  to  $\tilde{w}(d)$ . To avoid aliasing there must be enough low-pass filtering so that

$$\phi(d) \geq 2\tilde{w}(d). \quad (3.7)$$

If true, this equation indicates that all the resampling occurs above the Nyquist limit. If it can be shown to be true for a particular volume rendering technique, then the equation demonstrates that the technique does not introduce aliasing artifacts when resampling from the volume raster onto the integration grid.

### 3.3.4 An Anti-Aliasing Method for Splatting

As mentioned in Section 3.3.2 above, volume ray casting algorithms avoid aliasing by using reconstruction kernels which increase in size as the integration grid rays diverge, which satisfies Equation 3.7. This section gives a similar anti-aliasing algorithm for splatting.

As shown in Figure 3.2a, at distance  $k$  the ratio of the volume raster sampling frequency  $\rho$  and the integration grid sampling frequency  $\phi(d)$  is one-to-one.  $k$  can be calculated from similar triangles:

$$k = s \frac{D}{p}, \quad (3.8)$$

where  $s$  is the sample spacing of the volume raster,  $p$  is the extent of a pixel, and  $D$  is the distance from the eye point to the screen.

Figure 3.3 gives a “side view” of splatting as implemented by Westover [97, 98, 99], as well as the anti-aliasing method introduced in this chapter. In Figure 3.3 the y-axis is drawn vertically, the z-axis is drawn horizontally, the x-axis comes out of and goes into the page,

<sup>2</sup>This is because a portion of the volume raster may not require any filtering (Case 1), and for the portion that does require filtering (Case 2), if there is a perspective projection then the amount of filtering required is itself a function of  $d$ .

and figures are shown in both eye space  $(x_e, y_e, z_e)$  and perspective space  $(x_p, y_p, z_p)$ . The top row illustrates standard splatting. As in Figure 3.2,  $D$  is the distance of the view plane from the eye point, and  $k$  is calculated from Equation 3.8. For this example a single row of splats is being rendered, which are equally spaced along the  $z_e$  axis (Figure 3.3a). Each splat is the same size in eye space. Figure 3.3b shows the same scene in perspective space. Here  $D_p$  and  $k_p$  are  $D$  and  $k$  expressed in  $z_p$  coordinates. As expected, because of the non-linear perspective transformation, the splat spacing is now non-uniform along the  $z_p$  axis [96], and the size of the splats decreases with increasing distance from the eye.

The bottom row illustrates the anti-aliasing method. Before  $k$  splats are drawn the same size in eye space (Figure 3.3c). Beginning at  $k$ , splats are scaled so they become larger with increasing distance from the view plane. This scaling is proportional to the viewing frustum, and is given in Equation 3.9 below. Figure 3.3d shows what happens in perspective space. Before  $k_p$  the splats are drawn with decreasing sizes according to the perspective transformation. Beginning at  $k_p$ , all splats are drawn the same size, so splats with a  $z_p$  coordinate greater than  $k_p$  are the same size as splats with a  $z_p$  coordinate equal to  $k_p$ .

Figure 3.4 gives the geometry for scaling splats drawn after  $k$ . If a splat drawn at distance  $k$  has the radius  $r_1$ , then the radius  $r_2$  of a splat drawn at distance  $d > k$  is the projection of  $r_1$  along the viewing frustum. This is calculated by similar triangles:

$$r_2 = r_1 \frac{d}{k}. \quad (3.9)$$

Scaling the splats drawn after  $k$  is not enough to provide anti-aliasing, however. In addition to scaling, the energy that these splats contribute to the image needs to be the same as it would be if they had not been scaled. As shown in Figure 3.4, both splat 1 and splat 2 project to the same sized area on the view plane. Because they are composited into the view plane in the form of two-dimensional “footprint” filter kernels [98], the amount of energy the splats contribute to the view plane is proportional to their areas. The amount of energy per unit area contributed by the splats needs to be the same. This is accomplished by attenuating the energy of splat 2 according to the ratio of the areas of the

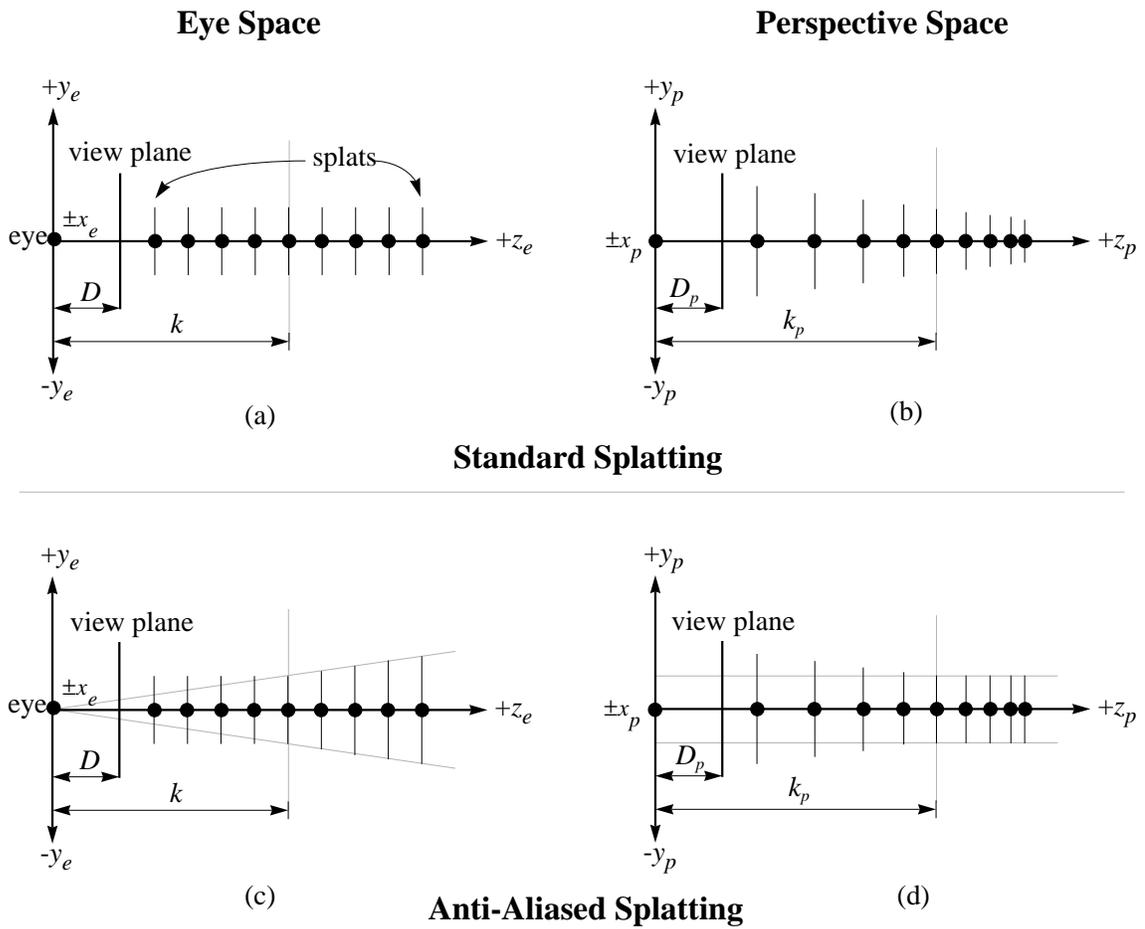


Figure 3.3: A comparison of the standard splatting method (top) with the anti-aliased method (bottom). (a) The standard splatting method drawn in eye space. (b) The standard splatting method drawn in perspective space. (c) The anti-aliased splatting method drawn in eye space. (d) The anti-aliased splatting method drawn in perspective space.

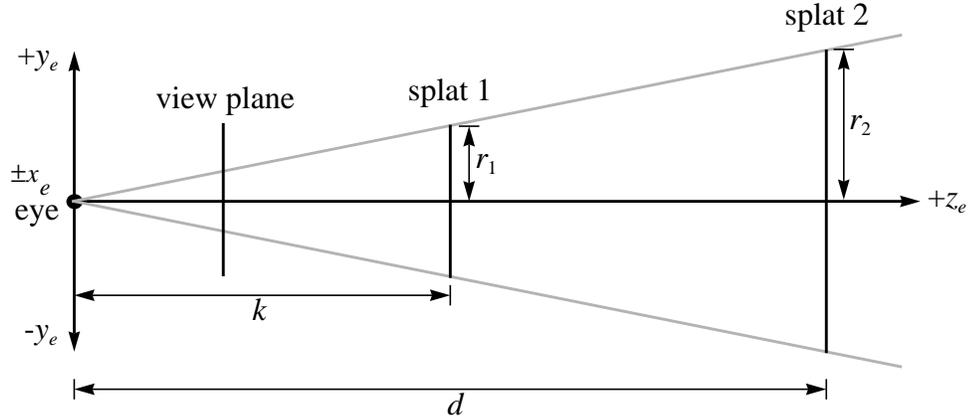


Figure 3.4: The geometry for scaling splats drawn after  $k$ .

splats:

$$E_2 = \frac{A_1}{A_2} E_1, \quad (3.10)$$

where  $A_1, A_2$  are the areas of the splats and  $E_1, E_2$  are some energy measure for the splats. Examples of energy measures include the volume under the splat kernel or the alpha channel of the polygon defining the 2D splat footprint.

Assume for now that the filter kernel is a circle for both splats. Then the areas of the splats are  $A_1 = \pi r_1^2$  and  $A_2 = \pi r_2^2$ . By Equation 3.9 we can express  $A_2$  in terms of  $r_1$ :

$$A_2 = \pi \left( r_1 \frac{d}{k} \right)^2. \quad (3.11)$$

Then

$$E_2 = \left( \frac{\pi r_1^2}{\pi \left( r_1 \frac{d}{k} \right)^2} \right) E_1, \quad (3.12)$$

which can be simplified to

$$E_2 = \left( \frac{k}{d} \right)^2 E_1. \quad (3.13)$$

Although here Equation 3.13 has been derived using circular filter kernels, it can also be derived using any other two-dimensional shape for the kernel, such as an ellipse, square, rectangle, parallelogram, etc.; these all derive the same equation. Equation 3.13 is used to attenuate the energy of all splats drawn after  $k$ .

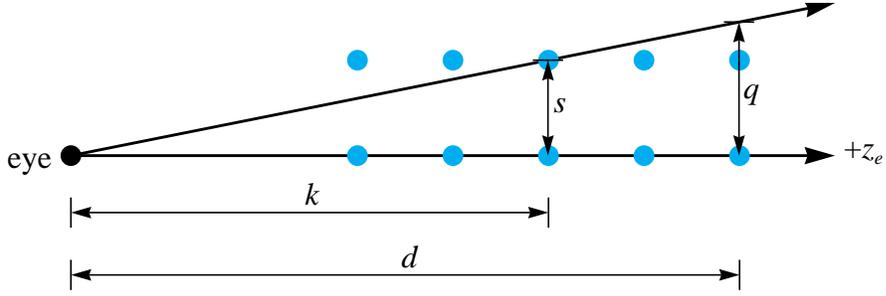


Figure 3.5: Calculating the integration grid sampling frequency.

### 3.3.5 Justification for the Method

This section demonstrates that Equation 3.7 holds for the anti-aliasing technique. It begins by deriving expressions for the two functions in Equation 3.7:  $\phi(d)$  (the integration grid sampling frequency at  $d$ ) and  $\tilde{w}(d)$  (the maximum volume raster frequency at  $d$ ).

The function  $\phi(d)$  is derived with a similar-triangles argument. Consider Figure 3.5, where  $q$  is the integration grid spacing at distance  $d$  from the eyepoint. By similar triangles

$$\frac{s}{k} = \frac{q}{d}, \quad (3.14)$$

which can be written as

$$\frac{1}{q} = \frac{1}{s} \cdot \frac{k}{d}. \quad (3.15)$$

Now  $1/s = \rho$ , and  $1/q$  is simply  $\phi(d)$ , the integration grid sampling frequency at  $d$ . Thus we have

$$\phi(d) = \rho \cdot \frac{k}{d}. \quad (3.16)$$

The function  $\tilde{w}(d)$  can be derived from the scaling property of the Fourier transform [6]:

$$f(at) \iff \left| \frac{1}{a} \right| F\left(\frac{1}{a}\omega\right), \quad (3.17)$$

where “ $\iff$ ” indicates a Fourier transform pair. This shows that widening a function by the factor  $a$  in the spatial domain is equivalent to narrowing the function in the frequency

domain by the factor  $1/a$ . In the anti-aliasing method, the widening for the splat kernels drawn after  $k$  is given by Equation 3.9:

$$a = \frac{d}{k}. \quad (3.18)$$

Thus we have

$$f\left(\frac{d}{k}t\right) \iff \left|\frac{k}{d}\right| F\left(\frac{k}{d}\omega\right), \quad (3.19)$$

which shows that as the splat kernels are widened by  $d/k$ , their frequency components are narrowed by  $k/d$ . Since all the frequencies of the splat kernel are attenuated by  $k/d$ , the maximum frequency  $w$  is attenuated by the same amount, and we have:

$$\tilde{w}(d) = w \cdot \frac{k}{d}. \quad (3.20)$$

Now we are ready to show that the technique satisfies Equation 3.7. We start with Equation 3.6:  $\rho \geq 2w$ , which implies that the volume raster has sampled the function above the Nyquist limit. Multiplying both sides by  $k/d$  we have

$$\rho \cdot \frac{k}{d} \geq 2\left(w \cdot \frac{k}{d}\right), \quad (3.21)$$

which we can write as:

$$\phi(d) \geq 2\tilde{w}(d). \quad (3.22)$$

This derivation says that if the volume raster has sampled the function above the Nyquist limit, the proposed anti-aliasing technique provides enough low-pass filtering so that aliasing is not introduced when the volume raster is resampled onto the integration grid.

## 3.4 Results

Figures 3.6–3.8 show a collection of images obtained from both the standard splatting algorithm and the anti-aliased technique reported in this chapter. For each figure, the image on the top is rendered without anti-aliasing, while the image on the bottom is rendered with anti-aliasing.

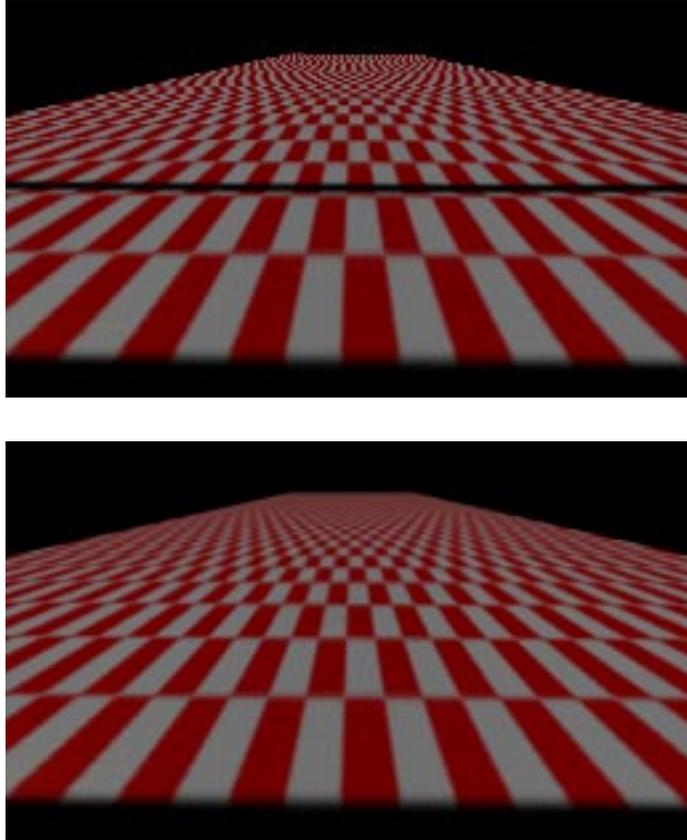


Figure 3.6: Rendered image of a plane with a checkerboard pattern. (upper) Rendered with standard splatting; the black line is drawn at distance  $k$ . (lower) Rendered with anti-aliased splatting.

Figure 3.6 shows a  $498 \times 398 \times 1$  volume consisting of a single sheet, where alternate  $10 \times 10$  squares are colored either red or white to create a checkerboard effect. The resulting dataset contains 198K splats which are rendered into a  $260 \times 150$  image. In the upper image a black line is drawn at the distance  $k$ ; beyond this line there is more than one voxel per pixel. As expected, the upper image shows strong aliasing effects, but these are smoothed out in the lower image.

Figure 3.7 shows a  $512 \times 512 \times 103$  volume containing a terrain dataset acquired from a satellite photograph and a corresponding height field. The resulting dataset contains 386K

splats (this is more than the expected  $512^2$  splats because extra splats are required to fill in the “holes” formed where adjacent splats differ in height). Each column of splats is given the color of the corresponding pixel from the satellite photograph. The dataset is rendered into a  $260 \times 179$  image. In the upper image a black line is drawn at the distance  $k$ . The upper image shows strong aliasing in the upper half of the image (containing about 90% of the data); when animated, these regions show strong flickering and strobing effects. In the lower image these regions have been smoothed out; and although this technique does not directly address temporal aliasing effects, when animated these regions are free of flickering and strobing effects.

Figure 3.8 shows a  $420 \times 468 \times 62$  volume containing a microtubule study acquired from confocal microscopy. The resulting dataset contains 344K splats which are rendered into a  $290 \times 290$  image. Unlike the previous images, which are drawn with a perspective projection, this image is drawn with an orthographic projection. However, because the volume has a higher resolution than the image it is still liable to aliasing effects (all of the splats are drawn beyond the distance  $k$ ). This is shown in the upper image, which contains jagged artifacts that shimmer when animated. In the lower image these effects have been smoothed out; when animated this shimmering effect disappears.

### 3.5 Summary and Future Work

As this chapter has demonstrated, the proposed anti-aliasing technique prevents the aliasing that arises from the reconstruction process when more than one voxel maps into a pixel. However, this is not the only source of aliasing in volume rendering. If the volume raster is not sampled above the Nyquist rate (e.g. if Equation 3.6 does not hold) then the volume samples themselves contain aliasing. This is the case with any volume dataset with contains a sharp boundary or binary classification — for example, the images shown in Figure 3.6 contain this type of aliasing. In addition, the choice of a particular reconstruction kernel always involves a trade-off between aliasing, smoothing, and other artifacts [64].

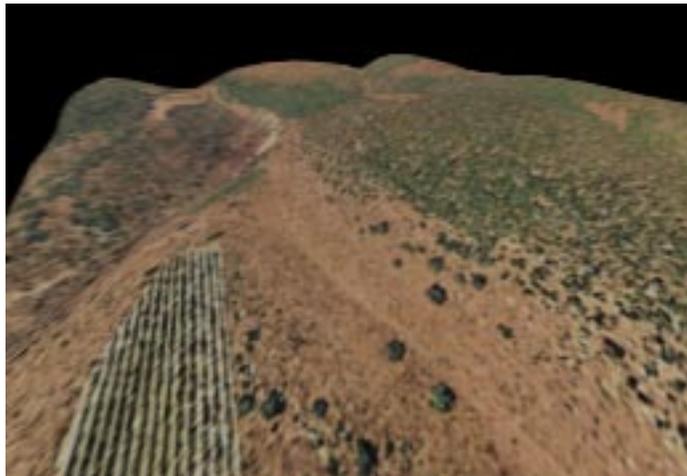
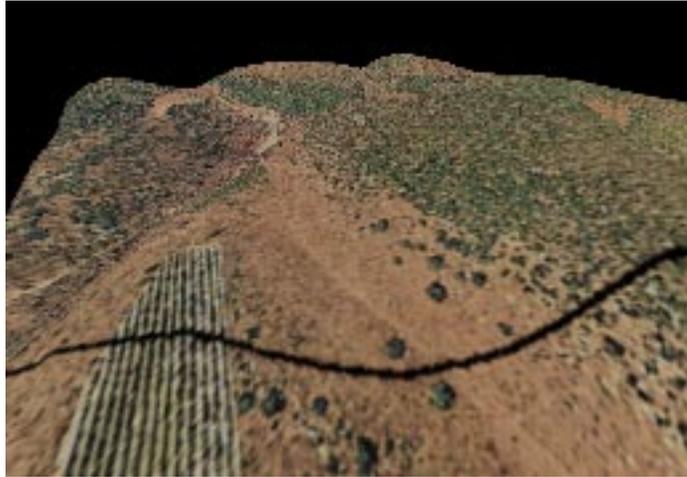


Figure 3.7: Rendered image of a terrain dataset. (upper) Rendered with standard splatting; the black line is drawn at distance  $k$ . (lower) Rendered with anti-aliased splatting.

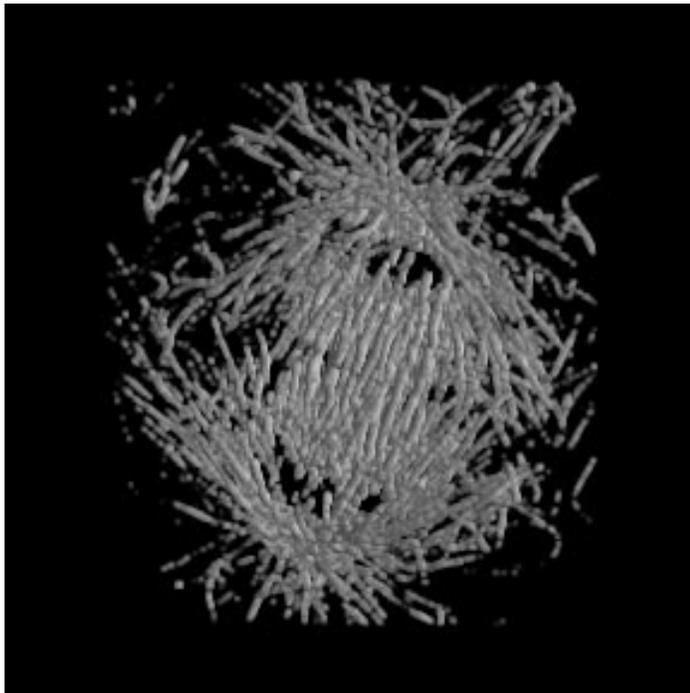
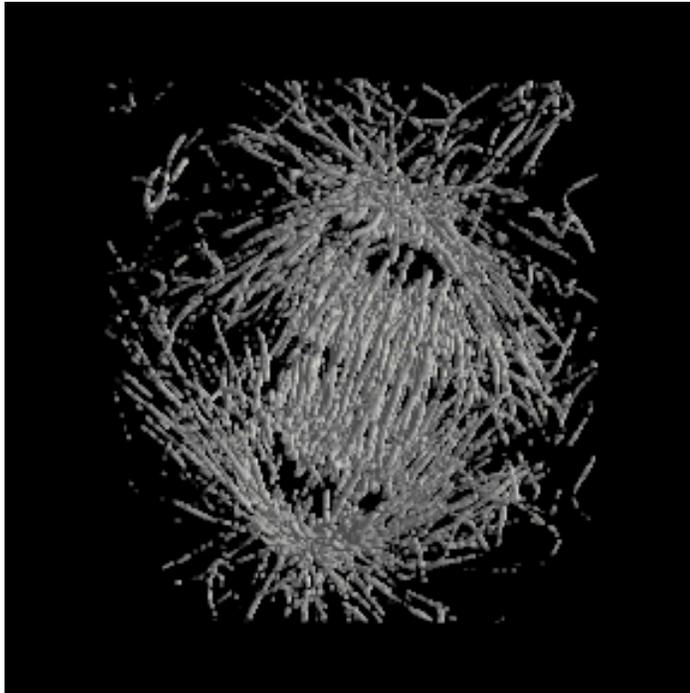


Figure 3.8: Rendered image of a scientific dataset. (upper) Rendered with standard splatting. (lower) Rendered with anti-aliased splatting.

While the general idea of performing anti-aliasing with a source- or texture-space filter is not new [1], this is the first time such a technique has been applied in the context of volume rendering. There are at least two areas for extending the technique: graphics hardware that supports splatting, and texture mapping.

**Graphics Hardware that Supports Splatting:** In this project there is a mismatch between the type of rendering algorithm that was written and the available programming tools. The Silicon Graphics rendering hardware that was used is designed to accelerate scenes using traditional surface graphics primitives such as polygons and spline surfaces; it does not contain an optimized splat primitive. To this end, new hardware architectures which better support operations which are common in volume rendering are needed. Some possible avenues for future work are:

- Extended bitblt-like operators that can be sub-pixel centered and subsequently composited.
- Hardware support for point rendering using different reconstruction kernels (cubic, Gaussian, etc.) with common footprints (circular, elliptical, etc.). Potentially this offers a far more efficient implementation of splatting than hardware texture mapping.
- A higher resolution alpha channel to allow for the accurate accumulation of very transparent splats.
- Splat primitives with automatic size scaling based on their z-depth.

**Texture Mapping:** This technique can be adapted to perform traditional texture mapping of surface-based geometric primitives such as polygons and patches. This would provide an accurate solution to the texture sampling problem — one that is more accurate than either mip-maps [102] or summed area tables [24].

Recall that traditional texture mapping algorithms transform a pixel (or the anti-aliasing filter kernel) from screen into texture space, using the inverse of the viewing transformation (Figure 3.9). In general, the preimage of a square pixel is an arbitrary quadrilateral

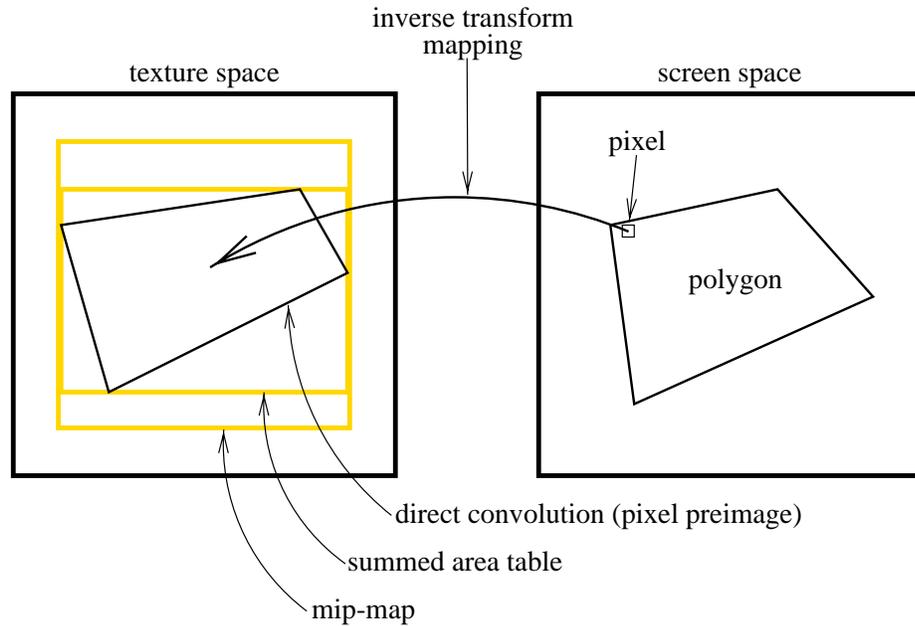


Figure 3.9: Traditional texture mapping, showing the pixel preimage for direct convolution, a summed area table, and mip-mapping.

in texture space [44]. As discussed in Section 3.2.2.1 above, direct convolution texture filtering methods integrate the texture map under this quadrilateral, which gives the color that is applied to the pixel.

While direct convolution methods perform very high-quality texture mapping, they are very expensive and are seldomly used. There are at least two reasons why direct convolution methods are so expensive: 1) each pixel must be inverse transformed, and 2) many areas of the texture map must be accessed multiple times. This second point is illustrated in Figure 3.10, which shows what can happen when two adjacent pixels are texture mapped. In this example, the preimages of the two pixels overlap in texture space. This means that the overlapped area must be accessed twice.

Most current texture mapping implementations use either mip-maps or summed area tables [96]. As shown in Figure 3.9, a summed area table integrates the texture map in the axis-aligned rectangular bounding box of the pixel preimage, while a mip-map integrates

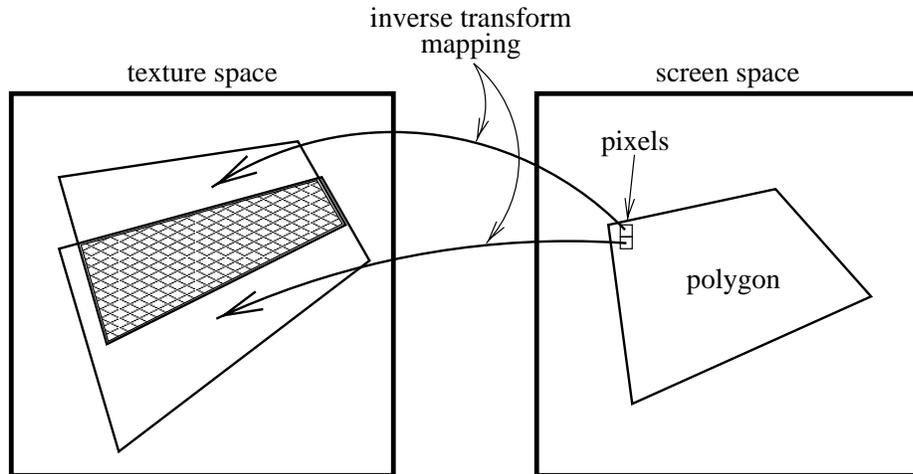


Figure 3.10: Traditional texture mapping, showing the preimages of two adjacent pixels.

the texture map in the axis-aligned square bounding box. However, this means that both methods integrate over a larger area than is required, and hence they perform more low-pass filtering than is actually needed. This results in a texture that is more blurry than necessary.

The anti-aliasing method discussed in this chapter can easily be modified to perform texture mapping (Figure 3.11). In this case, every texture sample is mapped according to the forward viewing transformation and composited into the image plane, using a variably-sized, attenuated filter kernel similar to the one given for splatting in Section 3.3.4 above. This has the following advantages for texture mapping:

- The resulting filtering quality is equivalent to high-quality direct convolution methods, because each pixel only receives the contribution of those texture samples that lie in the pixel's preimage. In particular, the filtering would be higher quality than either mip-mapping or summed area tables.
- The method is less expensive than direct convolution methods, because 1) each texture sample is forward transformed, which is less expensive than inverse-transforming each pixel, and 2) no texture sample has to be accessed more than once. This

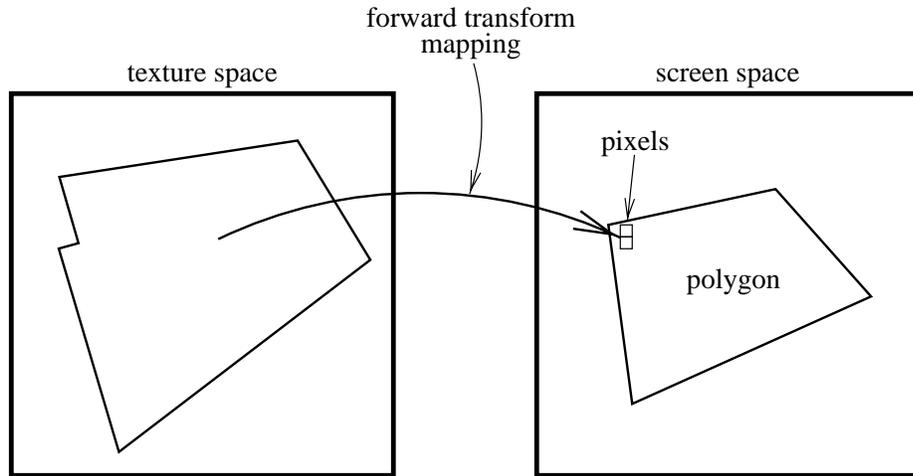


Figure 3.11: Texture mapping with the new method discussed in this chapter.

type of *coherent* texture access has been used in the REYES rendering architecture for performance reasons [19].

This is an exciting area for future work.

## CHAPTER 4

# A MOTION BLUR TECHNIQUE FOR OBJECT-ORDER RENDERING OF DISCRETE OBJECTS

### 4.1 Introduction

Any computer-generated animation must deal with the phenomena of *temporal aliasing*. This is a set of visual artifacts that arises because each animated frame is generated with the equivalent of a camera with an instantaneous shutter. The same set of visual artifacts are particularly evident in old stop-action movies such as the original *King Kong*. As given by Joy et al. [47, page 265], the main artifacts of temporal aliasing are:

- aliasing of high-speed motion — the classic backwards spinning wagon wheel in movies,
- strobing — fast moving objects appear to jump in discrete steps,
- scintillation — small objects appear to blink on and off,
- “crawling ant” effect — apparent motion along the edges of polygons as they slowly move,
- stretching and shrinking — a slowly moving object just a few pixels wide will appear to stretch and shrink by one pixel, and

- beating — a vertically moving object on an interlaced display will appear to beat in synchrony with the interlacing.

These effects can be reduced or eliminated by introducing *motion blur* into each animated frame. When filming natural scenes motion blur is introduced automatically: because the shutter of a film camera is open for a finite amount of time, any objects in motion move during the length of time the camera shutter is open. This means the projection of the moving objects on the film is blurred in the direction of motion. This blurring is termed *motion blur*. With computer-generated scenes (as well as with stop-motion animation) the motion blur does not happen naturally, but must be explicitly added to each frame.

This chapter introduces a new technique for adding motion blur to any object-order technique for rendering discrete objects. The technique is implemented in the context of splatting.

To put the technique in perspective, Section 4.2 discusses previous work in the area of motion blur. Section 4.3 describes the new motion-blur technique. This is followed by Section 4.4, which gives results and discusses the technique, and then by Section 4.5, which outlines some areas for future work.

## 4.2 Previous Work

The literature on motion blur can be broken into two broad categories, based on how the techniques attempt to solve the motion blur problem: *analytic methods* and *discrete methods*. Analytic methods attempt a solution to the problem which is represented at machine precision, and then directly render the motion-blurred image into the frame buffer. Discrete methods sample the scene at discrete time intervals and combine the resulting samples or images to create a composite, motion-blurred image.

### 4.2.1 Analytic Methods

**Continuous Function:** Korein and Badler [53] attempt to find continuous functions that describe the motion of object attributes (e.g. object location, object extent, object color, object depth, etc.) as a function of time. Because of the complexity of this process, the

technique is only defined for disks or polygons, and only linear motion is supported. They use a modified scanline algorithm which determines what objects are visible at each pixel at each time. This technique amounts to integrating each object over its path during each frame.

**Frequency Domain:** Norton et al. [75] describe a technique for anti-aliasing textures in both spatial and temporal dimensions. Their technique requires that the texture be approximated in the frequency domain by a Fourier series. This allows the expensive convolution required for anti-aliasing to be replaced with a cheaper multiplication. This multiplication is approximated by taking the first few terms of the sinc power series, with additional terms being clamped to the average value of the sinusoid. They achieve temporal anti-aliasing by approximating the movement of the projection of each pixel during a frame with a vector. They then generalize their 2D sampling technique to 3D, and use the motion vector for the extra dimension. The net result is box filtering to create an anti-aliased texture in the spatial domain, and box filtering to approximate integrating the object over its path during each frame.

**Post-Process:** Potmesil and Chakravarty [82] describe a system where motion-blurred objects are added to an image as a post-process. A recursive ray-tracing algorithm is used to render a static scene. Those objects which move during the exposure time of the image are marked and not rendered into the scene; the resulting image is the scene with all the moving objects absent. The projection of the moving objects are then integrated along their parametric motion paths by a convolution which occurs either in the spatial or frequency domain. Then the moving objects are alpha-channel composited into the image with a depth-and-time buffer. Because the objects are added as a post-process, the algorithm cannot correctly handle cases where blurred objects intersect each other.

Max and Lerner [69] describe a similar algorithm with a very efficient implementation. Objects are placed into groups which do not intersect in  $z$ , and rendered into separate images. Motion blur is achieved by blurring the entire raster in the direction of motion, using an efficient approach that first skews the image so that the direction of motion is

either entirely horizontal or vertical, blurs along this direction, and then unskews the image back to its original orientation. The separate images are then alpha-composited in back-to-front order. The motion blur is limited to straight translations.

In the same paper Max and Lerner [69] describe a similar algorithm which computes the exact (not necessarily linear) motion blur for vectors drawn as a chain of linked quadrilaterals. And in [66], Max shows how to extend the technique of [69] to render polygons while solving several outstanding problems.

**Pixel Filtering:** Catmull [10] describes a pixel-independent rendering algorithm. In this algorithm, shading and texturing are applied as separate processes to all polygons regardless of visibility. The polygons are sorted according to  $x$  and  $y$  extents. Then the algorithm loops pixel by pixel, and for each pixel determines the visible polygons using  $z$ -sorting and Weiler-Atherton clipping. Spatial anti-aliasing is performed using a weighed Gaussian filter, performed efficiently using the technique of Feibush et al. [28]. Motion blur is addressed by elongating the circular filter footprint for moving polygons along the direction of motion, or equivalently by compressing the polygon along the direction of motion for each pixel which the object passes over. Because this calculation is made separately for each pixel, the motion blur respects visible surface calculations, and it allows the polygon vertices to have different motions.

**Full Analytic:** Grant's algorithm [37] is a combination of the Catmull [10] and the Feibush et al. [28] algorithms. It models moving 3-space polygons as stationary 4-space polyhedra in image space. It uses a scanline extension of Catmull's algorithm to efficiently solve the time and visible surface problem at each pixel, utilizing scanline coherence in both the spatial and temporal domains. It then uses a 3D extension of Feibush et al.'s 2D algorithm to efficiently filter the resulting 3-space swept polygons using a spherical filter. The algorithm thus performs exact anti-aliasing computations in both the spatial and time domains. However, it is cumbersome in that the polygons must be specified in image space.

**Particle-Based:** Reeves [83] uses motion-blurred particles in his seminal work on particle systems. Each particle is motion blurred by drawing an anti-aliased line segment between the particle's initial position and its position about half-way through the frame. This simulates what happens with a real motion picture camera, which captures approximately half the motion that occurs between frames [83].

#### 4.2.2 Discrete Methods

**Distributed Super-Sampling:** To date the most successful motion blur technique, in the sense of accurately simulating the widest variety of motion blur phenomena, is *distributed super-sampling*, which was first implemented in the context of *distributed ray-tracing* [17, 18, 25, 56]. Each ray point samples the rendering integral [51], thereby approximating the integral with a Monte Carlo technique. Motion blur is achieved by stochastically distributing the point samples in the time spanned by the frame. This amounts to integrating the position of the objects across their arc of movement during each frame. Unlike the other methods reviewed here, distributed ray tracing correctly calculates motion blur for rotating objects, shading and highlights, shadows, intersecting objects, and moving objects which occlude or are occluded by non-moving objects. The drawback is that distributed ray tracing is a very expensive rendering algorithm.

The basic technique is described by Cook et al. [17, 18] and by Dippé and Wold [25]. Lee et al. [56] give statistical methods which are used to estimate the number of rays required to achieve a given quality. Hsiung et al. [46] describe a system for rendering relativistic effects using a ray-tracer where the speed of each light ray is not infinite. This algorithm can simulate motion-blur by perturbing the time component of spacetime events.

The slowness of ray tracing is partially solved by the Reyes architecture [19], which implements stochastic sampling with a z-buffer and sub-pixel-sized micropolygons. Reyes implements motion blur by jittering the location of the micropolygons according to the required motion.

**Image Accumulation:** Korein and Badler [53] describe a technique which is similar to distributed ray tracing. Multiple images of the entire scene are rendered at time instants distributed within the time spanned by the frame. The resulting series of images are then blended together with some filter function. Haeberli and Akeley [39] describe a hardware implementation of the same algorithm. A hardware z-buffer quickly renders many images of a scene with pixel centers jittered in both space and time. The hardware quickly sums each frame into an *accumulation buffer*, which attenuates each image according to the total number of images which are produced. The image accumulation technique shares most of the advantages of distributed ray tracing, and is much faster.

## 4.3 The Splatting-Based Motion Blur Algorithm

### 4.3.1 Motivation

The motion blur method described in this chapter was inspired by an implementation of the *accumulation buffer* method (Haeberli and Akeley [39]). Figure 4.1 demonstrates this technique. The first row shows a splat sampled at the starting and ending positions of a frame. In the second row the same splat is sampled at three positions, in the third row it is sampled at five positions, and in the fourth row it is sampled at nine positions. The energy of the sampled splats is attenuated according to the total number of samples taken, and the samples are all summed together in the accumulation buffer. Carrying this process to the limit of an infinite number of samples is equivalent to integrating the splat across its motion path during the time frame. This is shown in the last row of Figure 4.1.

While the accumulation buffer technique is general in the sense that it can calculate the correct motion blur for any geometric object, with splatting every object has the same simple geometric form: the projection of a splat is a circular or ellipsoidal disk. The simplicity of this primitive means it is easy to analytically calculate a splat's motion blur, as shown in the last row of Figure 4.1. This is the intuition behind the proposed motion-blur method.

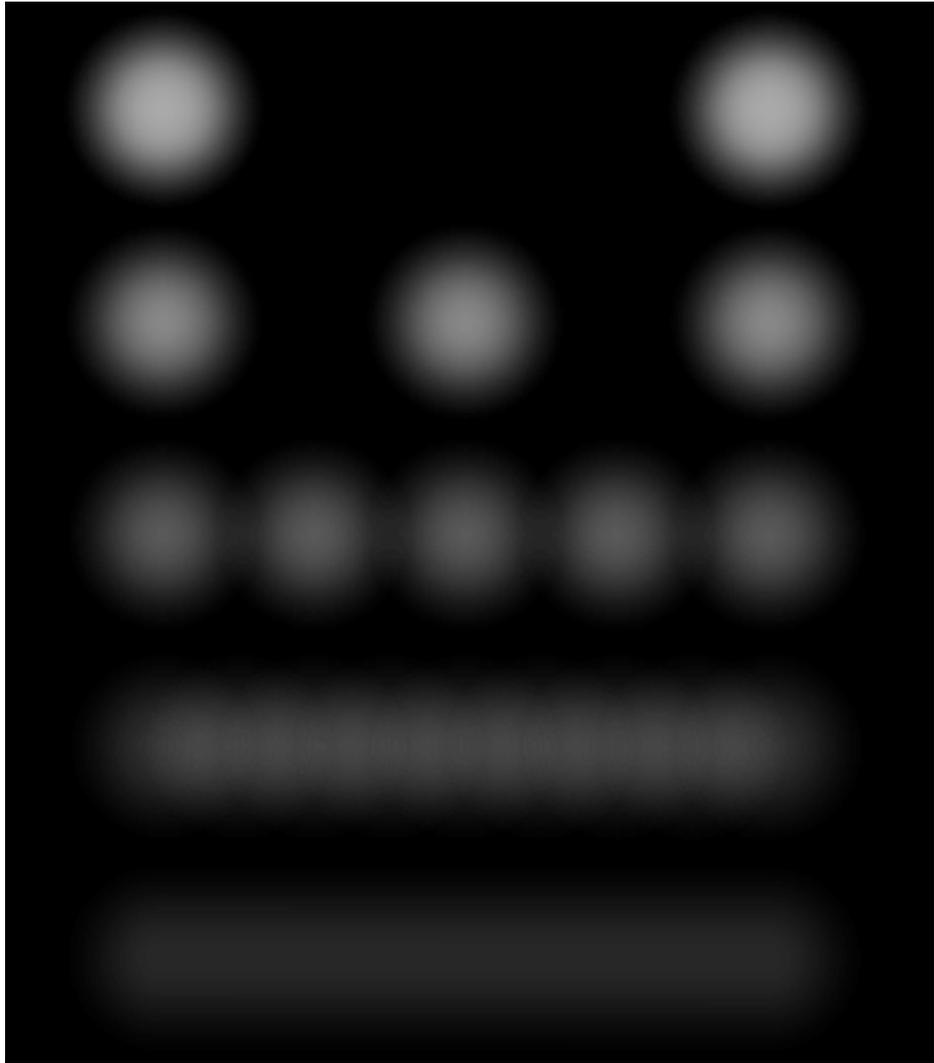


Figure 4.1: Motion blur calculated with the accumulation buffer technique.

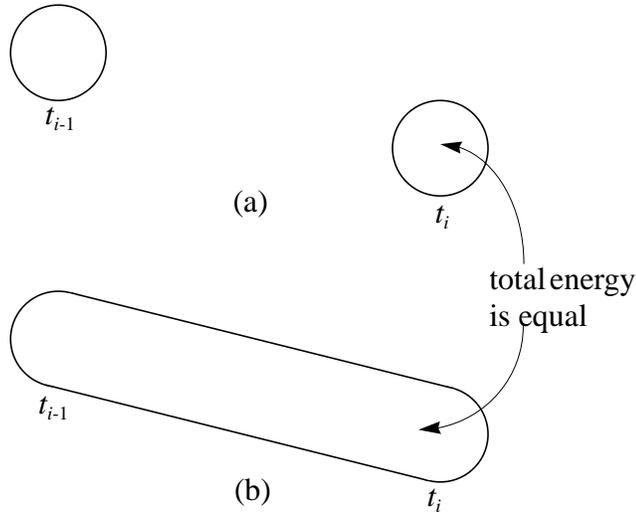


Figure 4.2: Drawing a motion-blurred splat from position  $t_{i-1}$  to  $t_i$ .

### 4.3.2 Method

The basic idea of the algorithm is as follows. Assume that at each time step  $t_0, t_1, t_2, \dots$  each splat has a certain position in eye space. Let the current time step be  $t_i$ , and assume that a particular splat has the positions shown in Figure 4.2a for  $t_i$  and the previous time step  $t_{i-1}$ . These positions are calculated in *projected eye space* — that is, they ignore the eye space  $z$ -coordinate.

Given that the splat is rendered at time  $t_i$ , we want to blur the motion of the splat between its old position at  $t_{i-1}$  and its current position at  $t_i$ . Figure 4.2b shows one way to do this. Here the splat's eye space path between  $t_i$  and  $t_{i-1}$  is connected with a line which has the same width as the splat's diameter. Following an argument analogous to that given in Section 3.3.4, the energy of the motion-blurred splat needs to be the same as the non-blurred splat. Because the area of the motion blurred splat is larger, this means the energy per unit area must be attenuated, and so the motion blurred splat is dimmer than the non-blurred splat.

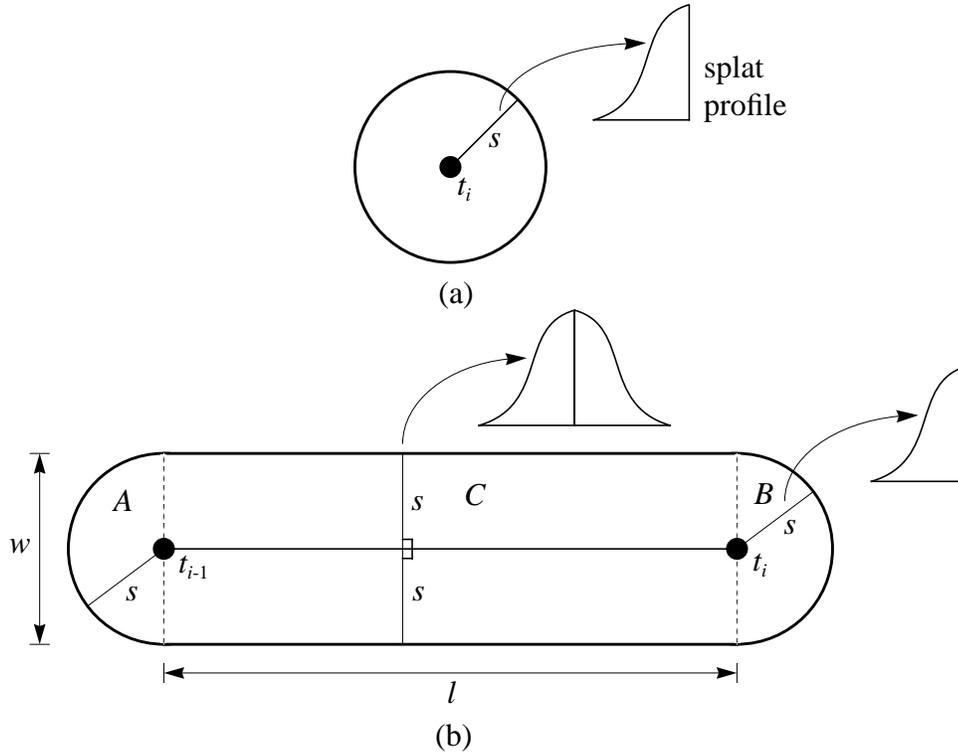


Figure 4.3: The construction of a non-blurred and a motion-blurred splat. (a) A non-blurred splat (splat 1). (b) A motion-blurred splat (splat 2).

Figure 4.3 shows how the motion blurred splats are drawn. Figure 4.3a shows a non-blurred splat (splat 1), while Figure 4.3b shows a motion-blurred splat (splat 2). Splat 1 is drawn at position  $t_i$ . The splat is radially symmetric, so its footprint is a circle and its radial profile  $s$  follows the splat's reconstruction kernel (typically a Gaussian or cubic spline). Splat 2 is drawn between positions  $t_{i-1}$  and  $t_i$ . The original circular footprint is cut in half and “stretched” between  $t_{i-1}$  and  $t_i$ . This results in the two half-circles  $A$  and  $B$ , connected by the rectangle  $C$  with width  $w$  and length  $l$ .  $A$  and  $B$  have the same radial profile  $s$  as splat 1.  $C$  has the profile  $s$  along every line perpendicular to the bisecting line  $\overline{t_{i-1}t_i}$ . Along every line parallel to  $\overline{t_{i-1}t_i}$ ,  $C$  has a constant value.

As discussed in Section 1.2.5, the splat footprint polygons are rotated in eye space so they are perpendicular to the ray from the eye point to the splat's position. Figure 4.4

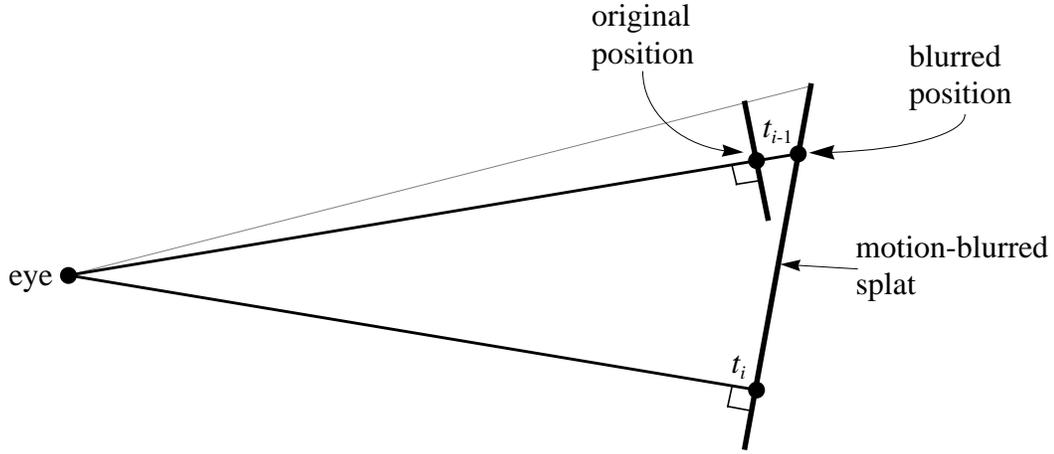


Figure 4.4: The motion-blurred splat is drawn perpendicular to the ray from the eye point to position  $t_i$ .

shows how this works for the motion-blurred splats. The figure shows a “side view” of a motion-blurred splat, looking on the edge of the splat polygon. The motion-blurred splat is perpendicular to the ray passing through the ending position  $t_i$ . Because of this, the motion-blurred splat does not precisely line up with the splat drawn at the beginning position  $t_{i-1}$ , but instead is slightly farther from the eye point and is not perpendicular to the ray passing through  $t_{i-1}$ . This error is small except for splats which undergo large amounts of movement close to the view point.

As discussed above, the motion-blurred splat 2 (Figure 4.3b) contributes the same amount of energy to the image as the stationary splat 1 (Figure 4.3a). This means the total energy of splat 2 must be attenuated according to the ratio of the areas of the splats:

$$E_2 = \frac{A_1}{A_2} E_1, \quad (4.1)$$

where  $A_1, A_2$  are the areas of the splats and  $E_1, E_2$  are some energy measure for the splats. Now  $A_2 = A_1 + lw$ , because splat 2 has the same area as splat 1 plus the area of the rectangle  $C$ . Therefore

$$E_2 = \frac{A_1}{A_1 + lw} E_1. \quad (4.2)$$

This equation is used to attenuate the energy of all motion-blurred splats.

Examples of energy measures include the volume under the splat kernel or the alpha channel of the polygon defining the 2D splat footprint. In the implementation described in this chapter the alpha channel is used as the measure of splat energy.

As discussed above, splats are motion blurred between their previous position  $t_{i-1}$  and their current position  $t_i$ . At time interval  $t_i$  the projected distance the splat has travelled between  $t_{i-1}$  and  $t_i$  is measured. If this distance is greater than some interval  $\delta$ , then the splat is motion blurred between  $t_{i-1}$  and  $t_i$ . The distance  $\delta$  is chosen to be small enough so that if the splat is not moved between  $t_{i-1}$  and  $t_i$ , or if the motion is completely towards or away from the eye point (e.g. all of the motion is contained in the eye space  $z$ -coordinate), then the motion blur is not calculated.

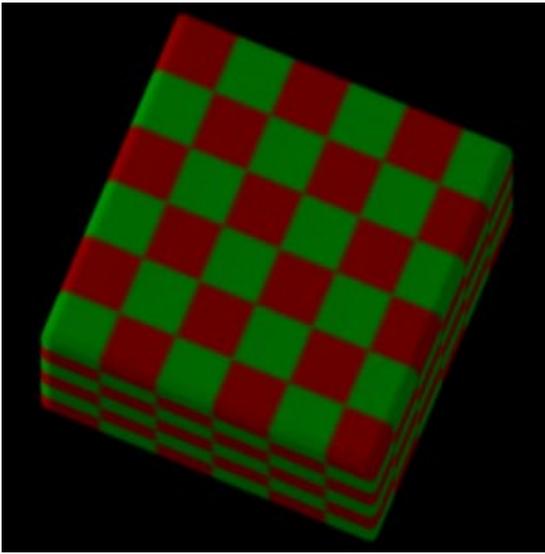
## 4.4 Results and Discussion

Figure 4.5 gives some results of the new motion-blur method. The figure shows one frame from an animation of a rotating  $60 \times 60 \times 60$  hollow cube which has been texture-mapped with alternating red and green  $10 \times 10 \times 10$  sub-cubes. The animation is generated with an orthographic projection. Figure 4.5a shows the frame with no motion blur. Figure 4.5b shows the frame with the new motion-blurred splats, while Figure 4.5c shows the new motion-blurred splats composited with a *sheet summation buffer* (see Section 1.2.2.2 on page 15). Figure 4.5d shows the frame motion-blurred using the *accumulation buffer* technique described above, with the box's motion sampled at five sub-positions.

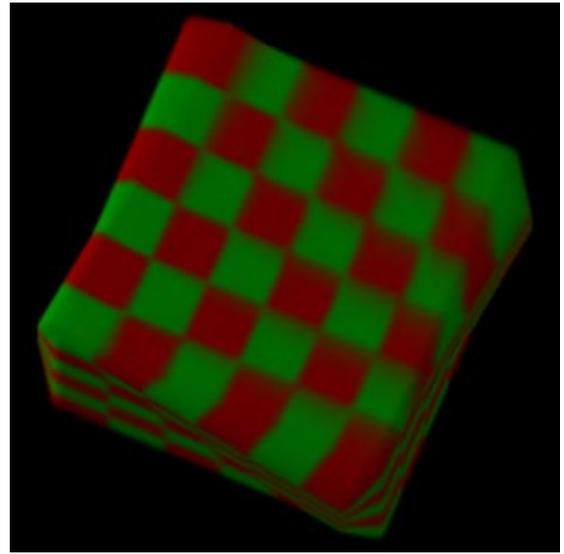
As discussed in Section 4.2 above, to date the most generally successful motion blur technique is *distributed super-sampling*. The accumulation buffer method is a good approximation to distributed super-sampling, so Figure 4.5d is a good benchmark image for motion blur.

The primary advantage of the new technique is speed. While generating Figure 4.5b required rendering the dataset five times, Figures 4.5b and 4.5c only required rendering the dataset once.

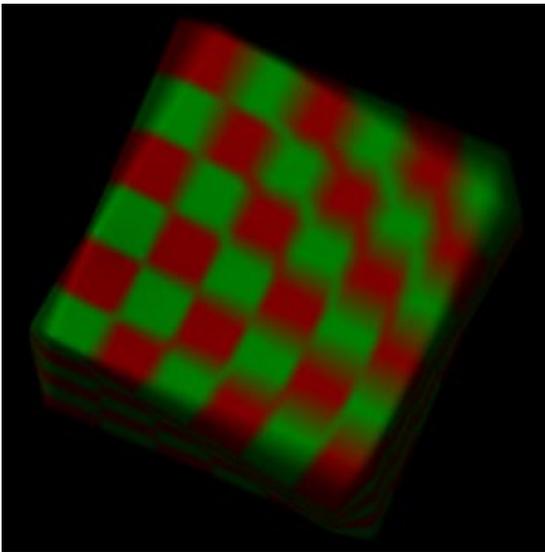
Figure 4.5b compares well to Figure 4.5d except along the edges, where a stretching artifact is apparent. This artifact illustrates the biggest limitation of the new technique: it



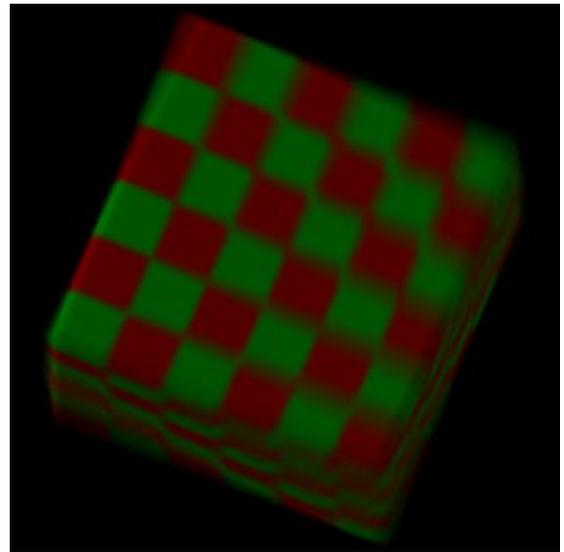
(a)



(b)



(c)



(d)

Figure 4.5: One frame from an animation of a rotating box demonstrating the new motion blur method. (a) No motion blur. (b) The new motion blur method with composited splats. (c) The new motion blur method with a sheet summation buffer. (d) Motion blur from the accumulation buffer method.

accentuates the *splat overlap problem*. As discussed in Section 1.2.2.1, the splat overlap problem arises because where splats overlap the last splat drawn hides the previously drawn splats. Normally this produces fairly small artifacts, as demonstrated in Figure 1.13 on page 20. But because the new technique draws bigger splats with more overlap, the artifacts become more apparent. This is why the edges of Figure 4.5b appear stretched: the splats along the edge are overwriting the splats just inside the edge.

One impact of this limitation is that the new motion-blur technique interacts with the Perspective Back-to-Front (PBTF) visibility ordering given in Chapter 2. The motion-blur technique magnifies the scanline artifacts shown in Figures 1.13a and 1.13d, which appear to march along the cube face because the PBTF ordering changes with every frame. Thus Figure 4.5 is rendered using the Westover Back-to-Front (WBTF) visibility ordering. Although this eliminates the scanline artifacts, the WBTF ordering itself suffers from the *splat ordering problem* described in Section 1.2.2.1 on page 12. Thus using the WBTF instead of the PBTF visibility ordering trades off an artifact that appears on every frame for a larger artifact that only appears on some frames.

A solution to this limitation is to use the new motion-blur technique with a *summed sheet buffer*. This produces better results: in Figure 4.5c the splats along the edge blend with the splats just inside the edge, and the stretching artifact is less apparent. Figure 4.5c compares favorably to Figure 4.5d. This is not a perfect solution, however, since a summed sheet buffer has other limitations which are discussed in Section 1.2.2.3.

## 4.5 Summary and Future Work

This chapter has presented a new motion-blur technique for discrete object-order rendering algorithms, and has demonstrated the technique with the splatting algorithm. The technique computes the projected path of each splat during each frame, and draws a splat elongated in the direction of motion.

The technique suggests new efforts in several areas:

- As given the technique approximates each splat's path with a linear function. It would be interesting to explore modeling a closer approximation to the splat's true path, perhaps fitting it with a parabolic or cubic spline.
- The technique draws each splat with uniform brightness and opacity. Higher-quality motion blur might result from non-uniform splat attributes, especially along the length dimension of the splat. For example, greater brightness in the middle of the splat tapering off towards each end might give a closer approximation to the accumulation buffer results, where the summed sub-samples tend to be brightest in the middle of the motion path.
- Finally, it would be interesting to explore different shapes for the splats. Teardrop-shaped splats might be useful for still images which give a clear, frozen image the impression of movement.

## CHAPTER 5

### APPLICATIONS TO TERRAIN RENDERING

#### 5.1 Introduction

Rendering terrain data is an important application area in computer graphics. Much early work in computer graphics was motivated by building flight simulators. This is a very taxing application, which requires real-time rendering of potentially very large databases. In addition, terrain rendering is important for geographic information systems and architectural landscape previewing systems.

One common source for terrain data are fractal algorithms [31, 62], which can procedurally generate terrain and other natural scenes. Another common source is actual earth terrain data obtained from aerial or satellite imagery [13]. The terrain data typically comes in the form of two datasets: a color or texture image (typically from an aerial or satellite photograph) and topology or elevation samples. The color dataset is usually at a much higher resolution than the elevation dataset [2]. These two data sources are registered and merged [13] to form two arrays: a 2D array of height values, and a 2D array of color values. Typically the color values have three channels corresponding to red, green, and blue; but other color channels (such as infrared, ultraviolet, etc.) are possible. These two arrays are the typical input to a terrain renderer.

Traditionally, terrain data has been modeled as a triangular mesh. This chapter explores the idea of modeling the terrain as a discrete object in a volume raster. Section 5.2 discusses previous techniques for rendering terrain datasets. Section 5.3 gives a series of images which demonstrate all the techniques presented in this dissertation.

## 5.2 Previous Work

Like the volume rendering algorithms reviewed in Section 1.1.2, terrain rendering algorithms can be classified according to how the terrain is rendered. This classification groups the algorithms into *ray casting*, *shear-warp*, and *object order* methods.

### 5.2.1 Ray Casting

The ray-casting methods typically cast a ray from each pixel and sample the terrain at the resulting intersection point. Unlike volume ray casting, which samples each ray many times, terrain ray casting algorithms usually only sample one point per ray. These algorithms typically store the height and color information in grid form; the grids are interpolated at the intersection point. They can be further characterized by how they represent rays and step through the data structure:

**Incremental Techniques:** Dungan [27], Coquillart and Gangnet [20], and Musgrave [73] all trace the 2D projection of the ray across the baseplane of the terrain grid. At each step, the height of the ray is compared to the height of the terrain data at that step. When the ray height drops below the terrain, the exact intersection point is found.

**Space-Leaping Techniques:** Cohen-Or et al. [14] and Lee and Shin [57] use a similar incremental technique, but accelerate the process by starting the traversal of each ray above the intersection point of the previous ray.

**Hierarchical Techniques:** Cohen and Shaked [16] store the height grid at multiple resolutions in a quadtree data structure. The ray steps first across the largest quadtree node, which contains the highest point of the terrain. If the ray height is below this, then the ray is recursively compared to the heights of the proper quadrants. This continues until the ray intersects the terrain data at a quadtree leaf node.

**Distance Transform Techniques:** Paglieroni and Petersen [77, 78] compute a distance transform of the height grid. This gives an area of empty space around every voxel, where

it is guaranteed that the ray will not encounter other voxels. The ray is stepped from a given voxel to the edge of the voxel's distance transform.

Because ray casting point-samples the terrain grid, all of these methods are particularly subject to aliasing artifacts. Many of the above methods perform some anti-aliasing by storing the terrain grid at multiple levels of resolution; the lower resolution grids are used for pixels which show areas of the terrain that are far from the view point. This still results in aliasing artifacts, however. Cohen-Or [15] describes a ray-casting terrain rendering technique that area-samples the terrain grid, resulting in high-quality terrain images.

### **5.2.2 Shear-Warp**

The shear-warp methods create a perspective projection of the height and color grids through a series of 1D shearing and warping operations. The data is resampled into a regular grid which is designed so that all voxels which might occlude a particular voxel are located in the same row or column. Then a back-to-front traversal along these rows or columns results in the correct visibility. This is followed by an inverse resampling that restores the data to a perspective projection. The basic implementation of this technique is described by Robertson [85]. Another implementation using a spherical projection is given by Miller [70]. The technique's advantages are that each 1D pass can be made quickly, and that the technique is easy to parallelize. Vezina and Robertson [95], Kaba et al. [48], and Kaba and Peters [49] all describe parallel implementations, some of which can render 30 frames per second [48, 49]. Robertson [86] also describes how the technique can be extended to rapidly generate shadows on a terrain surface.

### **5.2.3 Object Order**

Most of the object-order techniques fit polygons or patches to the terrain height field, and then render these using standard polygon or patch rendering techniques. The terrain color grid is texture mapped onto the polygons or patches. An example is Coquillart and Gangnet [20], which fits the surface with bilinear patches. Another is Kaneda et al. [52], which fits polygons that vary in size in such a way that they sample approximately equal areas of the terrain when rendered from a perspective projection. This is similar to Geymayer et al.

[34], except that the polygons are pre-fit into a pyramid data structure, and thus the terrain does not have to be resampled with each frame. Kaneda et al. [52] also demonstrates an advantage of polygon/patch fitting methods: it is easy to include additional polygons representing buildings or other ground structures.

Agranov and Gotsman [2] describe a hybrid order algorithm. They use ray-casting to determine the pixels along the screen border; this projects the screen as a polygon onto the terrain dataset. They then project all the triangles contained in this polygon onto the screen using a z-buffer.

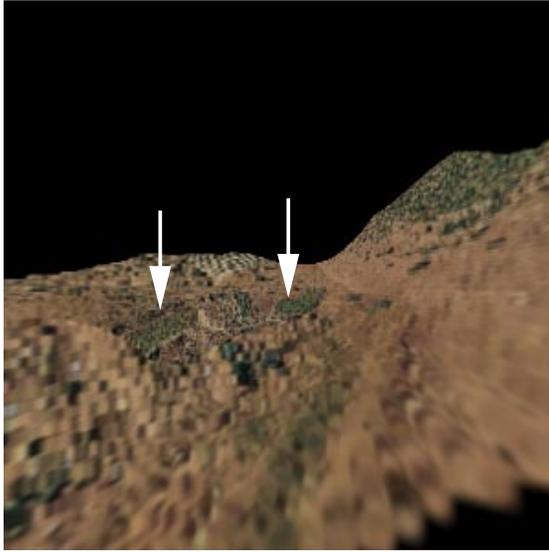
Wright and Hsieh [105] model the terrain as a set of voxel grids at different resolutions. The voxels are visited and projected onto the image plane in order of increasing distance from the image plane; as the distance to the image plane increases voxels from lower resolutions are used. This ensures that the projected voxels span approximately the same area of the image plane.

### 5.3 Terrain Rendering Examples

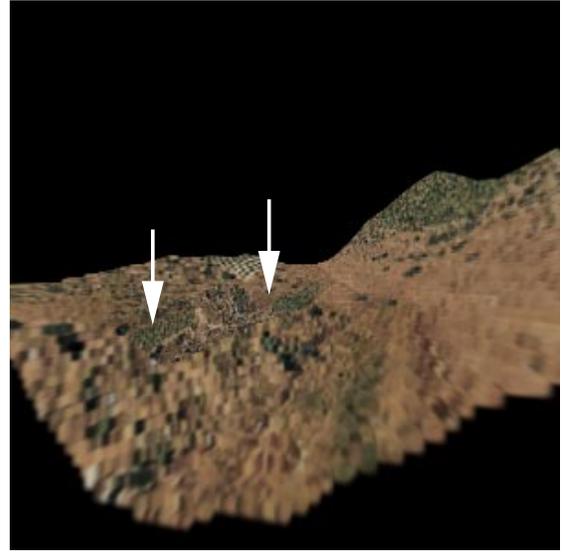
This section gives examples of the techniques given in this dissertation applied to rendering terrain datasets.

Figure 5.1 shows a series of four frames from an animation of a  $256 \times 256$  terrain dataset containing 133K splats. The frames are rendered with a perspective projection, using the Back-to-Front visibility ordering described in Chapter 2. Figures 5.1a and 5.1b show a visibility artifact, denoted by arrows, where portions of the dataset behind the left-hand hill incorrectly show through the hill. Figure 5.1 also demonstrates aliasing, which is particularly apparent on the hill which rotates towards the top of the dataset in Figures 5.1c and 5.1d.

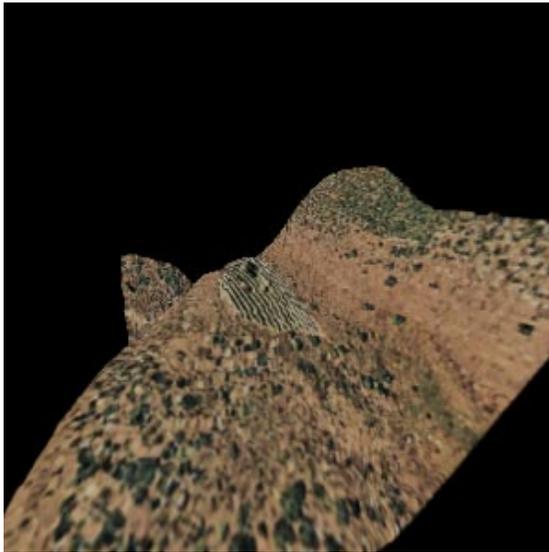
Figure 5.2 shows the same frames as Figure 5.1, except that now the dataset is rendered with the Perspective Back-to-Front (PBTF) visibility ordering given in Chapter 2. Note that the visibility artifact from Figures 5.1a and 5.1b is gone.



(a)



(b)

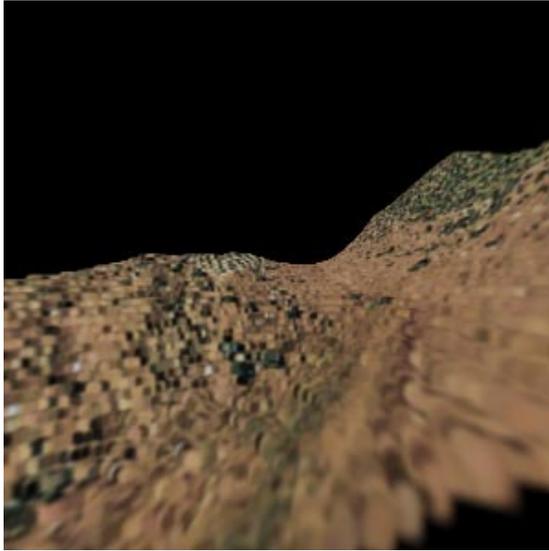


(c)

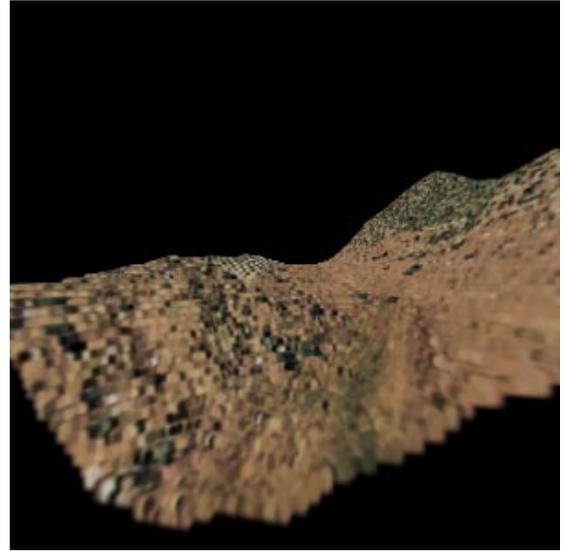


(d)

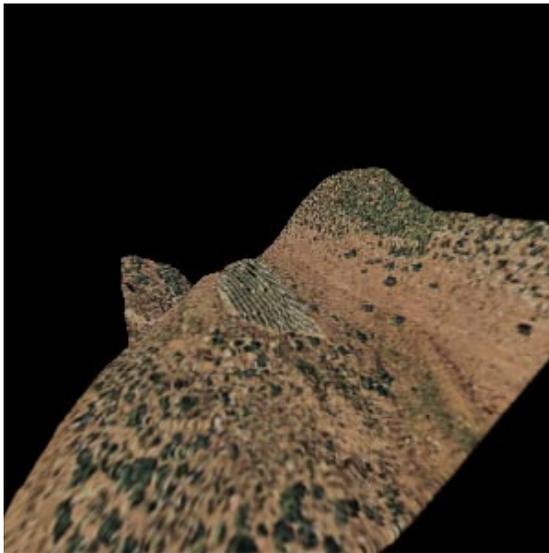
Figure 5.1: Four frames from a terrain animation with visibility problems denoted by arrows.



(a)



(b)



(c)



(d)

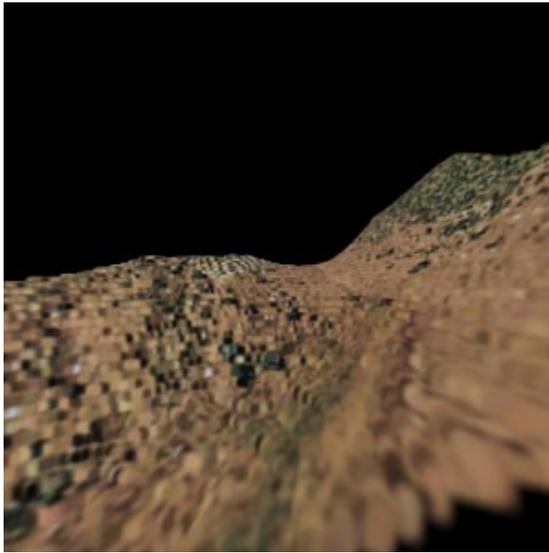
Figure 5.2: The same animation as Figure 5.1, using the Perspective Back-to-Front visibility ordering from Chapter 2.

Figure 5.3 shows the same frames as Figure 5.1, except that now the dataset is rendered with both the PBTF visibility ordering and the anti-aliasing technique from Chapter 3. The anti-aliasing is particularly noticeable on the upper hill in Figures 5.3c and 5.3d.

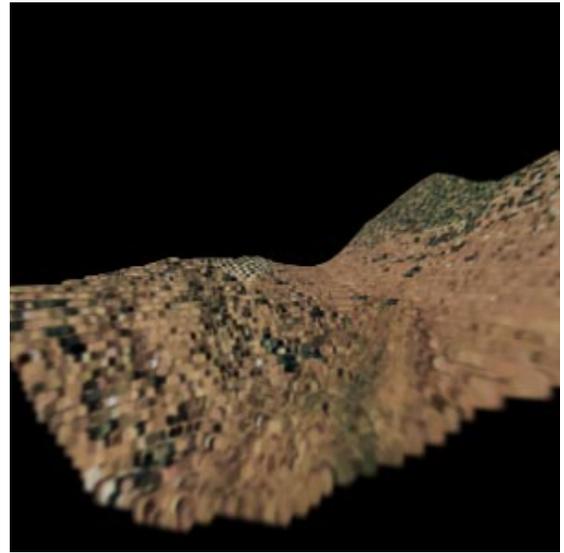
Figure 5.4 shows four frames from another animation of the same dataset, this time rendered with an orthographic projection and utilizing the Westover Back-to-Front (WBTF) visibility ordering described in Chapter 2. Figure 5.5 shows the same frames rendered using the motion-blur technique of Chapter 4, while Figure 5.6 shows the frames motion-blurred using the *accumulation buffer* technique (also described in Chapter 4), where the terrain's motion is sampled at 5 sub-positions.

As discussed in Section 4.4, these examples utilize the WBTF instead of the PBTF visibility ordering to avoid the scanline overlapping splat artifact described in Section 1.2.2.1 on page 12. As Chapter 2 discusses, the WBTF does not give the correct visibility for *every* perspective viewpoint (although it does give the correct visibility for *many* perspective viewpoints). Therefore the examples given in Figures 5.4, 5.5, and 5.6 use an orthographic projection.

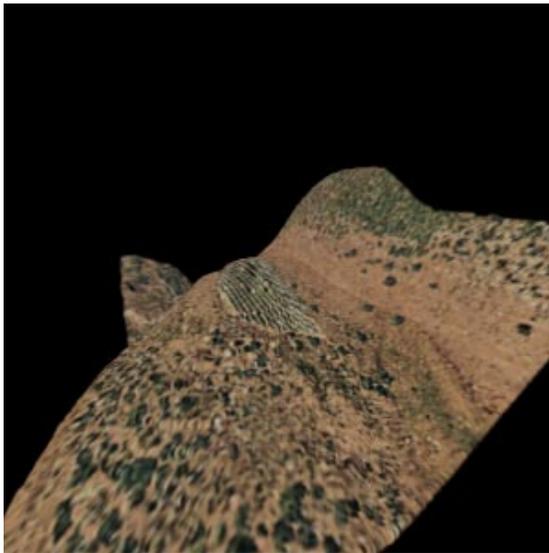
As discussed in Chapter 4, the accumulation buffer technique generates accurate motion blur, so it is instructive to compare Figure 5.5 with Figure 5.6. The frames in the figures compare quite favorably. The primary advantage of the motion-blurred frames is rendering speed: each image in Figure 5.5 required rendering only one frame, while each image in Figure 5.6 required rendering five frames. The disadvantages are stretched splats along the bottom of the dataset in Figures 5.5a and 5.5d (caused by the *overlapping splat problem* and discussed in more detail in Section 4.4), and the squared-off corners in Figure 5.5b, caused by approximating each splat's motion vector (which is an arc for the circular motion shown in Figure 5.5b) with a line segment.



(a)



(b)

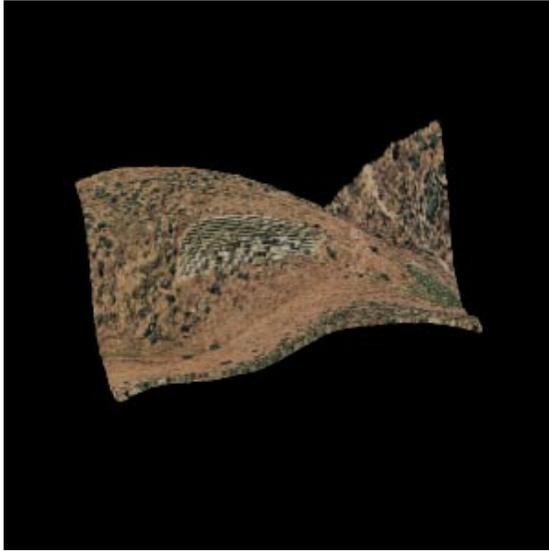


(c)

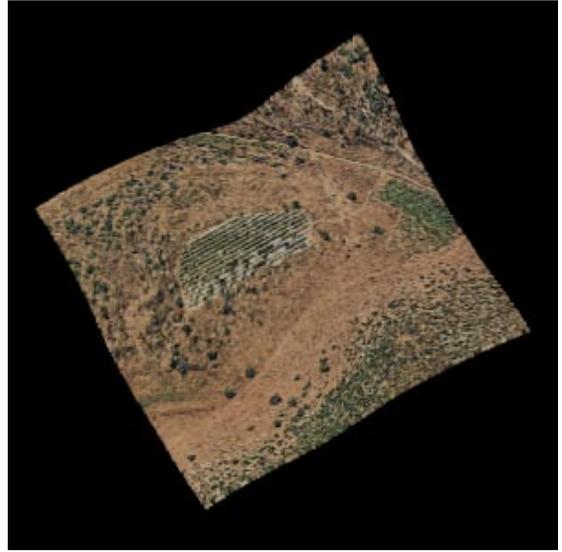


(d)

Figure 5.3: The same animation as Figure 5.1, using the Perspective Back-to-Front visibility ordering from Chapter 2 and the anti-aliasing technique from Chapter 3.



(a)



(b)

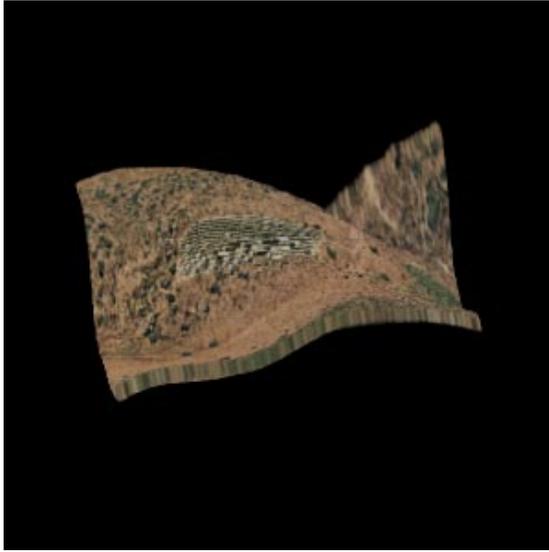


(c)

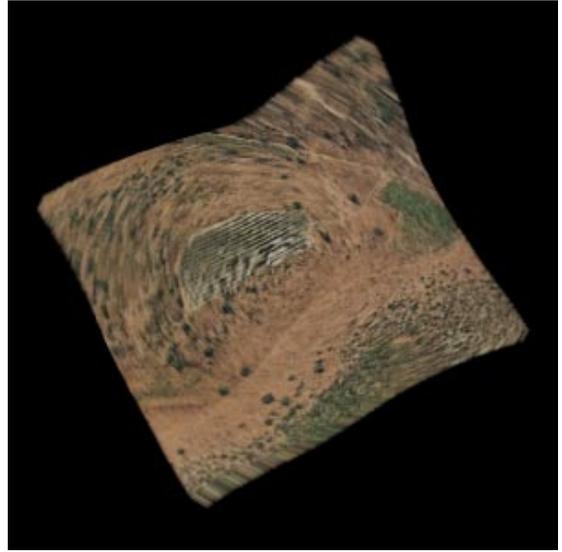


(d)

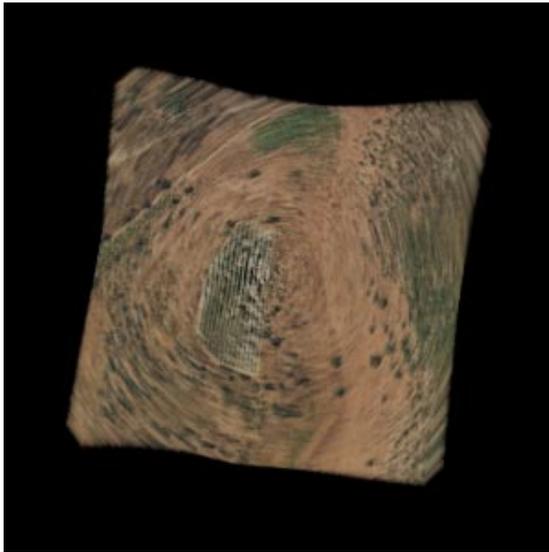
Figure 5.4: Four frames from a terrain animation rendered with an orthographic projection.



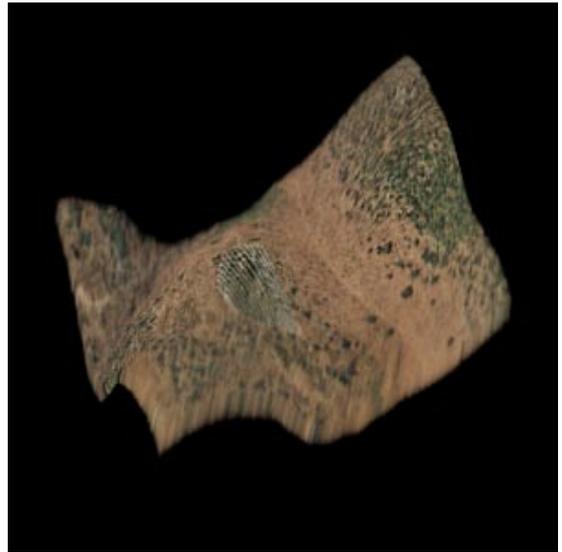
(a)



(b)

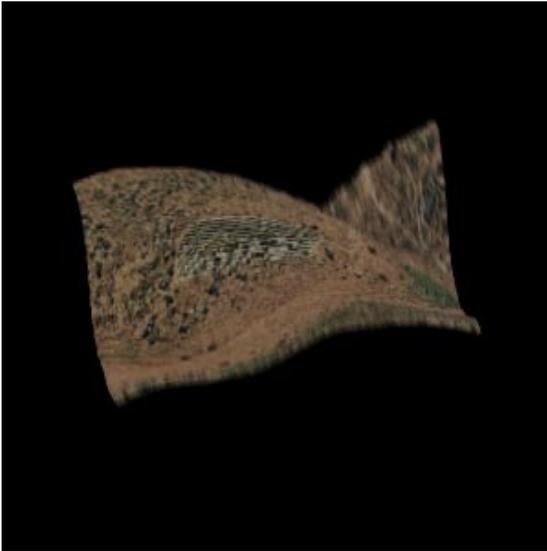


(c)

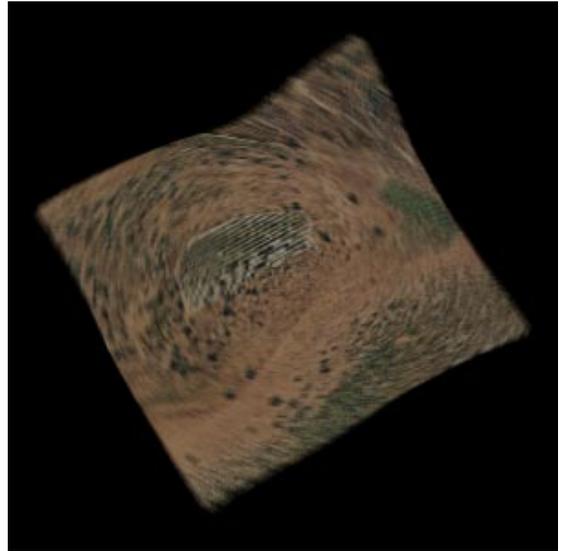


(d)

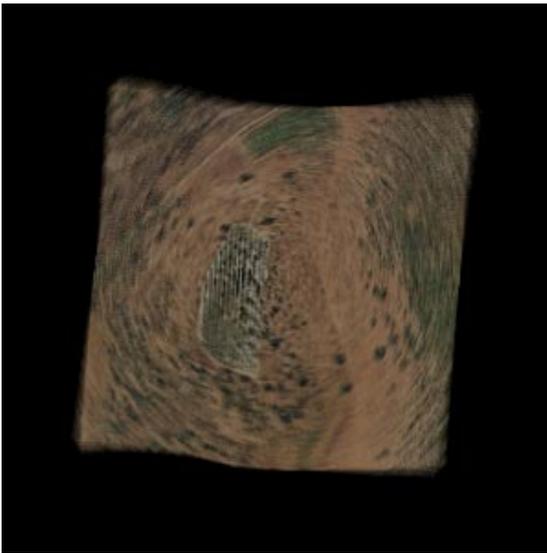
Figure 5.5: The same animation as Figure 5.4, using the motion-blur technique of Chapter 4.



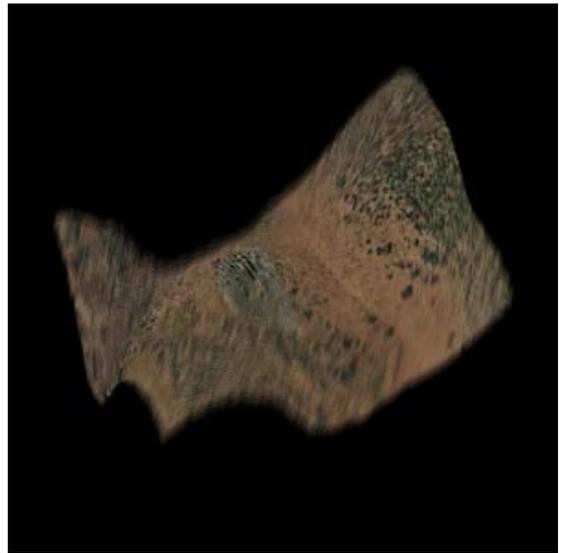
(a)



(b)



(c)



(d)

Figure 5.6: The same animation as Figure 5.4, using the *accumulation buffer* motion blur technique.

## CHAPTER 6

### CONTRIBUTIONS AND CONCLUSIONS

This dissertation contributes three new techniques to the science of rendering discrete objects and volumes. Although these techniques have been presented in the context of the splatting algorithm, each technique has additional applications that go beyond splatting.

**Perspective Back-to-Front Visibility Ordering:** Chapter 2 presents the Perspective Back-to-Front (PBTF) visibility ordering. This is an ordering of a rectilinear grid of objects which visits the objects in strict visibility order with respect to a viewpoint; it works for both orthographic and perspective projections. The location of the viewpoint is unrestricted — it may be located anywhere in space relative to the grid or inside the grid. Chapter 2 presents a proof of correctness for the technique, based on the idea of cutting planes. Chapter 2 also demonstrates that two commonly used visibility orderings for rectilinear grids — the Back-to-Front (BTF) and the Westover Back-to-Front (WBTF) — do not always give a correct visibility ordering for a perspective projection.

**Object-Order Anti-Aliasing Technique:** Chapter 3 presents a new anti-aliasing technique which can be applied when one grid is resampled and projected onto another grid. The method is different from existing techniques in that the reconstruction kernels have support in the source grid space, as opposed to the more common technique of support in the destination grid space. The technique is applied to the splatting algorithm, and as such it represents the first anti-aliasing technique for splatting. Furthermore, Chapter 3 contains a formal derivation which demonstrates that the technique provides enough low-pass filtering so that *no* aliasing is introduced during the reconstruction process.

**Splatting-Based Motion-Blur:** Chapter 4 presents a new technique for adding motion blur to rendered images of discrete datasets. The technique involves integrating the projection of each discrete data element across the view plane. The technique is given in the context of splatting. Because splats have a simple projection (a circle or ellipse), it is easy to analytically calculate the integrated path. The technique is much more efficient than existing motion-blur techniques such as supersampling.

**Applications for Terrain Rendering:** Chapter 5 applies these techniques to the application area of motion blur. Unlike traditional methods, which model the terrain with triangular surface patches, it represents the terrain by a collection of points. This point-based representation, in conjunction with the new rendering techniques, provides an innovative, efficient, and accurate solution to the terrain rendering problem.

## BIBLIOGRAPHY

- [1] Abram, G., Westover, L., and Whitted, T., “Efficient Alias-free Rendering using Bit-masks and Look-up Tables”, *Computer Graphics* (proceedings of SIGGRAPH), 19(3), July 1985, pp. 53–59.
- [2] Agranov, G., and Gotsman, C., “Algorithms for Rendering Realistic Terrain Image Sequences and their Parallel Implementation”, *The Visual Computer*, 11, November 1995, pp. 455–464.
- [3] Amanatides, J., “Ray Tracing with Cones”, *Computer Graphics* (proceedings of SIGGRAPH), 18(3), July 1984, pp. 129–135.
- [4] Anderson, D., “Hidden Line Elimination in Projected Grid Surfaces”, *ACM Transactions on Graphics*, 1(4), October 1982, pp. 274–288.
- [5] Aref, W. G., and Samet, H., “An Algorithm for Perspective Viewing of Objects Represented by Octrees”, *Computer Graphics Forum*, 14(1), 1995, pp. 59–66.
- [6] Bracewell, R. N., *The Fourier Transform and Its Applications* (2nd edition), McGraw-Hill, 1978.
- [7] Cabral, B., Cam, N., and Foran, J., “Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware”, In proceedings of *1994 Symposium on Volume Visualization* (Washington, D. C., October 17–18), IEEE Computer Society Press, 1994, pp. 91–98.
- [8] Carpenter, L., “The A-Buffer, An Antialised Hidden Surface Method”, *Computer Graphics* (proceedings of SIGGRAPH), 18(3), July 1984, pp. 103–108.
- [9] Catmull, E., “A Hidden-Surface Algorithm with Anti-Aliasing”, *Computer Graphics* (proceedings of SIGGRAPH), 12(3), August 1978, pp. 6–11.
- [10] Catmull, E., “An Analytic Visible Surface Algorithm for Independent Pixel Processing”, *Computer Graphics* (proceedings of SIGGRAPH), 18(3), July 1984, pp. 109–115.

- [11] Chen, L., Herman, G. T., Reynolds, R.A. and Udupa, J.K., “Surface Shading in the Cuberille Environment”, *IEEE Computer Graphics and Applications*, 5(12), December 1985, pp. 33–43.
- [12] Cline, H. E., Lorensen, W. E., Ludke, S., Crawford, C. R., and Teeter, B. C., “Two Algorithms for the Three-Dimensional Reconstruction of Tomograms”, *Medical Physics*, 15(3), May/June 1988, pp. 320–327.
- [13] Cohen, D., and Gotsman, C., “Photorealistic Terrain Imaging and Flight Simulation”, *IEEE Computer Graphics and Applications*, 14(2), March 1994, pp. 10–12.
- [14] Cohen-Or, D., Rich, E., Lerner, U., and Shenkar, V., “A Real-Time Photo-Realistic Visual Flythrough”, *IEEE Transactions on Visualization and Computer Graphics*, 2(3), September 1996, pp. 255–265.
- [15] Cohen-Or, D., “Exact Antialiasing of Textured Terrain Models”, To be published in *The Visual Computer*, 1997.
- [16] Cohen, D. and Shaked, A., “Photo-Realistic Imaging of Digital Terrains”, In proceedings of *Eurographics '93*, 1993, pp. 363–373.
- [17] Cook, R. L., Porter, T. and Carpenter, L., “Distributed Ray Tracing”, *Computer Graphics* (proceedings of SIGGRAPH), 18(3), July 1984, pp. 137–145.
- [18] Cook, R. L., “Stochastic Sampling in Computer Graphics”, *ACM Transactions on Graphics*, 5(1), January 1986, pp. 51–72.
- [19] Cook, R. L., Carpenter, L., and Catmull, E., “The Reyes Image Rendering Architecture”, *Computer Graphics* (proceedings of SIGGRAPH), 21(4), July 1987, pp. 95–102.
- [20] Coquillart, S. and Gangnet, M., “Shaded Display of Digital Maps”, *IEEE Computer Graphics and Applications*, 4(7), July 1984, pp. 35–42.
- [21] Crawfis, R. A. and Max, N., “Texture Splats for 3D Scalar and Vector Field Visualization”, In proceedings of *Visualization '93* (San Jose, California, October 25–29), IEEE Computer Society Press, 1993, pp. 261–266.
- [22] Crow, F. C., “The Aliasing Problem in Computer-Generated Shaded Images”, *Communications of the ACM*, 20(11), November 1977, pp. 799–805.
- [23] Crow, F. C., “A Comparison of Antialiasing Techniques”, *IEEE Computer Graphics and Applications*, 1(1), January 1981, pp. 40–48.
- [24] Crow, F. C., “Summed-Area Tables for Texture Mapping”, *Computer Graphics* (proceedings of SIGGRAPH), 18(3), July 1984, pp. 207–212.

- [25] Dippé, M. A. Z. and Wold, E. H., “Antialiasing Through Stochastic Sampling”, *Computer Graphics* (proceedings of SIGGRAPH), 19(3), July 1985, pp. 69–78.
- [26] Drebin, R. A., Carpenter, L., and Hanrahan, P., “Volume Rendering”, *Computer Graphics* (proceedings of SIGGRAPH), 22(4), August 1988, pp. 65–74.
- [27] Dungan, W., “A Terrain and Cloud Computer Image Generation Model”, *Computer Graphics* (proceedings of SIGGRAPH), 13(3), August 1979, pp. 143–150.
- [28] Feibush, E. A., Levoy, M., and Cook, R. L., “Synthetic Texturing Using Digital Filters”, *Computer Graphics* (proceedings of SIGGRAPH), 14(3), July 1980, pp. 294–301.
- [29] Fiume, E., Fournier, A., and Rudolph, L., “A Parallel Scan Conversion Algorithm with Anti-Aliasing for a General-Purpose Ultracomputer”, *Computer Graphics* (proceedings of SIGGRAPH), 17(3), July 1983, pp. 141–150.
- [30] Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F., *Computer Graphics: Principles and Practice (2nd edition)*, Addison-Wesley, 1990.
- [31] Fournier, A., Fussell, D., and Carpenter, L., “Computer Rendering of Stochastic Models”, *Communications of the ACM*, 25(6), June 1982, pp. 371–384.
- [32] Frieder, G., Gordon, D., and Reynolds, R. A., “Back-to-Front Display of Voxel-Based Objects”, *IEEE Computer Graphics and Applications*, 5(1), January 1985, pp. 52–60.
- [33] Fuchs, H., Kedem, Z. M., and Uselton, S. P., “Optimal Surface Reconstruction from Planar Contours”, *Communications of the ACM*, 20(10), October 1977, pp. 693–702.
- [34] Geymayer, B., Prantl, M., Muller-Seelich, H., and Tabatabai, B., “Animation of Landscapes Using Satellite Imagery”, In proceedings of *Eurographics '91*, 1991, pp. 437–446.
- [35] Glassner, A., “Adaptive Precision in Texture Mapping”, *Computer Graphics* (proceedings of SIGGRAPH), 20(4), August 1986, pp. 297–306.
- [36] Glassner, A. S., *Principles of Digital Image Synthesis* (in two volumes), Morgan Kaufmann, 1995.
- [37] Grant, C. W., “Integrated Analytic Spatial and Temporal Anti-Aliasing for Polyhedra in 4-Space”, *Computer Graphics* (proceedings of SIGGRAPH), 19(3), July 1985, pp. 79–84.

- [38] Greene, N., and Heckbert, P. S., “Creating Raster Omnimax Images from Multiple Perspective Views Using the Elliptical Weighted Average Filter”, *IEEE Computer Graphics and Animation*, 6(6), June 1986, pp. 21–27.
- [39] Haeberli, P. and Akeley, K., “The Accumulation Buffer: Hardware Support for High-Quality Rendering”, *Computer Graphics* (proceedings of SIGGRAPH), 24(4), August 1990, pp. 309–318.
- [40] Hanrahan, P., “Three-Pass Affine Transforms for Volume Rendering”, *Computer Graphics*, 24(5), November 1990, pp. 71–77.
- [41] Heckbert, P. S., and Hanrahan, P., “Beam Tracing Polygonal Objects”, *Computer Graphics* (proceedings of SIGGRAPH), 18(3), July 1984, pp. 119–127.
- [42] Heckbert, P. S., “Filtering by Repeated Integration”, *Computer Graphics* (proceedings of SIGGRAPH), 20(4), August 1986, pp. 315–321.
- [43] Heckbert, P. S., “Survey of Texture Mapping”, *IEEE Computer Graphics and Applications*, 6(11), November 1986, pp. 56–67.
- [44] Heckbert, P. S., *Fundamentals of Texture Mapping and Image Warping*, Masters Thesis, Department of Electrical Engineering and Computer Science, The University of California at Berkeley, Technical Report Number UCB/CSD 89/516, June 1989.
- [45] Herman, G. T. and Liu, H. K., “Three-Dimensional Display of Human Organs from Computed Tomograms”, *Computer Graphics and Image Processing*, 9(1), January 1979, pp. 1–21.
- [46] Hsiung, P., and Thibadeau, R. H., and Wu, M., “T-Buffer: Fast Visualization of Relativistic Effects in Spacetime”, *Computer Graphics* (1990 Symposium on Interactive 3D Graphics), 24(2), March 1990, pp. 83–88.
- [47] Joy, K. I., Grant, C. W., Max, N. L., and Hatfield, L., *Tutorial: Computer Graphics: Image Synthesis*, Computer Society Press, Washington, D.C., 1988.
- [48] Kaba, J., Matey, J., Stoll, G., Taylor, H., and Hanrahan, P., “Interactive Terrain Rendering and Volume Visualization on the Princeton Engine”, In proceedings of *Visualization '92* (October 19–23, Boston, MA), IEEE Computer Society Press, 1992, pp. 349–355.
- [49] Kaba, J., and Peters, J., “A Pyramid-based Approach to Interactive Terrain Visualization”, In proceedings of *1993 Parallel Rendering Symposium* (San Jose, California, October 25–26), IEEE Computer Society Press, 1993, pp. 67–70.

- [50] Kajiya, J. T. and Von Herzen, B. P., “Ray Tracing Volume Densities”, *Computer Graphics* (proceedings of SIGGRAPH), 18(3), July 1984, pp. 165–174.
- [51] Kajiya, J. T., “The Rendering Equation”, *Computer Graphics* (proceedings of SIGGRAPH), 20(4), August 1986, pp. 143–150.
- [52] Kaneda, K., Kato, F., Nakamae, E., Nishita, T., Tanaka, H., and Noguchi, T., “Three Dimensional Terrain Modeling and Display for Environmental Assessment”, *Computer Graphics* (proceedings of SIGGRAPH), 23(3), July 1989, pp. 207–214.
- [53] Korein, J. and Badler, N., “Temporal Anti-Aliasing in Computer Generated Animation”, *Computer Graphics* (proceedings of SIGGRAPH), 17(3), July 1983, pp. 377–388.
- [54] Lacroute, P. and Levoy, M., “Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation”, *Computer Graphics* (proceedings of SIGGRAPH), Annual Conference Series, 1994, pp. 451–458.
- [55] Laur, D. and Hanrahan, P., “Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering”, *Computer Graphics* (proceedings of SIGGRAPH), 25(4), July 1991, pp. 285–288.
- [56] Lee, M. E., Redner, R. A., and Uselton, S. P., “Statistically Optimized Sampling for Distributed Ray Tracing”, *Computer Graphics* (proceedings of SIGGRAPH), 19(3), July 1985, pp. 61–67.
- [57] Lee, C., and Shin, Y. G., “An Efficient Ray Tracing Method for Terrain Rendering”, In proceedings of *Pacific Graphics '95*, 1995, pp. 180–193.
- [58] Levoy, M., “Display of Surfaces from Volume Data”, *IEEE Computer Graphics & Applications*, 8(3), May 1988, pp. 29–37.
- [59] Levoy, M. and Whitted, T., “The Use of Points as a Display Primitive”, *Technical Report TR 85-022*, The University of North Carolina at Chapel Hill, Department of Computer Science, 1985.
- [60] Lorensen, W. E. and Cline, H. E., “Marching Cubes: A High-Resolution 3D Surface Construction Algorithm”, *Computer Graphics* (proceedings of SIGGRAPH), 21(4), July 1987, pp. 44–50.
- [61] Machiraju, R. and Yagel, R., “Efficient Feed-Forward Volume Rendering Techniques for Vector and Parallel Processors,” In proceedings of *Supercomputing '93* (November 15–19, Portland, Oregon), IEEE Computer Society Press, 1993, pp. 699–708.

- [62] Mandelbrot, B. B., *The Fractal Geometry of Nature*, Freeman Press, New York, 1983.
- [63] Mao, X., “Splating of Non Rectilinear Volumes Through Stochastic Resampling”, *IEEE Transactions on Visualization and Computer Graphics*, 2(2), June 1996, pp. 156–170.
- [64] Marschner, S. R. and Lobb, R. J., “An Evaluation of Reconstruction Filters for Volume Rendering”, In proceedings of *Visualization '94* (Washington, D.C., October 17–21), IEEE Computer Society Press, 1994, pp. 100–107.
- [65] Max, N., Hanrahan, P., and Crawfis, R., “Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions”, *Computer Graphics*, 24(5), November 1990, pp. 27–33.
- [66] Max, N., “Polygon-Based Post-Process Motion Blur”, *The Visual Computer*, 6, December 1990, pp. 308–314.
- [67] Max, N., “Cone-Spheres”, *Computer Graphics* (proceedings of SIGGRAPH), 24(4), August 1990, pp. 59–62.
- [68] Max, N. L., “Sorting for Polyhedron Compositing”, In H. Hagen, H. Müller, G.M., Nielson (Eds.) *Focus on Scientific Visualization*, Springer-Verlag, 1993, pp. 259–268.
- [69] Max, N. L., and Lerner, D. M., “A two-and-a-half-D Motion-Blur Algorithm”, *Computer Graphics* (proceedings of SIGGRAPH), 19(3), July 1985, pp. 85–93.
- [70] Miller, G. S. P., “The Definition and Rendering of Terrain Maps”, *Computer Graphics* (proceedings of SIGGRAPH), 20(4), August 1986, pp. 39–48.
- [71] Möller, T., Machiraju, R., Mueller, K., and Yagel, R., “Classification and Local Error Estimation of Interpolation and Derivative Filters for Volume Rendering”, In proceedings of the *1996 Symposium on Volume Visualization* (San Francisco, California, October 28–29), IEEE Computer Society Press, 1996, pp. 71–78.
- [72] Mueller, K. and Yagel, R., “Fast Perspective Volume Rendering with Splating by Utilizing a Ray-Driven Approach”, In proceedings of *Visualization '96* (San Francisco, California, October 27 – November 1), IEEE Computer Society Press, 1996, pp. 65–72.
- [73] Musgrave, F. K., *Grid Tracing: Fast Ray Tracing for Height Fields*, Technical Report YALEU/DCS/RR-639, Yale University Department of Computer Science Research, 1988.

- [74] Newman, W. M., and Sproull, R. F., *Principles of Interactive Computer Graphics (2nd edition)*, McGraw-Hill, 1979.
- [75] Norton, A., Rockwood, A. P., and Skolmoski, P. T., “Clamping: A Method of Antialiasing Textured Surfaces by Bandwidth Limiting in Object Space”, *Computer Graphics* (proceedings of SIGGRAPH), 16(3), July 1982, pp. 1–8.
- [76] Novins, K. L., Sillion, F. X., and Greenberg, D. P., “An Efficient Method for Volume Rendering using Perspective Projection”, *Computer Graphics* (Proceedings of the San Diego Workshop on Volume Visualization), 24(5), November 1990, pp. 95–102.
- [77] Paglieroni, D. W. and Petersen, S. M., “Parametric Height Field Ray Tracing”, In proceedings of *Graphics Interface '92*, May 1992, pp. 192–200.
- [78] Paglieroni, D. W. and Petersen, S. M., “Height Distributional Distance Transform Methods for Height Field Ray Tracing”, *ACM Transactions on Graphics*, 13(4), October 1994, pp. 376–399.
- [79] Perny, D., Gangnet, M., and Coueignoux, P., “Perspective Mapping of Planar Textures”, *Computer Graphics*, 16(1), May 1982, pp. 70–100.
- [80] Pfister, H. and Kaufman, A., “Cube-4 — A Scalable Architecture for Real-Time Volume Rendering”, In proceedings of *Visualization '96* (San Francisco, CA, October 27 – November 1), IEEE Computer Society Press, 1996, pp. 47–54.
- [81] Porter, T. and Duff, T., “Compositing Digital Images”, *Computer Graphics* (proceedings of SIGGRAPH), 18(3), July 1984.
- [82] Potmesil, M., and Chakravarty, I., “Modeling Motion Blur in Computer-Generated Images”, *Computer Graphics* (proceedings of SIGGRAPH), 17(3), July 1983, pp. 389–399.
- [83] Reeves, W. T., “Particle Systems — A Technique for Modeling a Class of Fuzzy Objects”, *Computer Graphics* (proceedings of SIGGRAPH), 17(3), July 1983, pp. 359–376. Also appears in *ACM Transactions on Graphics*, 2(2), April 1983.
- [84] Reynolds, R. A., Gordon, D., and Chen, L., “A Dynamic Screen Technique for Shaded Graphics Display of Slice-Represented Objects”, *Computer Vision, Graphics, and Image Processing*, 38(3), June 1987, pp. 275–298.
- [85] Robertson, P. K., “Fast Perspective Views of Images Using One-Dimensional Operations”, *IEEE Computer Graphics and Applications*, 7(2), February 1987, pp. 47–56.

- [86] Robertson, P. K., “Spatial Transformations for Rapid Scan-Line Surface Shadowing”, *IEEE Computer Graphics and Applications*, 9(2), March 1989, pp. 30–38.
- [87] Sabella, P., “A Rendering Algorithm for Visualizing 3D Scalar Fields”, *Computer Graphics* (proceedings of SIGGRAPH), 22(4), August 1988, pp. 51–58.
- [88] Samet, H., *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*, Addison-Wesley, 1990.
- [89] Shirley, P. and Tuchman, A., “A Polygonal Approximation to Direct Scalar Volume Rendering”, *Computer Graphics*, 24(5), November 1990, pp. 63–70.
- [90] Sobierajski L. M. and Avila R. S., “Hardware Acceleration for Volumetric Ray Tracing”, In proceedings of *Visualization '95* (Atlanta, Georgia, October 29 – November 3), IEEE Computer Society Press, 1995, pp. 27–34.
- [91] Sutherland, I. E., Sproull, R. F., and Schumacker, R. A., “A Characterization of Ten Hidden Surface Algorithms”, *ACM Computing Surveys*, 6(1), pp. 1–55.
- [92] Turkowski, K., “Anti-Aliasing through the Use of Coordinate Transformations”, *ACM Transactions on Graphics*, 1(3), July 1982, pp. 215–234.
- [93] Tuy, H. K., and Tuy, L. T., “Direct 2-D Display of 3-D Objects”, *IEEE Computer Graphics and Applications*, 4(10), November 1984, pp. 29–33.
- [94] Upson, C., and Keeler, M., “V-BUFFER: Visible Volume Rendering”, *Computer Graphics* (proceedings of SIGGRAPH), 22(4), August 1988, pp. 154–159.
- [95] Vezina, G., and Robertson, P. K., “Terrain Perspectives on a Massively Parallel SIMD Computer”, *Scientific Visualization of Physical Phenomena* (proceedings of *CG International '91*), 1991, pp. 163–188.
- [96] Watt, A. and Watt, M., *Advanced Animation and Rendering Techniques: Theory and Practice*, ACM Press, 1992.
- [97] Westover, L. A., “Interactive Volume Rendering”, In proceedings of *Volume Visualization Workshop* (Chapel Hill, N.C., May 18–19), Department of Computer Science, University of North Carolina, Chapel Hill, N.C., 1989, pp. 9–16.
- [98] Westover, L. A., “Footprint Evaluation for Volume Rendering”, *Computer Graphics* (proceedings of SIGGRAPH), 24(4), August 1990.
- [99] Westover, L. A., *SPLATTING: A Parallel, Feed-Forward Volume Rendering Algorithm*, Ph.D. Dissertation, Department of Computer Science, The University of North Carolina at Chapel Hill, 1991.

- [100] Whitted, T., “An Improved Illumination Model for Shaded Display”, *Communications of the ACM*, 23(6), June 1980, pp. 343–349.
- [101] Wilhelms, J. and Van Gelder, A., “A Coherent Projection Approach for Direct Volume Rendering”, *Computer Graphics* (proceedings of SIGGRAPH), 25(4), July 1991, pp. 275–284.
- [102] Williams, L., “Pyramidal Parametrics”, *Computer Graphics* (proceedings of SIGGRAPH), 17(3), July 1983, pp. 1–11.
- [103] Williams, P. L., “Visibility Ordering Meshed Polyhedra”, *ACM Transactions on Graphics*, 11(2), April 1992, pp. 103–126.
- [104] Wolberg, G., *Digital Image Warping*, IEEE Computer Society Press, 1990.
- [105] Wright, J. R., and Hsieh, J. C. L., “A Voxel-Based, Forward Projection Algorithm for Rendering Surface and Volumetric Data”, In proceedings of *Visualization '92* (October 19–23, Boston, MA), IEEE Computer Society Press, 1992, pp. 340–348.
- [106] Yagel, R., Ebert, D. S., Scott, J., and Kurzion, Y., “Grouping Volume Renderers for Enhanced Visualization in Computational Fluid Dynamics”, *IEEE Transactions on Visualization and Computer Graphics*, 1(2), July 1995, pp. 117–132.
- [107] Yagel, R. and Machiraju, R., “Data-Parallel Volume Rendering Algorithms”, *The Visual Computer*, 11(6), June 1995, pp. 319–338.
- [108] Yagel, R., and Shi, Z., “Accelerating Volume Animation by Space-Leaping”, In proceedings of *Visualization '93* (San Jose, California, October 25–29), IEEE Computer Society Press, 1993, pp. 62–69.