FILTERING VOLUMETRIC DATA

By
John W. Buchanan
B.Sc., University of Windsor, 1986
M.Sc., University of Toronto, 1988

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES
(DEPARTMENT OF COMPUTER SCIENCE)

Abstract

The display of volumetric data is a problem of increasing importance. The display of this data is being studied in texture mapping and volume rendering applications. The goal of texture mapping is to add variation to the surfaces that is not caused by the geometric models of the objects. The goal of volume rendering is to display the data so that the study of this data is made easier.

Three-dimensional texture mapping requires the use of filtering not only to reduce aliasing artifacts but also to compute the texture value which is to be used for the display. Study of two-dimensional texture map filtering techniques led to a number of techniques which were extended to three dimensions: namely clamping, elliptical weighted average (EWA) filters, and a pyramidal scheme known as NIL maps; (NIL stands for *nodus in largo*, the rough translation of which is *knot large*).

The use of three-dimensional textures is not a straightforward extension of the use of two-dimensional textures. Where two-dimensional textures are usually discrete arrays of texture samples which are applied to the surface of objects, three-dimensional textures are usually procedural textures which can be applied on the surface of an object, throughout the object, or in volumes near the object. We studied the three-dimensional extensions of clamping, EWA filters, and NIL maps for filtering these textures. In addition to these three techniques a direct evaluation technique based on quadrature methods is presented. The performance of these four techniques is compared using a variety of criteria, and recommendations are made regarding their use.

There are several techniques for volume rendering which can be formulated as filtering operations. By altering these display filters different views of the data can be generated. We modified the NIL map filtering technique for use as a filter-prototyping tool. This extension incorporated transfer functions into the NIL map technique. This allows the manipulation of the transfer functions without requiring the re-computation of the NIL maps. The use of NIL maps as a filter-prototyping tool is illustrated with a series of examples.

Contents

List of Tables

LIST OF TABLES

List of Images

List of Figures

Acknowledgments

Many people helped.

Alain, from a guy with a *texan* accent to a mentor and a friend, I enjoyed the walk. Mary Jane, much more than a partner, a friend. Pierre, never did miss an opportunity to argue. Jeremy, many questions, few answers. Roy, always the right question. Ruhamah, always ready for a game of tag. Michal, charcoal eyes. George and Anne, family now. Michal, Teresa, Laura, and Anna, brats one and all.

Additional thanks to: Bob, for delivering the thesis to many places. Bob and Victoria, for taking me to a tragedy before my defense. Grace, for all her work. Peter, for his salsa recipe. The members of the imager lab, for making working in the lab so interesting.
The work reported in this thesis was strongly supported by my committee. I am very grateful to them for their support and encouragement.

Supervisory committee

Kellogg Booth, Computer Science
David Forsey, Computer Science
Alain Fournier (research supervisor), Computer Science
Maria Klawe, Computer Science
David Lowe, Computer Science
Günther Schrack, Electrical Engineering

University examiners

William Casselman, Mathematics
James Little, Computer Science

External examiner

Jane Wilhelms, University of California, Santa Cruz

This work was partially supported by grants from the Natural Science and Engineering Council (NSERC) of Canada.

ACKNOWLEDGMENTS

Dedication

Alfred John Buchanan

20 Feb 1935-14 Mar 1977

It was one of those gifts that we loved so much. Dad had brought us some scraps from the print shop. This time it was a whole stack of cards measuring 5cm by 8cm. David and I had decided to make a periscope, we knew we needed a tube and mirrors at each end. After making the tube 5cm by 5cm we wanted to buy the mirrors for the end. –But what size should these mirrors be? We did not have a clue, so of we went to see dad. After carefully listening to our dilema he pulled out a pencil and wrote some numbers on paper then told us that we wanted 5cm by 7cm mirrors. You know what? They worked perfectly.....

This was magic, pure and simple, and I wanted it. How could someone get the measurements of a piece of glass off a piece of paper. In that moment you taught me more than you will ever know. Thanks for the magic dad.

DEDICATION

Chapter 1

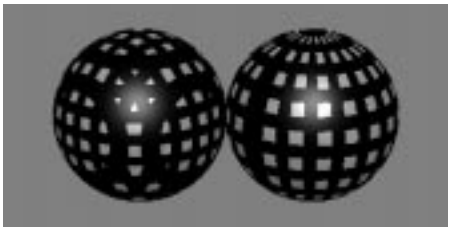
Introduction

A man who carries a cat by the tail
learns something he can
learn in no other way.
Mark Twain

1-1 Overview

The objects we encounter in every day life are usually far more complex than the objects that we can model and display using computer graphics. A great deal of this complexity is due to the detail of texture found on the surfaces of these objects. These textures are usually small enough so that modeling them with computer graphics primitives is not feasible. Texture mapping is an answer to the problem of incorporating this kind of texture into the images that we generate. Initially texture mapping was restricted to two-dimensional textures. By digitizing the surface characteristics of different objects in the real world we were able to wrap these textures onto our objects. Two-dimensional texture mapping was not a sufficient tool for modeling many of the textures that we encounter, because textures are defined throughout the material from which the object is made. Three-dimensional textures were introduced as a tool with which to simulate these solid textures. In Plate 1.1 we see examples of two-dimensional and three-dimensional textures applied to a sphere.

The study of three-dimensional textures has so far concentrated mainly on the modeling of textures. The resulting procedural models have provided us with a rich class of three-dimensional textures. In most cases the incorporation of a three-dimensional texture map into display (*rendering*) systems is a simple process that typically requires less work than two-dimensional textures. These textures are then computed by passing the shading parameters for a surface point to a procedural texture *engine*. Because the textures are procedurally defined they can be made to alter any of the variables in the shading equation. Even though the original three-dimensional textures were applied to



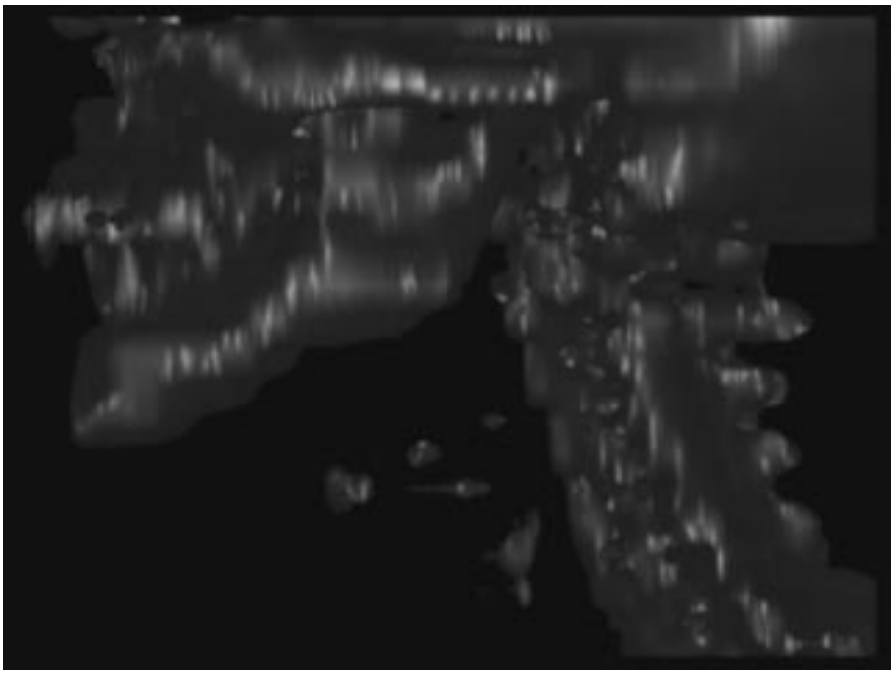
Three-dimensional (left) and two-dimensional texture (right) on a unit sphere. These procedural textures have similar definitions in two and three space.

Plate 1.1

the surfaces of objects, there have been a number of systems that have extended three-dimensional textures to include textures that are defined near the surfaces of the objects. When these *volumetric* textures are used, the display method must be adapted to compute the texture throughout the region in which it is defined. In a sense we can view these regions of texture as small volumetric data sets.

Volume rendering is an area of study that has received much attention of late. The research in this area is driven by a variety of applications that generate complex three-dimensional data sets. Examples of sources for data are, medical imaging (CAT scans, MRI scans), geology (seismological surveys), and simulations (fluid mechanics, stress analysis. Methods for displaying volumetric data may be categorized into two classes, surface extraction and direct display. Surface extraction methods use some property of the data to generate traditional computer graphics primitives, such as triangles or quadrilaterals. These objects are then displayed using traditional computer graphics techniques. Direct display techniques attempt to display the data without using intermediate geometric primitives.

The techniques for the direct display of the data can be split into two groups, pixel-based and voxel-based. Pixel-based volume renderers calculate the image pixel by pixel, usually by traveling through the data along the line of sight through each pixel. In most cases this is approximated using a ray casting technique that computes the shading



Geometric objects representing a volumetric data set. This data set is a 256×256 × 21 8 bit MRI scan of the region between the collar bone and the bridge of the nose. The 230,000 triangles used to generate this image were constructed using the marching cubes technique. The iso-surface was generated using a low-threshold of 0.31 and a high-threshold of 0.60.

Plate 1.2

Direct display of volumetric data. This display of the MRI data set was constructed using Sabella's volume rendering technique.

Plate 1.3

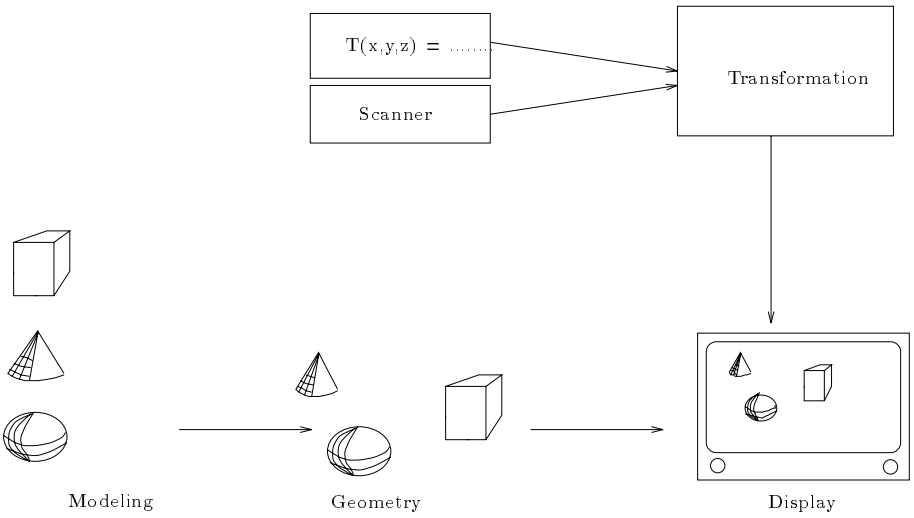
along the ray. Voxel-based volume rendering techniques compute the image by projecting individual voxels onto the screen and updating the affected pixels. If occlusion is important then the order in which the voxels are processed is important, and, they must be processed relative to the viewing direction.

The display of volumetric data, whether in texture mapping or volume rendering, will continue to be an interesting area of study. Texture mapping has been shown to be very useful as part of the computer graphics pipeline. The increased computational power makes the study of the display of larger and more complex datasets possible.

The computer graphics pipeline [Fole90, Newm79] has proven useful as a conceptual framework. Even though texture mapping is a subset of the final step of the pipeline (see Figure 1.1), we can think of the texture mapping process as itself being a pipeline. The modeling stage consists of developing the procedural models. The transformation stage is typically a set of simple transforms to map the object into the texture space. The display stage is where the texture is evaluated or sampled. Most of the research in three-dimensional texture mapping has concentrated on the modeling of textures.

The process of displaying volumetric data also fits into a pipeline model. Given a set of data we make a choice of which model will be used for the display. The transformation stage allows the user to set the viewing parameters to view a particular aspect of the data. It is only natural that the first two steps be simple because the modeling required for the display of a data set consists of choosing a display model and the viewing transformations. This typically results in a view of the data set where the data set almost fills the entire viewing screen.

Whenever a signal is sampled there is the possibility that this sampling will be inadequate. When the sampling of the signal is inadequate a variety of effects known as *aliasing* can appear in the reconstructed signal. This problem is exacerbated in computer graphics because we are required to sample objects and their associated textures in a non-uniform manner. The solution to this problem is well known, and requires the filtering of the textures to remove the frequencies that are causing the problems. The cost of evaluating these filtered samples is high. This high cost stimulated the search for efficient approximations to two-dimensional filters. An overview of the two-dimensional texture filtering literature is presented in Chapter 3.



The general graphics pipeline augmented with a texture pipeline.
Figure 1.1

1-2 Three-dimensional textures

Texture mapping has allowed us to generate images of rich visual complexity[Catm74]. Initially two-dimensional textures were used to modulate the surface characteristics of the objects. This technique was soon extended to allow the perturbation of the normals on the surface [Blin78a], and to modulate the transparency on surfaces [Gard84]. Using two-dimensional texture maps textured objects were easy to model and display. A problem with this technique was that sculpted objects were difficult to display. Three-dimensional textures [Gard84, Gard85, Grin84, Perl85, Peac85, Four86] were introduced partly to address this weakness of two-dimensional textures. Instead of requiring a complex mapping from the surface of the object to the texture space, a simpler set of affine transforms proved sufficient in most cases. Over this three-dimensional texture space a procedural model of the texture was defined. This model of the texture was then sampled at the surface points of the object being displayed. Because these textures were

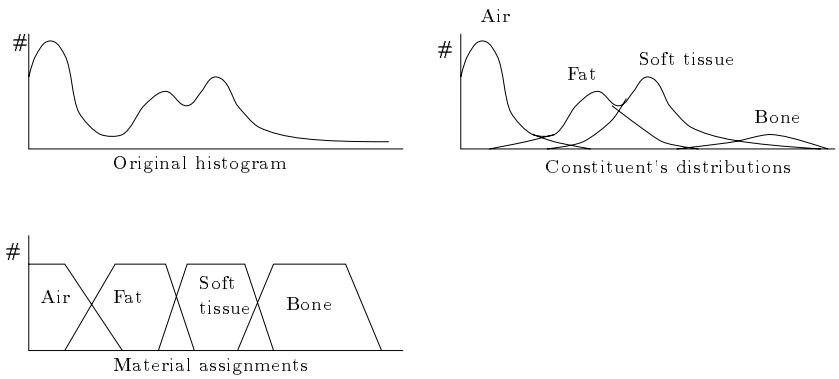
procedurally defined any of the shading parameters could be altered by this procedural texture *engine*. This is in contrast to two-dimensional texture mapping where most of the textures are discrete. The relative difficulty of acquiring three-dimensional data for the optical characteristics of solid materials and the costs of storing sufficiently high resolution data may account for this choice.

Early work on three-dimensional textures concentrated on texturing the surface of the object. More recent three-dimensional texture mapping systems have placed texture in a neighbourhood *near* the surface of the object [Perl89, Kaji89, Lewi89, Eber90]. These textures model objects with high frequency or fuzzy surfaces, such as fur. These textures require the computation of the texture throughout the texture region instead of at a single point.

1-3 Volume rendering

Volumetric data display is proving to be a useful tool in a variety of areas. In medical imaging it has allowed better diagnosis to be made. In geology it has given us a better understanding of the underground structure. The study of methods for displaying volumetric data has primarily concentrated on scalar volumetric data. The methods developed to display this data can be divided into two classes. The first set of methods displays the volumetric data by first fitting geometric objects to the data [Artz81a, Artz81b, Artz80, Artz79a, Artz79b, Chen85, Herm83, Herm82, Herm80, Wilh90a, Wilh90b, Lore87, Shir90, Galy91, Clin88]. These geometric objects or primitives are then passed to a *traditional* display system. We will refer to this set of techniques as *geometric volume rendering.* The second set of methods displays the data without fitting geometric objects to the data. This set of methods assumes that the data represents density samples taken throughout the volume. Displays of this volume are then generated by simulating the transport of light through the volume. These techniques have been called *direct volume rendering.* In the ensuing discussion we will refer to these techniques simply as *volume rendering* because geometric volume rendering techniques lie outside the scope of this manuscript.

Two approaches to volume rendering are being studied, voxel-based and pixel-based.



Density distribution, probability functions, and transfer functions for CAT scans. The linear approximations on the bottom are the transfer functions used by Drebin et al. [Dreb88]

Figure 1.2

Voxel-based techniques, [West90, Wilh91, Laur91, Dreb88, Upso88, Max90] calculate a volume of influence around the voxel and then project this volume onto the screen, this process is also known as voxel *splatting*. The pixels that lie in the projection of the voxel are updated as required. If occlusion is a desired property of the display process then the processing of the voxels must be done in an order determined by the viewing direction. The direction of this processing adds another label to these techniques; thus we have back-to-front and front-to-back voxel-based volume rendering. The manner in which a voxel is displayed depends on the shape of the voxel, the distribution of density through the voxel, and the intent of the method.

Pixel-based volume rendering techniques [Levo90a, Levo90d, Levo90b, Levo90c, Sabe88, Levo88, Upso88, Novi90] typically cast a ray from the eye through the pixel into the scene computing the integral of the densities/intensities along the ray. The resulting intensities are used to determine the colour of the pixel.

Voxel-based techniques have the advantage that they do not need to have all the data in main memory at the same time, but they have the disadvantage that they may have to process all of the data. Pixel-based techniques have the advantage that they can finish processing along a line of sight when one of the calculated quantities passes a pre-set threshold, however these techniques require that all of the data, or a major portion of it [Novi90], be in main memory. One could argue that as memory sizes and compute power increase this will not be a problem. But it is probably the case that no matter how large memory becomes, there will always be a larger data set that needs to be studied.

1-3.1 Transfer functions

Different aspects of the data can be highlighted by concurrently displaying separate transformations of the data. This separation is often done by means of procedurally defined *transfer functions*. These functions will either map a scalar data set into another scalar data set or into a multi-dimensional data set. By carefully tailoring these transfer functions different properties of the data can be highlighted. An example of transfer functions is presented by Drebin et *al.* [Dreb88]. Transfer functions were used to segment the data into three data sets corresponding to fatty tissue, muscle, and bone. These transfer functions and the related density distribution curves are presented in Figure 1.2.

To date most of the volume rendering systems presented point sample the data. Pixel-based techniques point sample each pixel and then use sample points along the ray generated from the pixel. Voxel-based techniques generate an approximation of the projection of the voxel onto the screen. The pixels that lie in this projection of the voxel are then updated according to the shading/display model being used. A few systems [Upso88, West90] deal with approximations to a three-dimensional integral over the volume of the voxel that is cast onto the pixel. Unfortunately these systems rely on point sampling to generate their approximations to the integrals. In this thesis we present a study of the filtering requirements of volume rendering and propose the use of three-dimensional NIL maps for approximating these filters.

1-3.2 Fourier-based methods

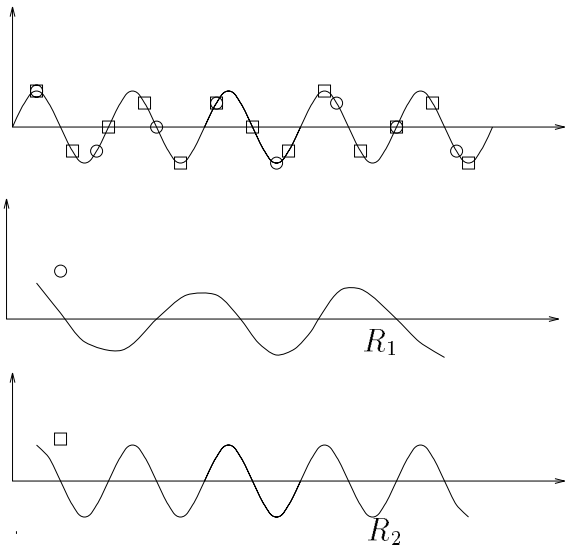
Levoy [Levo92] recently presented a technique for volume rendering that used the Fourier projection-slice theorem. The data is first transformed into the Fourier or frequency domain. Given the orientation of the desired display a corresponding slice of the Fourier transform is sampled. The inverse two-dimensional Fourier transform of this data yields an orthogonal display of the data. Most volume rendering techniques require the processing of $O(n^3)$ voxels per image. In contrast to this the Fourier based technique requires the processing of $O(n^2)$ pixels for the construction of the sample slice. The cost of transforming this slice into the spectral domain is $O(n^2 \log n)$. The pre-processing step costs $O(n^3 \log n)$. This means that the total cost is per view $O(n^2 \log n)$. This technique works well but allows little control of the display model since it is fixed.

In order to compute the slice an interpolating filter of width W is used. The cost of evaluating each element of the slice is W^3. This implies that the cost of evaluating the spectrum slice is actually $O(W^3 n^2)$. Work in this area continues with encouraging results [Tots93, Malz93].

1-4 Filtering

Given a signal $T(t)$ that is defined over some interval $[a, b]$ we wish to store a sampled or digitized version of the signal in such a way that we can later reconstruct the signal. We know from signal processing theory that the frequency of the sampling grid used must be greater than twice the highest frequency in the signal[1]; this is known as the Nyquist frequency. In Figure 1.3 we present an illustration of the problem that occurs when a bad sample set is used to reconstruct a signal. The circles depict a sample set with a frequency lower than the Nyquist frequency, and the squares a sample set with a frequency that is higher than the Nyquist frequency. The reconstructions that result from applying the ideal reconstruction filter to these sample sets are illustrated underneath. The erroneous signal that results from the first sample set is called an *aliased* signal. When we are sampling signals with more complex frequency spectra we must either

[1] For a good overview of signal theory see [Rose76].

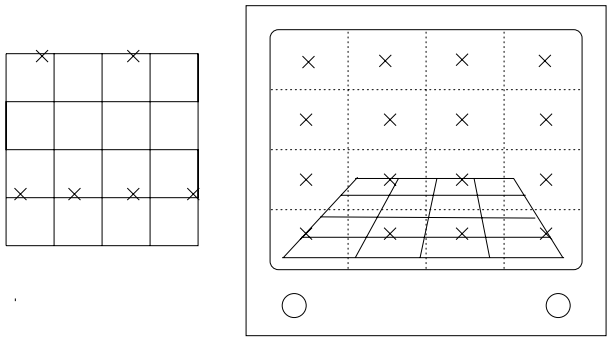


Two reconstructions of a sampled signal. The first reconstruction (R_1) results from using an ideal (sinc) reconstruction filter on the samples defined by the circles. The second reconstruction is generated from the samples defined by the squares. Because the first sample set sampled below the Nyquist frequency it is impossible to correctly reconstruct the original signal.

Figure 1.3

sample at or above the Nyquist frequency or remove the high frequency components that are causing the aliasing. This latter process is called *filtering*.

These sampling problems are compounded in computer graphics since we must often sample our signal using non uniform sample grids. A simple example of this situation is presented in Figure 1.4. In this figure we see a view of a square that is being displayed using a perspective transform. Because the picture elements or pixels on the viewing plane are regularly spaced we sample the square based on this grid. On the left of the illustration we see the texture displayed with the sample points highlighted on it. We notice that these samples are not regularly distributed throughout texture space even though the samples are uniformly distributed on the viewing plane. If the square is substituted with a large plane the spacing between the texture sample points can become arbitrarily large near the horizon.



Non uniform sampling in computer graphics. The uniform grid of samples generated on the screen does not generate a uniform grid of samples on the texture.

Figure 1.4

For a given texture the resulting sampling frequency may be lower than the texture's Nyquist frequency. We may choose to take more samples to ensure that the Nyquist frequency is satisfied. By averaging these samples a representative value can be computed. In order to compute this average we must determine an area of the texture space from which the samples are to be taken. This is usually accomplished by using some profile or perimeter of a pixel, such as a circle. The texture elements or *texels* within this projected pixel are then averaged to generate the filtered sample.

Once we have developed this filtering process we can start to consider more sophisticated filters. The Bartlett filter and the Gaussian filter have proven useful in the computer graphics literature. These filters have been used for anti-aliasing. The use of filters in computer graphics is not restricted to anti-aliasing. By tailoring different filters we can compute different effects, such as motion blur and depth of field. In both of these situations the filters are not removing aliasing frequencies but are being used to compute the value of the texture. In the case of motion blur the filter is deformed, or spread out, along the path in which the surface point is traveling. By averaging the texture under this filter we can produce motion blur effects. In a similar way the *blurriness* of the filter can be defined as a function of the distance from the focal plane. In this way a depth of field effect can be computed. Because the cost of directly computing the convolution of

these filters is high, research has focused on finding reasonable approximations to these filters.

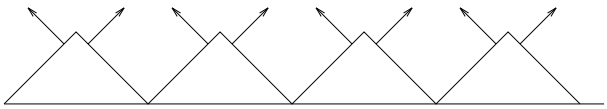
In order to apply a two-dimensional texture map to an object in a meaningful way we must develop a u, v parametrization of the surface. This parametrization is then used to index into the two-dimensional texture. Constructing these parametrizations becomes increasingly difficult as the complexity of the object increases. As the complexity of the u, v parametrization increases so does the complexity of the filter shape required for anti-aliasing. Fortunately there has been some work done on constant cost approximations to these filters [Will83, Four88a, Gots93]. One of the most successful of these approximations uses pyramidal representations of the data to approximate the filter[Four88a]. The constant cost is a result of approximating the filter at different levels of the pyramid, thus requiring a fixed number of samples to be taken from the data.

Suppose we wish to texture an object in such a way that it appears as though it were carved out of some solid material. Three-dimensional textures provide adequate tools for this task, since the coordinates on the surface of an object can be used to index into the solid texture. This *carved out of* effect is difficult if not impossible to accomplish using two-dimensional textures.

With two-dimensional textures much of the complication of the filter's shape was due to the complex parametrizations used. With three-dimensional texture maps we do not have to concern ourselves with this aspect since the transformations used to map the texture onto the object are not that complex. Instead, we now have to worry about the model of the material that is used for the object. If the material is opaque the texture is only visible on the surface, and if the material is translucent then the texture is important in a region or volume near the surface of the object. This means that if we are attempting to filter the texture on an object that is made of a opaque material we must tailor the filter to concentrate on the texture at the surface. On the other hand when we wish to filter textures on objects that have translucent properties then we must use a filter that processes a volume near the surface of the object[2].

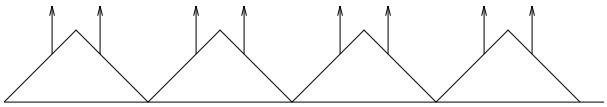
[2]Note that in the case of translucent textures the texture may have to be evaluated throughout the whole volume along the viewing direction.

Current implementations of three-dimensional textures allow any of the variables of the shading equation to be altered. This raises a number of filtering issues. Consider the case where it is not the colour of the object that is altered, but the normal of the surface (*bump mapping*). The bumpy or rough surface that is produced cannot be filtered using the traditional filtering approach since the normal is not a linear component of the shading equation. Even though the usual filters cannot be used on these normals, the filters do provide a means of removing the aliasing [Blin78c, Blin78a]. A simple example serves here to illustrate how using standard filters removes aliasing for bump mapping, but does not do it in a "correct" way. Consider the surface with the distribution of normals detailed in Figure 1.5. If these normals are replaced with their average the resulting normal distribution bears no resemblance to the original distribution (Figure 1.6). There is ongoing work in the area of filtering normal distributions [Four93].



Normal distribution

Figure 1.5



Filtered normal distribution

Figure 1.6

1-4.1 Colour textures

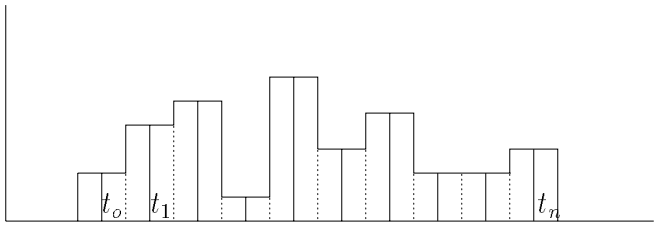
In this thesis we will be dealing with those elements of the texture space that can be '*averaged*' i.e. variables that are linear. When dealing with three-dimensional texture we will primarily concern ourselves with textures that alter the colour of the object.

1-4.2 Filtering semantics

In its most general form a filter is a weighting function $F(t)$ applied to another function $T(t)$. There are different reasons for filtering. Anti-aliasing was the original reason for this work. It soon became apparent that the use of filters for three-dimensional textures provided a far richer set of operations than simply anti-aliasing. In both three-dimensional texture mapping and volume rendering there are a large number of possible filters that can be applied to the data. We will not attempt to enumerate these filters in this thesis, but rather we will select some examples of filters and show how their application can be evaluated using a set of approximating techniques. These techniques are developed and evaluated in Chapter 4 and 5 of this dissertation. Chapter 6 discusses the use of filters for volume rendering.

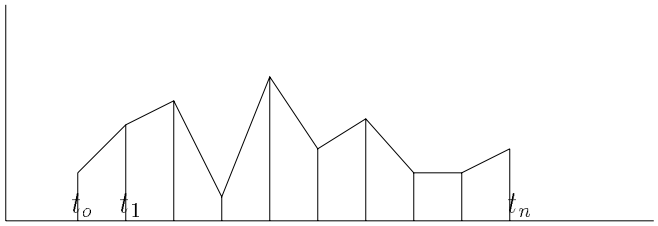
1-4.3 Reconstruction filters

The reconstruction of the data is another area where there is potential for aliasing or reconstruction artifacts to appear. For the most part the two reconstruction filters that have been used are the box filter and the tri-linear interpolation filters. Figures 1.7 and 1.8 illustrate these reconstruction filters in one dimension for a particular sample set. When the box reconstruction filter is used the image often looks as though the volume is made up of uniform cubes. This effect is somewhat diminished when tri-linear interpolation is used. To date there is no discussion on using higher order reconstruction filters in volume rendering applications. Because most three-dimensional textures are procedurally defined the issue of reconstruction filters has not been so relevant.



Box filter reconstruction of a one-dimensional signal. The bold vertical lines indicate the position and values of the sample set.

Figure 1.7



Linear interpolation reconstruction of the signal used in the previous figure.

Figure 1.8

1-4.4 Filter approximation requirements

In some sense two-dimensional filtering is a much simpler task than the filtering of three-dimensional textures. The initial motivation for this work was to provide filters for anti-aliasing of three-dimensional textures. As we started to look at three-dimensional textures it became obvious that filters could be used for more than simple anti-aliasing. For some of the newer *volumetric* textures the display of the texture required an averaging over a region near the surface of the object. This calculation can easily be formulated as a filtering operation. Similar filters can be used in the display of volumetric data. Different displays of the volumetric data can thus be generated by designing different display filters.

This leaves two possible avenues of research, filter design and filter evaluation or approximation. Considering the path by which we arrived at this point, it is natural that we chose to concentrate on filter approximation techniques. Our hope is that the results in this thesis will allow the further investigations of filter design with applications in volume rendering and texture mapping. In order to support further study into filter design we will base our analysis primarily on the flexibility of the filter approximating technique. Other evaluation criteria might include, pre-processing costs, filter evaluation cost, fidelity measures, and visual effects.

1-5 Thesis Goals

The main contributions of this thesis can be summarized as follows:

- An overview of the three-dimensional texture mapping filtering issues is provided. We study the relationship between two-dimensional computer graphics sampling and filtering problems and the related three-dimensional problems.
- The usefulness of filtering is shown to be greater than that of simply anti-aliasing. We show how some of the display problems in volumetric texture display and volume rendering can be formulated as filtering problems.
- The evaluation of these filters is costly. We develop three techniques for approximating the evaluation of filters. These are:

  - Direct convolution evaluation using numerical quadrature.
  - Elliptical Weighted Average (EWA) filters. This technique is developed as an extension of the similarly named two-dimensional technique presented by Greene and Heckbert [Gree86].
  - NIL maps. This technique is also developed as an extension of the corresponding two-dimensional technique presented by Fournier and Fiume [Four88a].

- These three techniques are studied along with a fourth technique from the literature. This technique is known as Clamping [Nort82, Perl85] We compare their flexibility, performance, and domain of application. Examples of their application to texture mapping and volume rendering are also included.

- We present the results of an initial investigation into the use of filters for volume rendering.

1-6 A Road-map to this thesis

There are five chapters that follow. These are:

- **Chapter 2**
  Definitions
  An overview and precise definition of the terms, concepts, and formulas used throughout the dissertation.

- **Chapter 3**
  Related work
  This work was influenced by the two-dimensional texture mapping, the three-dimensional texture mapping, and the volume rendering literature. In this chapter we present an overview of the literature from these three areas.

- **Chapter 4**

  Filtering techniques

  Three filtering techniques were developed for volumetric data, direct evaluation by quadrature, EWA filters, and NIL maps. These techniques are presented with discussions on their possible implementation. A fourth technique from the literature, known as clamping, is also presented briefly.

- **Chapter 5**

  Comparison and application of the techniques

  The four techniques are evaluated with regard to their application to texture mapping and volume rendering. We present example images illustrating application of these techniques to texture mapping.

- **Chapter 6**

  Filters for volume rendering

  Three examples are used to illustrate the use of filters for volume rendering.

- **Chapter 7**

  Conclusions

  Summary, conclusions, and recommendations of dissertation.

- **Appendix A**

  Implementation details of NIL maps

  Implementation details of NIL maps. A variety of speed-ups for the technique are presented.

- **Appendix B**

  High level view of NIL code

  The code at the heart of the NIL map implementation for volume rendering.

- **Appendix C**

  Code required to make a NIL motion blur filter

  An example of C code required for the implementation of a three-dimensional motion blur filter.

Chapter 2

Definitions

It is not necessary to understand
things in order to argue about them.
Pierre de Beaumarchais.

Definition 1 Volumetric Data:

Volumetric data is a discrete data set or a continuous function defined over a volume $V \subseteq \Re^3 \rightarrow \mathcal{D} \subseteq \Re^p$.

Definition 2 Discrete Volumetric Data:

A discrete set of samples of volumetric data acquired by some sampling process over a finite volume.

Definition 3 Procedural Volumetric Data:

A continuously defined volumetric data set which is defined algorithmically.

Three-dimensional texture maps have been implemented as procedural textures. Most of the acquired volumetric data used in volume rendering is discrete.

Definition 4 Texture:

A texture is a map from a geometric space $\Re^n$ *to a texture space* $\Re^p$.

Examples of textures include, colour textures ($\Re^2 \rightarrow \Re^{(red,green,blue)}$), normal perturbation or bump maps ($\Re^2 \rightarrow \Re^{(n_x,n_y,n_z)}$). In practice two-dimensional texture maps have been discrete and three-dimensional textures have been procedural.

Definition 5 Frame Buffer:

A portion of main memory dedicated to the storage of an image.

Usually a frame buffer is associated with a display device. If this is the case there is an implicit mapping from the frame buffer values to the intensities displayed on the display device.

Definition 6 Pixel:

The smallest addressable element of a frame buffer.

As such the strict definition of a pixel is a numerical value representing the colour at a point in an image. There is some confusion inherent in this term, partly due to the confusion between an image stored in a frame buffer and an image displayed on a physical screen. When an image is displayed on a screen the pixel values of the frame buffer are converted into an intensity setting over an area of the screen. Sometimes pixels have been defined as the smallest addressable elements on a display device.

The linking of a pixel to a physical display device is not without its merits. We can use a model of the "physical pixels" as an indication of the area of the viewing screen over which we must integrate the image display. In this sense pixels may have a shape. We must emphasize, however, that the shape of the pixel model bears little if any resemblance to the physical shape of pixels on a display device. For further discussion on this topic please see [Lyon89, Naim89].

Definition 7 Texel:

A texel is a texture element.

This term is used when referring to discrete textures. In three dimensions the texel may be either a volume over which a procedural texture is defined [Kaji89] or a sample of a procedural texture.

Definition 8 Voxel:

A voxel is an element of a discrete volumetric data set.

Notice that this definition is independent of the volume which the sample is thought to represent. This volume has been tailored according to the display technique. Some people consider it a rectangular parallelepiped of uniform density [Wilh91], others consider it a rectangular parallelepiped whose density is defined by a tri-linear interpolation of its eight corner points [Upso88], and others have taken the volume to be a spherical or elliptical volume enclosing the sample point [West90].

Definition 9 Filter:

Consider a signal $T(t)$ defined on $[-\infty, \infty]$. A filter is any function $F(t)$ such that the integral

$$I = \int_{-\infty}^{\infty} T(t)F(t)dt$$

is well defined.

Definition 10 Space invariant filter:

A space invariant filter is a function which is translated and applied to a signal. The filtering operation is then a function of the centre of the filter t_o.

$$I(t_o) = \int_{-\infty}^{\infty} T(t)F(t - t_o)dt.$$

This operation is called the convolution of the filter with the signal at t_o. In two dimensions this convolution is defined by

$$I(u_o, v_o) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} T(u, v)F(u - u_o, v - v_o)dudv,$$

and in three dimensions by

$$I(u_o, v_o, w_o) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} T(u, v, w)F(u - u_o, v - v_o, w - w_o)dudvdw.$$

Definition 11 Ray tracing:

The process of intersecting a line defined by an origin and a point on the viewing plane with geometric objects.

Typically this process is used for rendering or displaying objects. The viewing plane is discretized according to the size of the frame buffer being used. The rays are used to approximate the path which the light arriving at a point on the viewing plane has followed. This is typically done backwards, i.e. proceeding from the viewer's position out into the geometric definition of the scene.

Definition 12 Ray marching:

The process of stepping along a ray and sampling some function at each step.

These samples may be used to find a property of the function or may be incorporated into an overall computation which yields a single value. In the case that an object is defined by an implicit surface $F(x, y, z) = 0$, the ray marching technique can be used to find the surface. If the object being displayed has a volumetric texture near the surface the ray marching technique can be used to approximate the transport of light through this texture volume, thus yielding a color for the display of the pixel in question.

Definition 13 Voxel splatting:

The process of projecting the volume representation of a voxel onto a viewing plane. When the area of the image screen is found the affected pixels within this area are updated.

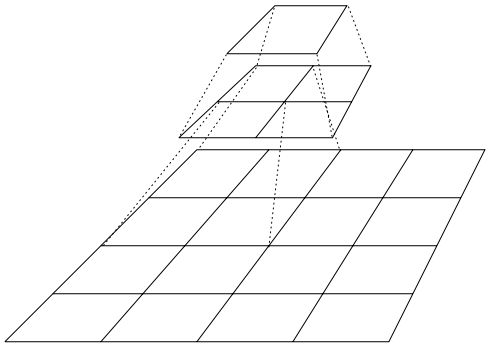
Definition 14 Pyramidal data structure:

A pyramidal data structure is a hierarchical data structure[Tani75, Rose75]. The data structure is constructed in such a way that for a one-dimensional data set each level requires half the storage of the next lower level. This provides a pyramid of representations for the data.

This technique has been used primarily in the vision literature. The first application of this storage technique serves as an example to illustrate both the data structure and how the levels provide multi-resolution representations of the data.

Definition 15 MIP map:

Multum In Parvo (Many things in a small place)
A MIP [Will83] map is a pyramidal data structure for representing two-dimensional textures. Each level contains texels which are the average of the four texels in the level directly below them. If the original image is of resolution $x = y = 2^p$ then the height of the MIP map is p. In this case the single texel on the $p + 1^{th}$ level is an average of the whole texture. This scheme is illustrated in figure 2.9.



Pyramidal representation of two-dimensional data

Figure 2.9

Definition 16 NIL map:

Nodus In Largo (Knot large)

A NIL [Four88a] map is a pyramidal data structure in which a set of basis functions have been pre-integrated with a texture. These pre-integrated basis functions can then be used to approximate the convolution of filters with a texture.

Definition 17 EWA filters:

Elliptical weighted average filters is a technique which takes advantage of the radial symmetry of a Gaussian filter so that the filter evaluation can be accomplished using a simple table-lookup.

If an affine transformation was used to generate the elliptical area or volume over which the filter is to be evaluated this technique can be used.

Definition 18 Transfer functions:

A set of functions which map a scalar data set into one or more different scalar sets.

In this dissertation the term transfer function indicates a function which maps a density distribution into another density distribution. The use of these functions is important in the volume rendering literature since it allows users to easily segment the data they are studying. These segments of the data are often mapped into other scalar data sets. These new data sets can then be displayed concurrently or separately.

Chapter 3

Related work

He who boasts of his descent
praises the deeds of another.
Seneca

The work reported here was motivated by a variety of sources. The images and ideas of three-dimensional texture mapping motivated the research into this area. Many of the details that remain open to study in three-dimensional texture mapping have been extensively studied in two-dimensional texture mapping. Two techniques for the filtering of two-dimensional texture stand out when this literature is read with the idea of extending the techniques to three-dimensional textures, namely EWA filters and NIL maps. The initial study of three-dimensional texture filtering showed that in many cases the problems being studied were similar to those being studied in volume rendering. In this section we present an overview of the relevant literature from two-dimensional texture mapping, three-dimensional texture mapping, and volume rendering.

3-1 Two-dimensional textures

Computer graphics has shown that the display of shaded polygons is easily done. Fortunately for us, the real world is much richer than this.[1] This richness is due partly to the texture details on many of the objects we see. In order to generate comparable images we must develop techniques for adding this complexity to objects. One approach would be to generate geometric models of the texture and use these models to compute the texture components of the image. The work on anisotropic reflection [Ward92, Poul90, Poul89, Kaji85, Bren70] is an example of such an approach.

Two-dimensional colour textures was the first tool with which texture information could be added to geometric objects. Using scanned textures, objects can be mapped

[1] This assertion is one that the post-modern architects seem bent on nullifying. Their frenzied desire to populate our cities with buildings that look like their simple computer graphics models still continues.

with these textures. In carpentry veneering is often used to enhance the look of furniture built from inferior wood[2]. When one encounters furniture that has been veneered it is quite easy to spot the discontinuities in the texture. Two-dimensional texture mapping suffers a similar problem.

The mapping from the surface of the object to the two-dimensional texture space is accomplished by generating a two-dimensional parametrization of the surface. When we need the texture parameters for a point (x, y, z) on the surface we evaluate the corresponding parameters (u, v) and use them to index into the texture. As an example of such a mapping we can use the sphere. For a given point on its surface (x, y, z) we have the corresponding spherical coordinates (r, θ, ϕ). Because we are interested in texturing the surface of the object we can ignore r and use θ and ϕ as the texture parameters. If we have a texture defined over the $[1, 0] \times [1, 0]$ square in u, v space we have to find a mapping from $[0, 2\pi] \times [0, \pi]$ to $[1, 0] \times [1, 0]$. Thus the map

$$\begin{aligned} u(x, y, z) &= \frac{\theta(x, y, z)}{2\pi} \\ v(x, y, z) &= \frac{\phi(x, y, z)}{\pi} \end{aligned}$$

will wrap the texture around the sphere. This mapping introduces a singularity at each of the poles of the sphere. These singularities and the complexity of some of the parametrizations required for two-dimensional texture mapping motivated some of the early work on three-dimensional texture mapping [Peac85].

Two-dimensional texture mapping techniques have been concerned with the related sampling and filtering issues from the very beginning. Most of the textures that are being used by two-dimensional texture mapping applications are discrete textures. The sampling of these texture data sets introduced many undesirable artifacts. As we discussed previously the only solution was to incorporate filtering into the texture sampling process. In general this requires filtering with space variant filters, a process that precludes

[2]Even though the primary application of the veneer process is to disguise, another similar process is marquetry. In this process small, usually geometric, shapes are glued to a surface to produce strikingly beautiful patterns. 'Marquetry taken to an extreme' would be a fitting description of intarsia. This process uses marquetry to generate images of impressive complexity. These processes are *hopefully* a better analogy for the two-dimensional texture mapping process.

the use of multiplication in the Fourier domain. This means that the only remaining option is to compute the direct convolution of the texture with the filter. This usually involves computing the integral of the filter centered at u_o, v_o with the texture over some integral area $\mathcal{A}$.

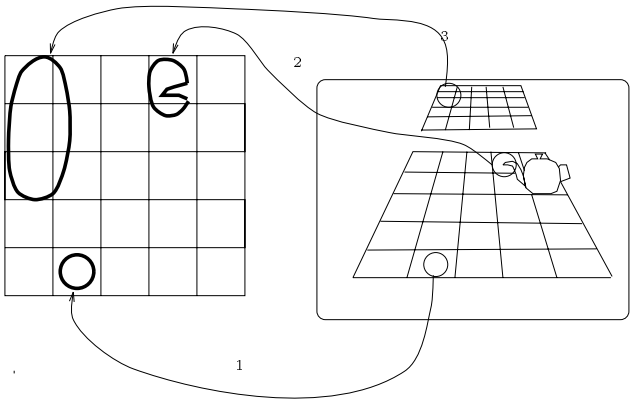
$$I = \int\int_{\mathcal{A}} T(u,v)F(u-u_o, v-v_o)dudv$$

Figure 3.10 shows a simple computer graphics scene. The textured objects in this scene are the two checkered planes that are mapped with the same texture. By picking three pixels we illustrate some of the problems of finding the appropriate filter to apply to the texture. For simplicity's sake let us assume that the area on the viewing plane over which we want to compute the texture is a circle. The first pixel circle is near the bottom of the screen and when it is transformed into texture space it suffers little distortion. The second pixel is near the top left of the screen and its projection into texture space introduces a strong distortion caused by the perspective transformation. The third pixel contains two objects, the spout of the teapot and the checkered board behind the teapot. The projection of this pixel into texture space should take into account the amount of the texture that is obstructed by the teapot's spout. The solution to the problem of occlusion has usually been addressed by subdividing the pixel into smaller regions until we can assume that only one object covers each sub-pixel. An average of these sub-pixels is then used as the final pixel colour [Fium83a, Fium83b, Carp84]. If it were possible to tailor the filter so that the weights of the filter in the occluded regions were set to zero then this occlusion problem would be addressed. Thus we have a complex set of areas over which the convolution of a filter and the texture must be computed. As the scene complexities increase so the complexity of the shape of these filters will also increase.

Once we have found the area of integration we must compute the integral. The cost of evaluating these integrals motivated the search for filter approximating methods.

Catmull [Catm74, Catm75] is generally credited with introducing two-dimensional texture mapping to computer graphics. Because the texture was tied to bi-cubic parametric patches the (u,v) parameters of the patch were used to index into the texture.[3]

[3]This parametrization of the surface does allow textures to be applied to the surfaces of the objects. There



Texture distortions due to perspective projection, transformation, and occlusion.
Figure 3.10

Aliasing was reduced by filtering the patch segments with a box filter placed over the pixel. In addition to this box filter the texture was pre-filtered to remove excessively high frequencies. A pyramid filter was proposed by Blinn and Newell [Blin78b, Blin78a]. Using the quadrilateral approximation to the projection of a square pixel into texture, a pyramid filter was computed as a weighted average of the texels under the distorted pyramid. Feibush, Levoy, and Cook [Feib80b] proposed a rather complex method for approximating a Bartlett filter or triangle filter. First the bounding rectangle of the pixel is projected into the texture space. The texels in the resulting quadrilateral are projected back to the screen and a weighted sum of the texels that project into the circular representation of the pixel is performed. A modification to this method is presented by Gangnet, Perny, and Coueignoux [Gang82, Pern82]. In their method sample points on the screen are projected into the texture and used in the weighted sum. In most circumstances these two techniques perform essentially the same filtering operation, the projection of screen samples into texture space requires the evaluation of the texture at points between texels. If the cost of evaluating the reconstruction filter for the texture

is a problem in that the parametrization induced by a particular basis may not be a desirable parametrization of the surface.

is high then Gagnet's technique will be costly. On the other hand there may be situations where the somewhat superior sampling technique employed by Gangnet produces a better result.

A different approach was proposed by Norton, Rockwood, and Skolmoski [Nort82]. They approximate a box filter in the Fourier domain (sinc filter) by the first two terms in its power series expansion $1 - x^2/6$. The textures that can be filtered with this technique are those textures that can be expressed as $T(x, y) = A + F(x, y)$, where A is a constant term and $F(x, y)$ has a simple Fourier representation. The clamping is applied to the $F()$ component of the signal depending on the area of the texture map that is to be filtered.

The high cost of computing these approximations motivated the study of pre-filtering techniques for approximating filters. Dungan, Stenger, and Sutty [Dung78] used pyramidal data structures [Tani75, Rose75] to allow the use of pre-filtered images in texture mapping. They generated a filtered pyramid of the image using a box filter to generate the different levels. Based on the area that the filter covers in texture space they select a level in the pyramid and use the appropriate texel in this level for the filter. Williams [Will83] suggests using a tri-linear interpolation scheme on the same pyramid where the sample point now lies between levels. Bi-linear interpolation is used on each level to determine two values of the texture. Linear interpolation between levels is used to determine the final value of the texture that is to be used.

A variant of this pyramidal approach is to use a summed area table as suggested by Crow [Crow84] and also by Ferrari and Sklansky [Ferr84, Ferr85]. This technique allows the approximation of rectangular area filters to be computed. The method proposed by Ferrari and Sklansky also allows arbitrary rectilinear polygons to be used as filters. Glassner further extended this method to allow arbitrary quadrilaterals to be approximated [Glas86]. The tables for this approach are constructed by pre-integrating in u and in v. An extension to this idea was proposed by Heckbert [Heck86]. His method relies on the identity

$$f(x) * g(x) = \frac{d^n f}{dx^n}(x) * \left[\left(\int \right)^n g(x) dx^n \right]$$

With Heckbert's technique filters are approximated by axis-aligned B-splines.

Greene and Heckbert [Gree86] use the radial symmetry of the Gaussian filter in their approximation technique for Gaussian filters. Assuming that pixels are circular their projection onto a plane is either an ellipse, a parabola, or a hyperbola. When the projection of the pixel is an ellipse the resulting elliptical Gaussian is easily found by the combination of a scale and rotation. The projection of the filter can easily be computed when the circular representation of the pixel projects to an ellipse. For each texel in the bounding box of this ellipse the distance from the centre of the ellipse is evaluated. The value of the filtered sample is then the weighted sum of all the texels inside the ellipse. If the filter being approximated is a radially symmetric filter, then the values of the filter can be pre-computed in one-dimension. The weight of the filter is then a simple table look up instead of the computation of the filter weight, which may be costly. Discussions of the generalization of EWA filters to three dimensions are presented in Chapter 4.

A study of conformal mapping with an application to texture mapping by Fiume, Fournier, and Canale [Fium87] motivated the study of space variant filters. NIL maps are another use of pyramidal data structures proposed by Fournier and Fiume [Four88a]. In their approach the filter is approximated by a set of parametric patches[4]. By analyzing the integrals of the filter approximation with the texture it was found that the integrals of the basis functions with the texture are independent of the shape of the filter. Because these integrals are independent of the filter they can be pre-computed. The approximation to the integral of the filter with the texture is then computed by finding the control points for the patches that are to approximate the filter. These control points are then used as weights for the pre-integrated basis functions. If a hierarchical approach is used for approximating the filter the cost of this approximation is no longer dependent on the size of the filter but on the number of patches in the hierarchy that approximates the filter. In chapter 5 we present a more detailed description of NIL maps and their extension to three dimensions.

A method derived from NIL maps has recently been proposed by Gotsman [Gots93]. Instead of using parametric spline patches to approximate the filters they construct a set

[4]In their implementation they used constant, bi-linear, bi-quadratic, and bi-cubic Catmull-Rom patches as an example. They point out that there is no restriction on the class of patches used so long as they can be defined by a reasonably 'nice' set of basis functions.

of basis functions using singular value decomposition. By restricting the class of filters that they are going to approximate they are able to generate basis functions that better approximate this class of filters. They illustrate their method by developing a set of basis functions for Gaussian filter applied to ellipses.

Most of the work in two-dimensional texture mapping has been concerned with the development of fast approximations to the required filtering operations. Little consideration has been paid to the properties of the filters. In their paper, Mitchell and Netravali [Mitc88] present a class of cubic reconstruction filters with a discussion of the tradeoffs involved in using these filters for reconstruction of a signal. The class of filters they studied is parameterized by two parameters. Over this two-dimensional space they classify the filters based on the observed visual characteristics of these filters. The main characteristics used for this classification were ringing, blurring, and anisotropy. Based on this visual classification they present a *map* of these filters over the two-dimensional space defined by the parameters. The map divides fairly simply according to the visual characteristics they were looking for.

One of the points they make in their paper is that it is difficult to measure objectively the performance of filters in computer graphics. They rank their filters using several subjective visual properties. This is probably the best measure we may have for filters in computer graphics.

Unfortunately this is the only work of which the author is aware in which the properties of filters in the context of computer graphics have been studied. This kind of analysis is more popular in the vision field where filters are used for a variety of purposes such as edge detection [Rose76, Cann86], texture segmentation [Bovi87], and detection of optical flow [Adel85, Heeg87, Heeg88].

There has been some work in the development of texture models for two-dimensional texture mapping. Feibush and Greenberg [Feib80a] showed how *artificial* textures could be used in the context of architectural design. Schweitzer [Schw83] showed how artificial textures could be added to objects to aid in their understanding. The textures were tied to geometric properties of the objects such as curvature or normal orientation. Using statistical models for texture Gagalowicz et al. [Benn89, Gaga88, Gaga87, Gaga86, Ma86, Gaga83] developed a texture mapping technique that relies on these statistical models.

First a white noise texture is mapped onto the textured regions of the image. These textured regions are then adjusted so that they have the same statistical properties as that of the texture model. Because the process is performed on the image after the rendering of the geometric objects has occurred there is no simple way of ensuring that the same point on an object receives the same texture in two different images.

Another approach to two-dimensional texture mapping is to use these techniques to display textures that are related to a dynamic process. By altering the texture van Wijk [Wijk91] was able to animate two-dimensional data sets in a variety of ways. Reaction diffusion [Turk91, Witk91] is a model that has been proposed for the growth of textures. Examples of textures that this process is able to model include the fur markings on many animals such as zebras, tigers, and giraffes. The reaction diffusion process is used to produce two-dimensional textures that can then be mapped onto the objects using a two-dimensional parametrization. Even though reaction diffusion textures can produce striking images it is not clear how to set up the parameters to produce a particular texture. Generating new textures using this technique can involve the search in a large parameter space.

Research in filtering two-dimensional texture maps has concentrated on finding approximations to the convolution of the filter with the texture. This convolution is necessary because the shape of the filter changes throughout the scene. A variety of schemes have been proposed for this approximation. Of these schemes two approaches stand out. Elliptical weighted average (EWA) filters [Gree86] and NIL maps [Four88a]. EWA filters allow the use of the Gaussian or other radially symmetric filter with little memory overhead. The cost of evaluating the filter is dependent on the size of the filter in texture space. When it is known that a radially symmetric filter is to be used, this choice seems to be the appropriate one. When a more complex filter is required we may not be able use this technique. NIL maps, on the other hand, allow us to approximate arbitrary filters. The cost of evaluating one of these approximations is not dependent on the size or shape of the filter, but rather on the quality of the approximation required. In contrast to EWA filters the NIL map filtering approximation requires a considerable amount of memory for the storage of the NIL maps. In the next chapter we discuss the extension of these two filtering techniques to three dimensions.

3-2 Three-dimensional textures

The first references to three-dimensional textures were made by Fournier and Amanatides [Grin84], Gardner [Gard84, Gard85], Perlin [Perl85] and Peachey [Peac85]. Fournier and Amanatides mentioned it in another context. Gardner used an approximation to the Fourier series to model a variety of the effects seen in clouds. His textures were used to modulate the translucence of planes or ellipsoids. These objects were then grouped together to approximate clouds. Perlin [Perl85] used solid textures as part of his rendering package. Peachey [Peac85] presented the idea of solid textures with the goal of removing some of the parametrization problems associated with two-dimensional textures. Perlin's work was largely based on using a band-limited noise function for the modeling of textures. By applying a variety of transformations to these noise functions he was able to generate different textures. Because many of Perlin's textures are developed from band-limited noise functions, a clamping [Nort82] approach can be used to ensure that no aliasing frequencies are introduced to the texture while it is being generated. The cut off frequency for the clamping of the signal is computed as a function of the size of the projected pixel. Peachey recognized that, even though the introduction of these solid textures removed some of the aliasing problems that were due to the problems of two-dimensional parametrizations, the use of three-dimensional textures introduced another source of potential aliasing problems.

Lewis [Lewi89] proposed a different way of generating the noise function and its associated transformations. In addition to a new method for generating the noise function he, Perlin and Hoffert [Perl89], and Kajiya and Kay [Kaji89] showed how textures could be applied in regions or volumes near the surface of the object.

Kajiya and Kay developed a volumetric model for fur. This fur *texel* was then mapped onto the surface of the object. When a ray intersected the object the texture was computed by first calculating the entry and exit points of the ray into the texel volume. Between these two points ray marching was used to incrementally compute and accumulate the shading. The resulting value was used as the value of the pixel. Their shading model included interference and scattering.

Perlin and Hoffert, and Lewis generated deformed objects using three-dimensional textures. By embedding their objects in a solid density function they were able to use a thresholding of this function to define a surface. The object was then displayed using a ray marching technique. In this case the ray would not be computing the shading at each point, but rather looking for the point at which the pre-defined threshold was reached. When this threshold was found the shading was computed as if a surface was present at that point. This allowed them to generate objects with highly complex surfaces. These volumetric textures clearly blur[5] the distinction between textures and objects.

Greene [Gree89] presented volume based technique for growing plants. When automata were placed in a voxel grid with the correct rules, interesting plant-like objects were generated. The resulting volumetric objects were displayed using volumetric display methods, either surface extraction or direct display.

Ebert and Parent [Eber90] presented a system that allows the rendering of standard geometrical objects along with gaseous objects. The gaseous objects are modeled using Perlin's turbulence function. In order to display these hybrid scenes they used a variation of the A-buffer scan line rendering technique [Fium83a, Fium83b, Carp84].

In previous work the author [Buch91] made the case for the filtering of three-dimensional textures. Using Simpson's mid-point adaptive quadrature rules over rectangular parallelepiped regions provided a means for evaluating the required integral integrals [Burd81]. The cost of evaluating this integral can be high,[6] mainly due to the adaptive nature of the quadrature method used and the variety of shapes that the filter can assume. The cost of evaluating the integral is directly proportional to the volume of the rectangular parallelepiped over which the integral is being evaluated and the fourth derivative of the product of the filter with the texture function.

To date there are two filtering options for three-dimensional textures, clamping, as proposed by Perlin, and filtering over rectangular volumes, as proposed by the author. In the situations where we know the Fourier spectra of the signal, clamping the signal may suffice, however, most procedural textures are defined directly rather than by their

[5] Pun intended

[6] The Simpson's quadratic quadrature rule in one dimension requires a minimum of 5 samples to be taken. In a three-dimensional implementation this means that a minimum of 125 samples must be taken.

Fourier spectrum. If we could find the Fourier transform of these textures then clamping would be feasible. The class of functions that is available in most procedural texture *engines* is not a set of functions for which *nice*[7] Fourier transforms exist. An example of this is the step function, whose Fourier transform has infinite support. Because the procedural implementations of these textures usually allow the introduction of arbitrarily high frequencies, sampling these textures and finding their Fourier transform are not usually options. The method proposed by the author allows the filtering of arbitrary three-dimensional textures but in many situations this filtering becomes prohibitive in cost.

3-3 Volume rendering

The increase in available volumetric data and the decrease in the cost of computation power has stimulated research into the field of volume rendering. The display of this data has been studied along two directions, surface extraction and direct display.

The premise of surface extraction methods is that there is some property of the data that can be directly related to a surface in the data. In medical imaging it is obvious that the surface of a bone corresponds to a discontinuity in the density function[8]. Using this density discontinuity it is possible to extract surfaces that approximate the surface of the bones in the data. This approach has been followed by a variety of people [Artz81a, Chri78, Fuch77, Lore87]. The complexity of the generated geometric primitives depends on the reconstruction filter that is applied to the data. If the simple sample and hold filter is used the resulting signal is made up of rectangular voxels of uniform density. If the reconstruction is more complex then the procedure for generating the geometric objects that represent the surface becomes more complicated. This complication is due to the fact that a continuous signal must now be sampled and the surface reconstructed

[7] As in most cases "nice" means "easy to evaluate".
[8] CAT scans exhibit this property.

from these samples. Fortunately [9] the other reconstruction filter that is used is the tri-linear interpolation filter. Because this results in voxels with a tri-linear distribution of density throughout them it is simple to estimate the location of an iso-valued surface (iso-surface).

The direct rendering of volumetric data is accomplished by imposing a physical model onto the data [West90, Wilh91, Laur91, Dreb88, Upso88, Max90, Levo90a, Levo90d, Levo90b, Levo90c, Sabe88, Levo88, Upso88, Novi90]. This model usually has no relationship with the physical object from which the data was obtained. It also has little to do with the process by which the data was obtained. The display of the data is accomplished using the properties of this display model. Most display techniques use a varying density gas model. The density of the gas is related to the data set. Because most of data being displayed is scalar, this relationship is usually a linear one. If the data is not scalar then either a more complex display model must be found or the different dimensions of the data can be mapped into different scalar volumes that can then be displayed using the gas model. Kajiya and von Hertzen [Kaji84] suggested

$$I = \int_{t_a}^{t_b} e^{-\tau \int_{t_a}^{t} \rho(x(\mu),y(\mu),z(\mu))d\mu} \times \left[\sum_i I_i(x(t),y(t),z(t))p(cos\Theta_i)\right] \times \rho(x(t),y(t),z(t))dt$$

as an approximation to the amount of light that arrives at a point on the screen, where $x(t), y(t), z(t), x(\mu), y(\mu), z(\mu)$ are the equations that define the ray along which this integral is being computed and I_i is the integral representing the light that arrives from light source L_i to points in the data $x(t), y(t), z(t)$. Θ_i is the angle between the viewing direction and the direction of the ray. The integrals I_i need only be computed when the light locations are changed. τ is a user parameter that controls the strength of the scattering effect.

[9] The term *'fortunately'* here is used in the context of the complexity of the resulting code. The truth of the matter is that the issues of reconstruction filters and their associated properties in the context of volume rendering needs to be studied. Trousset and Schmitt [Trou87] indicate that there is a problem in using a simple reconstruction filter for the evaluation of the gradient. Rather than introducing a higher order filter they simply extend the $3 \times 3 \times 3$ box filter to a $5 \times 5 \times 5$ box filter.

If the integrals towards the light are dropped, as suggested by Sabella [Sabe88], then we have

$$I = \int_{t_a}^{t_b} e^{-\tau \int_{t_a}^{t} \rho(x(\mu),y(\mu),z(\mu))d\mu} \rho(x(t),y(t),z(t))dt \tag{3.1}$$

Integrating this equation over a pixel area on the viewing plane defined by $(u_0, u_1) \times (v_0, v_1)$gives

$$\int_{u_0}^{u_1}\int_{v_0}^{v_1} I du dv = \int_{u_0}^{u_1}\int_{v_0}^{v_1}\int_{t_a}^{t_b} e^{-\tau \int_{t_a}^{t} \rho(x(\mu),y(\mu),z(\mu))d\mu} \rho(x(t),y(t),z(t))dt du dv \tag{3.2}$$

Because the display model we use is arbitrary we can approximate $e^{-\tau \int_{t_a}^{t} \rho(x(\mu),y(\mu),z(\mu))d\mu} \approx e^{-\tau(t-t_0)}$. When this approximation is used the computation of the shading can be viewed as the convolution of an exponential decay filter with the data set.

$$\int_{u_0}^{u_1}\int_{v_0}^{v_1} I du dv = \int_{u_0}^{u_1}\int_{v_0}^{v_1}\int_{t_a}^{t_b} e^{-\tau(t-t_o)} \rho(x(t),y(t),z(t))dt du dv \tag{3.3}$$

This shows how the volume rendering problem can be formulated as a filtering operation. As in texture mapping, the cost of evaluating these convolutions is high, but if the approximation methods we develop for these filters are flexible enough there is no reason to restrict our choice of filters. We may wish to query some local property at some point in the data. If this query can be formulated as a filter operation then we may be able to use one of the approximation methods to evaluate it.

Thus we would like to provide users with a system that allows two modes of interaction, global and local. In the general or global inquiry stage the user is interested in finding global properties of the data. The local query mode may involve the search for a particular property near some point in the data. The filters used in these scenarios will be different. These different filters will then be approximated using similar evaluation techniques.

3-4 Wrapup

Adding textures to objects allows us to easily enhance the display of computer graphics models. Whether we choose to use two-dimensional textures or three-dimensional textures there is a set of filtering issues that must be addressed. In two dimensions we must filter the texture with a space variant filter. Much of the research in two-dimensional texture mapping has studied approximations to the evaluation of filters. Two of these filtering techniques were selected for further study, namely EWA filters and NIL maps.

Three-dimensional textures extended texture mapping so that objects could be textured with solid textures. The filtering of three-dimensional textures has not been studied much except for an application of the clamping technique. Most of the textures that are provided by three-dimensional texture mapping are procedural. Despite the increase in memory sizes, the cost of storing textures of sufficiently high resolution will continue to be prohibitive for the next few years, therefore filtering techniques proposed for three-dimensional textures must allow for the use of procedurally defined textures.

In conclusion we wish to develop filtering methods for three-dimensional texture mapping that:

- Allow arbitrarily shaped and scaled filters to be used.
- Handle three-dimensional procedural textures.

Volume rendering has studied a variety of approaches for displaying volumetric data. Of interest here is the set of volume rendering techniques that use ray tracing for the display of the data. These techniques are remarkably similar to the techniques used for the display of textures defined in volumes near the surface of objects. Transfer functions are useful tools for volume rendering. This means that it is important that any filtering techniques used for volume rendering not preclude the use of transfer functions.

Thus we require filters for volume rendering that:

- Allow approximation of arbitrarily shaped and scaled filters.
- Incorporate transfer functions.

In the next chapter we describe four filtering techniques. Each of these techniques has applications in texture map filtering, and one of the techniques has applications in the volume rendering area. The evaluation and comparison of the filtering techniques is presented in Chapter 5.

Chapter 4
Filtering techniques

Elutriate
Edulcorate

In this chapter we present the details of four filtering techniques for volumetric data. These techniques are clamping, direct evaluation, EWA filters, and NIL maps. Clamping is the application of the two-dimensional filtering technique of the same name [Nort82] to three-dimensional textures as suggested by Perlin [Perl85]. The direct evaluation technique [Buch91] uses Simpson's mid-point adaptive and Gaussian cubic quadrature rules for approximating the integral of the filter applied to the texture. The implementation of EWA filters for the filtering of three-dimensional textures is an extension of two-dimensional EWA filters presented by Greene and Heckbert [Gree86]. NIL maps are an extension of two-dimensional NIL maps presented by Fournier and Fiume [Four88a]. In addition to the extension to three dimensions it is demonstrated how NIL maps can be used with procedural textures and transfer functions.

4-1 Clamping

In their paper Norton *et al.* [Nort82] showed how the filtering of textures could be approximated using a Fourier representation of the texture. Given a complex-valued[1] texture defined by

$$I(x, y) = e^{i(kx+ly)}$$

they show that the application of a box filter over a parallelogram centered at (x_o, y_o) and defined by

$$(x_o, y_o) + s(x_1, y_1) + t(x_2, y_2) \quad \text{with} -1 < s, t < +1$$

[1] The signal need not be complex-valued. As is often the practice, complex variables are being used here to develop the theory. In practice all operations will be restricted to real valued signals.

can be approximated in the Fourier domain by

$$e^{(ikx_o+ily_o)}\left(1-\frac{(kx_1+ly_1)^2}{6}-\frac{(kx_2+ly_2)^2}{6}\right)=e^{(ikx_o+ily_o)}C(x_1,x_2,y_1,y_2,k,l).$$

Where $C()$ is the approximation to the Fourier representation of the box filter generated by using the first two terms of the power series expansion of the sinc function. Norton *et al.* further simplified this approximation by truncating this quadratic function so that it never became negative. The clamping function C is then defined by

$$C(x_1,x_2,y_1,y_2,k,l)=\begin{cases}\left(1-r(\frac{(kx_1+ly_1)^2}{6}\frac{(kx_2+ly_2)^2}{6})\right)\\ \text{if } \left(1-r(\frac{(kx_1+ly_1)^2}{6}-\frac{(kx_2+ly_2)^2}{6})\right)<1\\ 0 \text{ otherwise}\end{cases},$$

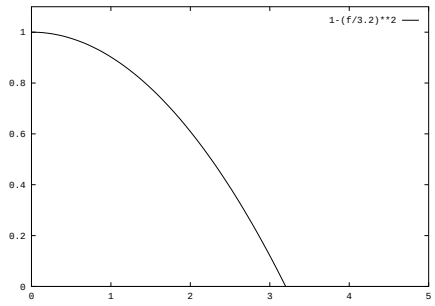
where r is a spread constant that can be used to control the width of the filter.

Perlin [Perl85] suggested a variant of the above method for filtering three-dimensional textures. The maximum allowable frequency is computed from the size of the projected pixel into texture space. By comparing the frequency of a texture component with this maximum frequency it is possible to exclude texture components that may introduce aliasing. The use of this step function for a clamping function can introduce discontinuities in the resulting image. We can use a simple variant of the quadratic approximation suggested by Norton *et al.* to reduce the impact of these discontinuities.

Given a pixel area on a viewing screen we project it onto the tangent plane of the object. From the size of this projection we compute the maximum allowable frequency f_{max}. The clamping function we use is then

$$C(f)=max(0,1-\left(\frac{f}{f_{max}}\right)^2).$$

This has the effect of gradually removing the aliasing frequencies before they are truncated. In Figure 4.11 we show the clamping function that results when $f_{max}=3.2$.



Clamping function $C = max(0, 1 - \left(\frac{f}{f_{max}}\right)^2)$ when $f_{max} = 3.2$

Figure 4.11

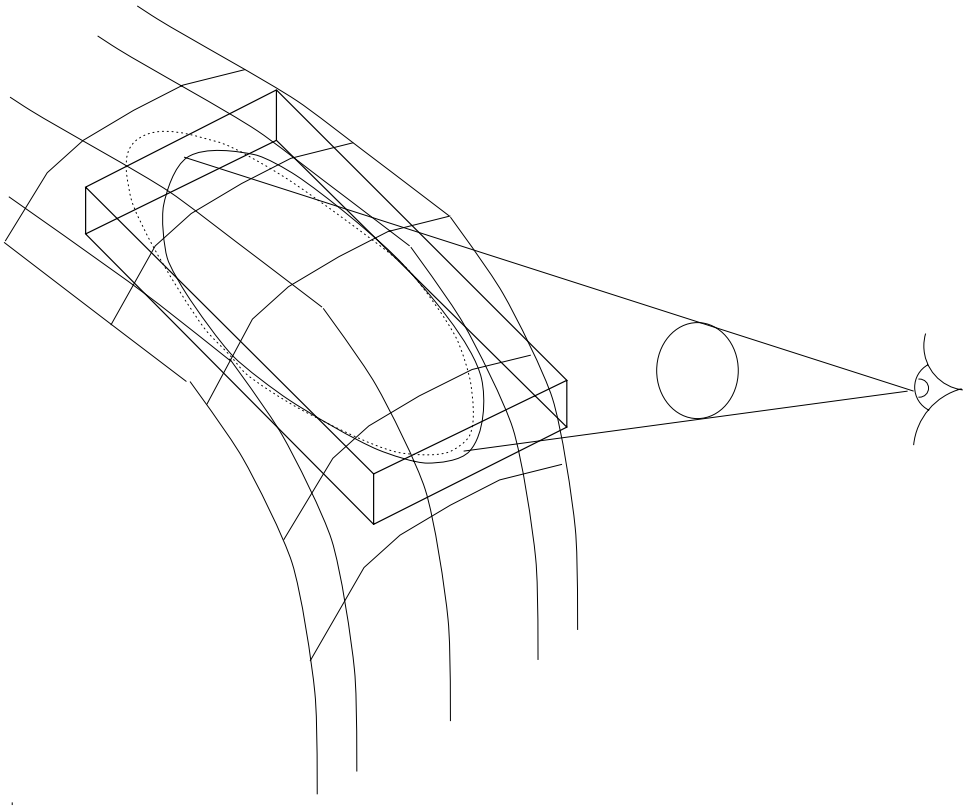
4-2 Direct evaluation

We developed a system for approximating three-dimensional filters using numerical integration [Buch91]. Two volumes of integration were developed, one aligned with the tangent plane of the object and the other aligned along the line of sight. The volume of integration aligned with the tangent plane is useful for evaluating the texture on objects with high opacity. The volume of integration aligned with the ray is useful for objects whose material has low opacity.

For those materials that have high opacity we require the evaluation of a filter over the surface of the object. The area over which this integral is to be evaluated is the projection of the pixel onto the surface. This area can be approximated by the projection of the pixel onto the tangent plane of the object. In order to reduce the aliasing that may be introduced by sampling the texture on the tangent plane we evaluate the filter over a volume that encloses this projection. This volume is computed by finding the rectangle in the tangent plane that encloses the ellipse[2]. This ellipse is the projection of the circular representation of the pixel onto the tangent plane. This rectangle is then extruded in a

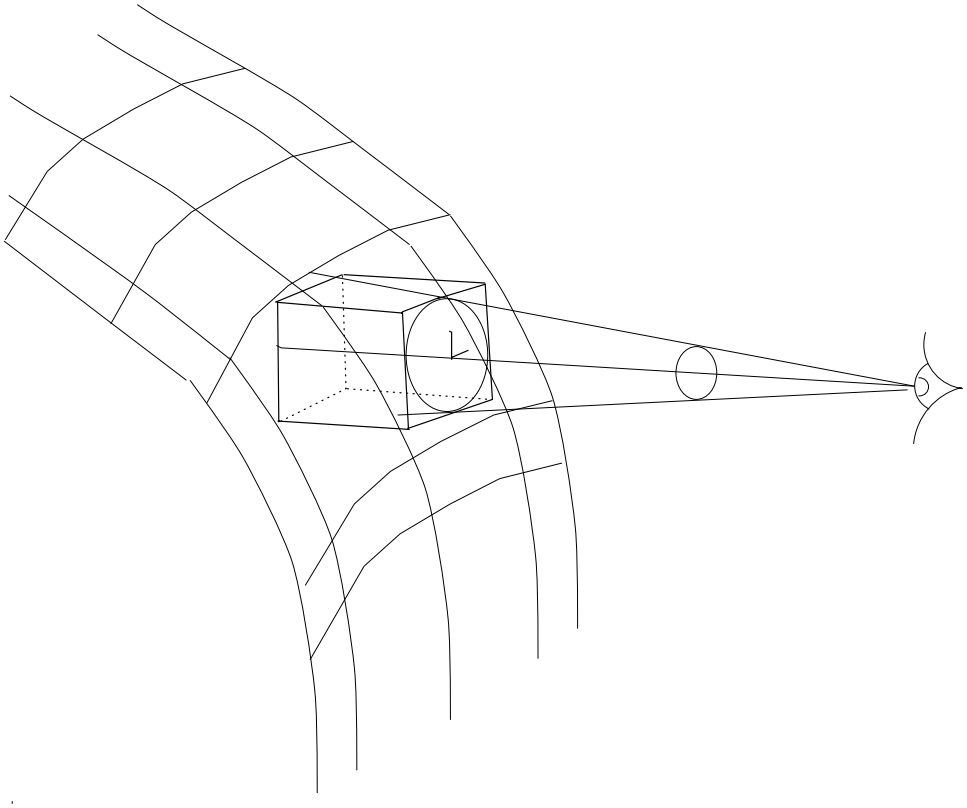
[2]In most situations the projection of the circular pixel onto the tangent plane will be an ellipse. In the rare cases where the circle projects to either a parabola, or an hyperbola we can either use a large ellipse to approximate the projection or use a pre-computed average for the filtered texture sample. If the DC, or average, component of the texture is known then this could be used as the average of the texture.

direction normal to the tangent plane. This is illustrated in Figure 4.12.



Box enclosing projection of circular pixel onto tangent plane.
Figure 4.12

The volume of integration aligned with the ray is constructed in a similar fashion. The pixel is projected onto a plane perpendicular to the viewing ray. The center of this projected pixel lies at the intersection of the viewing ray and the object. The bounding rectangle of this circle is then extruded into the object. The depth of this extrusion is controlled by the user. By varying this depth parameter (ϵ), materials of differing opacities can be modeled and displayed.



Box enclosing projection of circular pixel onto plane parallel to viewing plane.

Figure 4.13

Once a volume of integration V is defined we must evaluate the integral of the filter $F(u,v,w)$ and the texture $T(u,v,w)$ over this volume.

$$I = \int\int\int_V F(u,v,w)T(u,v,w)dudvdw$$

Evaluating this integral analytically is often not feasible, thus we must resort to numerical integration methods. Two numerical integration or quadrature methods were studied. These are adaptive Simpson's quadrature and fixed-point cubic Gaussian quadrature rules [Burd81]. When an integral volume is relatively thin in one or more of its dimensions it is appropriate to use the Gaussian rule for evaluating the integral along that direction.

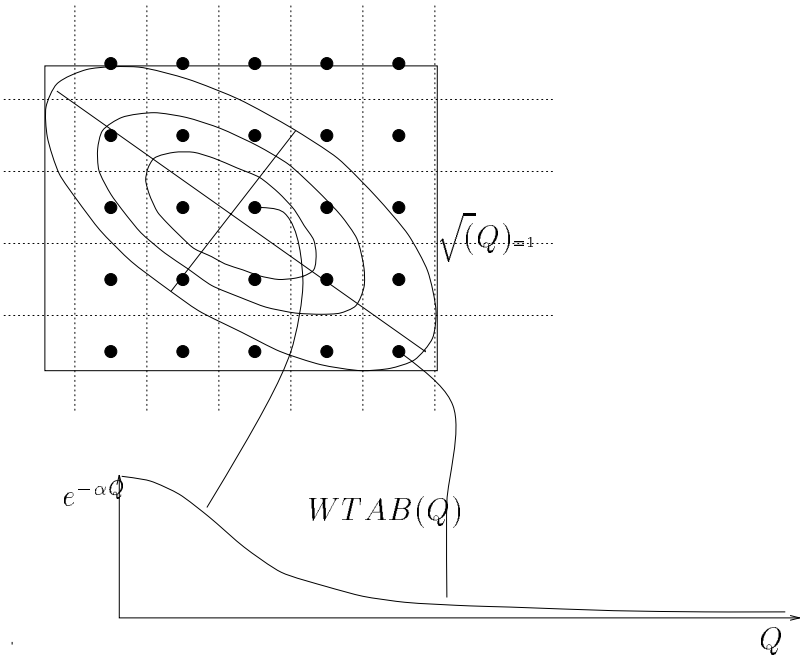
For the integral volumes that are aligned with the surface of the object we have found that a hybrid method consisting of Simpson's adaptive mid-point method in the two dimensions over the plane and a cubic Gaussian quadrature normal to this plane works quite well; in this case the minimum number of samples taken is $5 \times 5 \times 3 = 75$. This hybrid method works well because in most applications of this filter the dimension of the box normal to the plane is much smaller than the other two dimensions. Because this is not the case when we consider the integral volumes aligned with the ray, we have found that we need to use Simpson's adaptive quadrature rule in all three dimensions. When the full adaptive method is used the minimum number of samples taken is $5 \times 5 \times 5 = 125$.

The cost of using this approximation to the filter is dependent on the quadrature rule chosen. Simpson's adaptive rule uses an estimate of the fourth derivative of the function being integrated. This means that the cost of evaluating these three-dimensional filters is proportional to the magnitude of the fourth derivative of the product of the filter and the texture. If we wish to bound the cost of this approximation we can either restrict the level of subdivision that we allow Simpson's rule to take or we can use a fixed-cost quadrature rule.

4-3 Elliptical Weighted Average filtering

In a system designed for distorting images for later projection on a OMNIMAX^{TM} screen Greene and Heckbert [Gree86] approximate the convolution of a truncated Gaussian with textures over arbitrarily oriented ellipses. These ellipses are the projection of circular

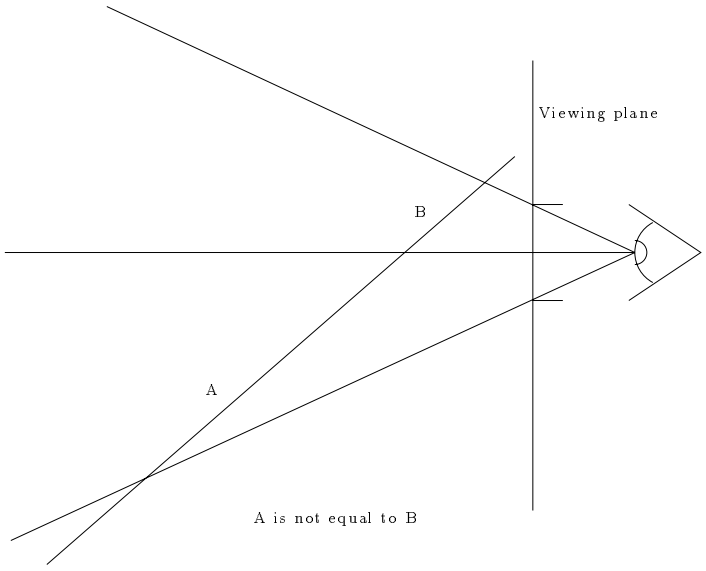
pixels into the texture space. The Gaussian filter is centered over the resulting ellipse and its weights are evaluated using the radial symmetry of the Gaussian filter. For each



Elliptical weighted average filter evaluation.
Figure 4.14

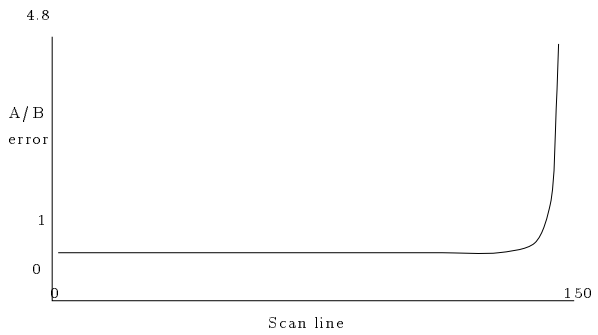
texel that lies within the truncated Gaussian the weight of the filter at that position is computed by first evaluating the square of the radial distance $||Q||^2 = du^2 + dv^2$ of the point from the center of the ellipse. This radial distance is then used to index into the pre-computed one-dimensional Gaussian filter. By using this pre-computed array of filter values the cost of evaluating the Gaussian filter is reduced to a table look-up operation. The use of this filtering technique allows the approximation of radially symmetric filters centered in the projected ellipse. When the perspective projection is being used the center of a circle does not project to the centre of the ellipse.

This is illustrated in Figure 4.15. In Figure 4.16 we see the error associated with centering the Gaussian filter as the filter projection approaches the horizon in the display of the image in Plate 4.2. This error is only significant near the horizon. Another



If the center of the circle projected to the center of the ellipse then A and B would be equal. This diagram shows that this is not the case.

Figure 4.15

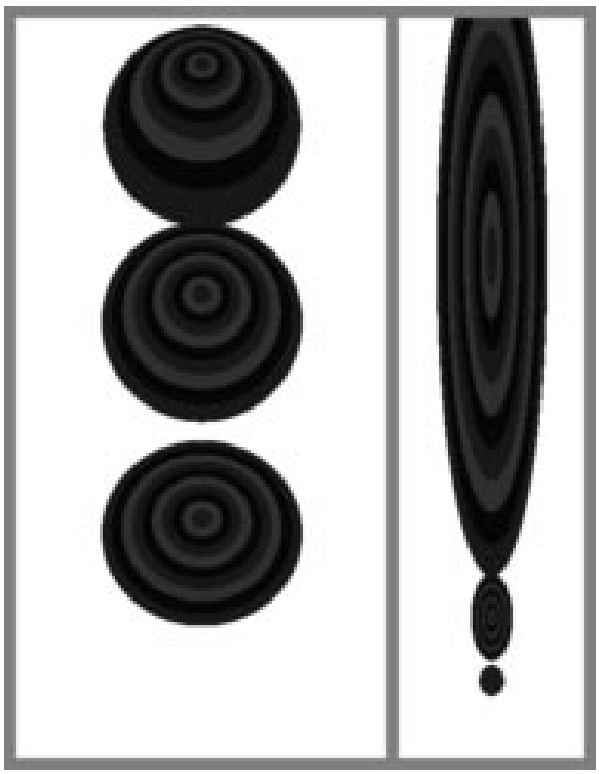


Error associated with centering a Gaussian. This ratio was computed on the image used in the next chapter.

Figure 4.16

example of this problem can be seen in Plate 4.1. In this plate we present two views of three ellipses. The first view has the eye positioned so that these ellipses project to circles of the same size on the screen. As can be seen from the second view, which shows an orthographic view of the ellipses, the texture on the ellipses is centered where the projected centre of the texture is not at the centre of the circles in the first view.

For three-dimensional texture mapping there are two extensions of this technique that we wish to explore. The first is to apply this technique to the projection of the pixel onto the tangent plane of the object. The other option is to extend this technique to three dimensions so that the filtering is performed over an ellipsoidal volume near the surface of the object. In this case the third axis of the ellipsoid will be defined by the same parameter that defined the normal dimension of the integral volume for objects made of opaque materials. When procedural textures are being used there are no restrictions on the positions of the sample points. This allows us to distribute the sample points in the bounding box aligned with the ellipsoid, rather than in the bounding box aligned with the texture. The number of points used to approximate a filter depends on the size of the filter. Because these ellipses can become quite large it is necessary to limit the number of sample points.



A Gaussian centered in texture space is not centered in screen space.
Plate 4.1

The filtered texture sample is then a weighted sum of texture samples. The texture is sampled at the sample point locations and the weight of this sample point is computed by a look-up of the pre-computed Gaussian.

4-4 NIL maps

In 1988 Fournier and Fiume [Four88a] presented an algorithm that used a pyramidal representation of the data to compute the filtering of two-dimensional textures in constant time (with respect to filter shape and size). An overview of NIL maps is presented, followed by discussions of the extensions of this technique to three dimensions, to include transfer functions, and to handle procedural textures.

Consider a signal $T(t)$ defined over some region $[a, b]$. We wish to evaluate the convolution of a filter $F(t)$ centered at t_o. This is evaluated using the following integral.

$$I(t_o) = \int_{-\infty}^{\infty} F(t - t_o)T(t)dt \tag{4.1}$$

If we assume that the signal is identically 0 outside of the region $[a, b]$ the integral becomes:

$$I(t_o) = \int_{a}^{b} F(t - t_o)T(t)dt \tag{4.2}$$

Now let us approximate the filter F as K curve segments over the interval $[a, b]$. With out loss of generality we can assume that the segments are of unit width, that is $b-a = K$. If this is not the case it is a simple matter to introduce a change of variables so that $b-a = K$ is satisfied. Each curve segment is defined by a set of basis functions $B_0, B_1, ..., B_{M-1}$ so that the filter is approximated by the union of these K segments.

$$F(t - t_o) = \bigcup_{m=0}^{K-1} \sum_{i=0}^{M-1} b_i^m(t_o, F)B_i(t - m) \tag{4.3}$$

Note the unusual notation for the control points of the curves. We have indicated that the control points are dependent on both the filter F and its location t_o. For simplicity's sake we will drop the $b_i^m(t_o, F)$ notation in favor of the simpler b_i^m notation. Substituting Equation 4.3 into Equation 4.2 we get:

$$I(t_o) = \int_{a}^{b} \bigcup_{m=0}^{K-1} \sum_{i=0}^{M-1} b_i^m B_i(t - m)T(t)dt \tag{4.4}$$

Because the integral is being evaluated over these K segments of the curve we can evaluate this integral as K distinct integrals. So, separating the integrals and introducing a change of variable $t \rightarrow t + m$ we have:

$$I(t_o) = \sum_{m=0}^{K-1} \int_{0}^{1} \sum_{i=0}^{M-1} b_i^m B_i(t)T(m + t)dt. \tag{4.5}$$

Because the b_i^m coefficients are independent of t:

$$I(t_o) = \sum_{m=0}^{K-1} \sum_{i=0}^{M-1} b_i^m \int_0^1 B_i(t) T(m+t) dt. \tag{4.6}$$

This means that the NIL cells C_i^m can be pre-computed.

$$C_i^m = \int_0^1 B_i(t) T(m+t) dt \tag{4.7}$$

The approximation to the integral 4.2 is then:

$$I(t_o) = \sum_{m=0}^{K-1} \sum_{i=0}^{M-1} b_i^m C_i^m \tag{4.8}$$

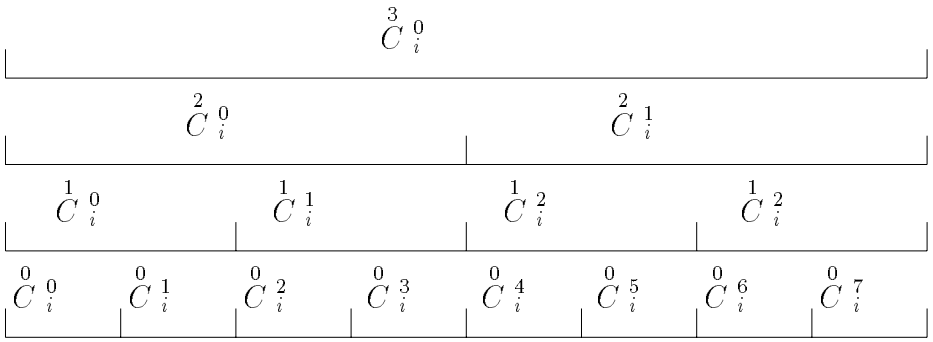
Unfortunately the cost of evaluating this approximation is still dependent on the width of the filter. A solution to this problem is to use our knowledge of the filter's properties to guide the approximation of the integral.

The first step in developing this constant-cost approximation is to compute a pyramidal NIL map. Let's use γ to indicate the level of the NIL map, where $\gamma = 0$ indicates we are on the lowest level of the representation, and increasing γ means we are generating levels whose NIL cells cover increasingly larger portions of the texture. For the time being let us also assume that $K = 2^p$ for some integer p. We deal with the case where K is not an integral power of 2 in Appendix A.

These new levels of the pyramid are defined by:

$$C_i^{\gamma m} = \int_0^1 B_i(t) T(2^\gamma m + 2^\gamma t) dt. \tag{4.9}$$

The pyramid that this produces is illustrated in figure 4.17. Consider a filter that is relatively smooth and spans the whole texture; such a filter is illustrated in Figure 4.18. Let us lay down eight points of interest, or trigger points, where we want the approximation of the integral to focus. Because the filter spans the whole texture we distribute these trigger points uniformly through the texture. We now place these trigger points at the level of the NIL pyramid that encloses them, in this case it is the top level. We will subdivide this NIL cell if there are more than two trigger points in it. This



Pyramid data structure for a one-dimensional NIL map

Figure 4.17

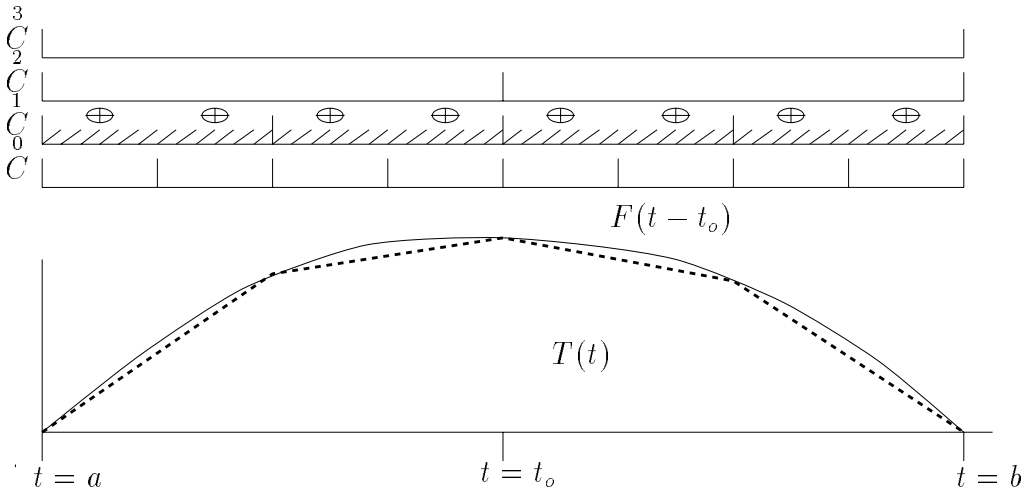
subdivision proceeds until only two or less trigger points lie in a NIL cell. The resulting subdivision is shown by the positions of the trigger points. The dotted line shows the resulting approximation to the filter if the basis B_i chosen is linear. [3] Pseudo code for this subdivision algorithm is presented in Figure 4.19.

In Figure 4.20 we illustrate a filter that has non-zero weights in a localized region around t_o. Because we know that the filter is localized in the area around t_o we increase the number of the trigger points in this region. Performing the subdivision as indicated yields a different hierarchy. This hierarchy focuses the evaluation of the filter in the region of interest, that was highlighted with a higher density of trigger points.

Another example of the use of this technique is presented in Figure 4.21. In this figure we see a highly localized filter whose approximation requires the use of negative levels of the NIL map. Using these negative levels gives a better approximation of the integral because they provide a better fit to the filter curve. Again we defer the details of the implementation of these negative levels to Appendix A.

Fournier and Fiume [Four88b] showed that in the case of two-dimensional texture maps the number of NIL cells chosen in this manner was linear in the number of trigger points and the tolerance. The proof of this in three dimensions is presented in the next

[3] Note that this approximation is a virtual approximation to the filter function. The values of the control points are used for the approximation of the integral of the filter with the texture and not for evaluating the curve.



Filter that spans the whole texture.

Figure 4.18

chapter.

4-4.1 Normalization

Using NIL maps as indicated allows the approximation of a range of filters in time independent of their width or shape. We wish to impose one further condition on this method. Given a filter whose integral over the interval $[a,b]$ is $K = \int_a^b F(t)dt$ and a constant texture $T(t) = T_o \quad \forall t \in [a,b]$ the integral

$$I(t_o) = \int_a^b T(t)F(t-t_o)dt \tag{4.10}$$

becomes

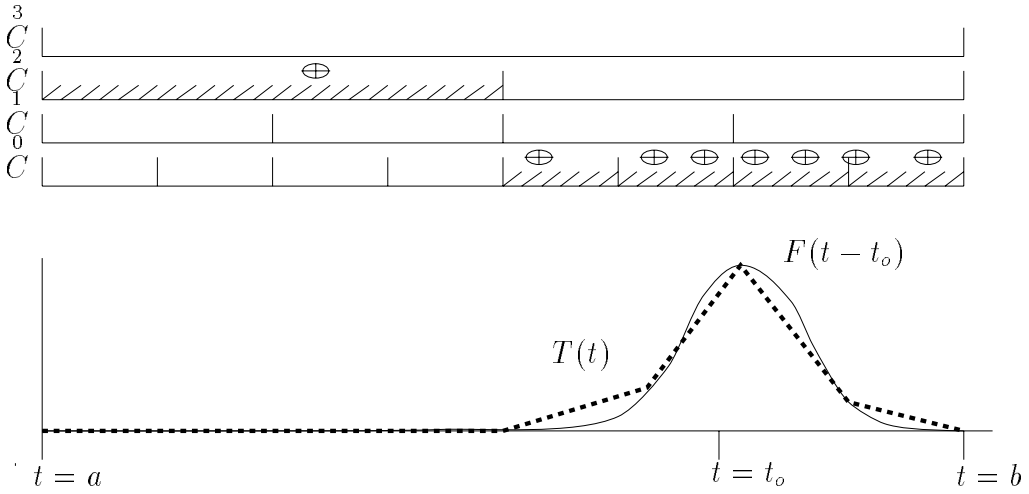
$$I(t_o) = T_o \int_a^b F(t-t_o)dt = T_oK \tag{4.11}$$

because the texture is constant. We shall require that the approximation of this convolution by any NIL map approximation be equal to the quantity T_oK. Because the texture is constant over the $[a,b]$ interval the NIL map coefficients $\overset{\gamma}{C}{}_i^m$ are simply the integral

```
First find the smallest cell that encloses the trigger points

cell ⇐Find_Lowest_Enclosing_NIL_Cell(trigger_points)
PROCEDURE Find_Nil_Hierarchy (points,cell, tolerance,γ_min)

Check to see if tolerance satisfied

  IF ( number_of_points_in_cell(cell, points)≤ tolerance ) then
    RETURN ( cell )
  END if

Check to see if maximum subdivision reached

  IF ( γ(cell)= γ_min ) then
    RETURN ( cell )
  END if

Split the cell into its eight children and process them

  hierarchy ⇐null
  FOR CHILD IN CHILDREN(CELL) DO
    hierarchy ⇐hierarchy + Find_Nil_Hierarchy(points,child,tol)
  END for
  RETURN ( hierarchy )
END Find_Nil_Hierarchy
```

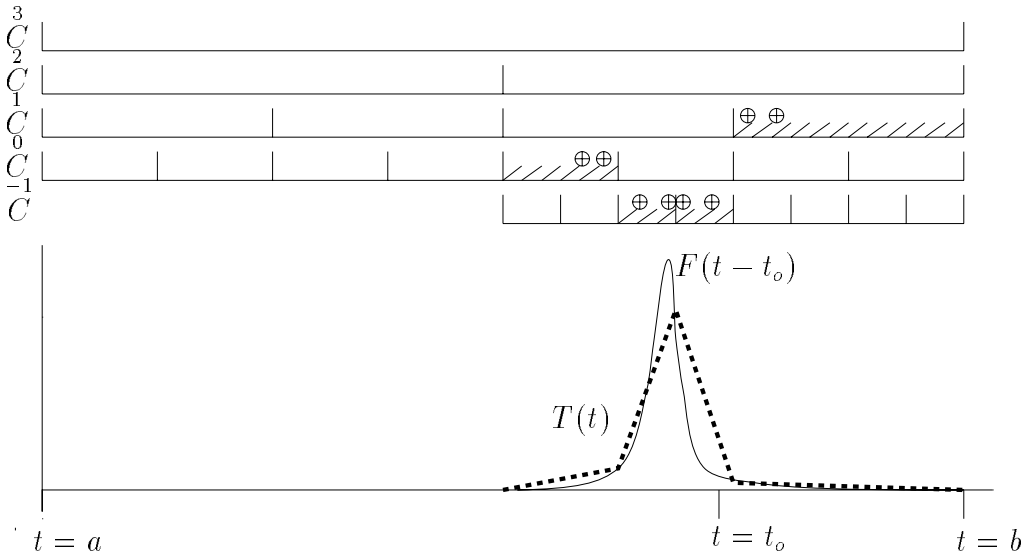
Generation of NIL map hierarchy

Figure 4.19



Using information about the shape of the filter we are able to place the trigger points closer together. By positioning the trigger points in this manner the resulting hierarchy is better suited to approximate this filter.

Figure 4.20



Narrow filter showing the need for negative levels.

Figure 4.21

of the basis functions multiplied by the constant K since:

$$C_i^{\gamma m} = \int_0^1 B_i(t) T(2^\gamma m + 2^\gamma t) dt = T_o \int_0^1 B_i(t) dt. \tag{4.12}$$

So for a particular approximating NIL hierarchy $\mathcal{H}$ to the filter $F(t)$ we require:

$$I_{\mathcal{H}} = T_o K = \mathcal{S} \sum_{p \in \mathcal{H}} \sum_{i=0}^{M-1} b_i^{\gamma m} T_o . \int_0^1 B_i(t) dt \tag{4.13}$$

Because T_o is constant we have that:

$$I_{\mathcal{H}} = T_o K = \mathcal{S} T_o \sum_{p \in \mathcal{H}} \sum_{i=0}^{M-1} b_i^{\gamma m} \int_0^1 B_i(t) dt. \tag{4.14}$$

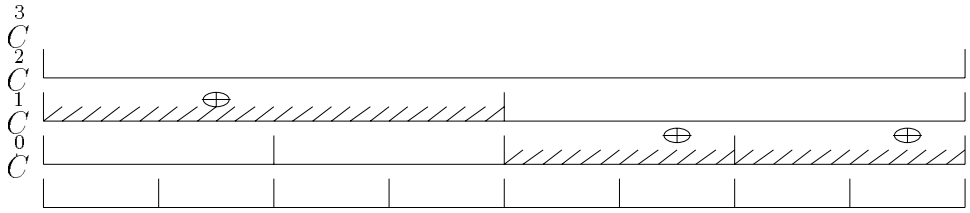
Thus the scale factor $\mathcal{S}$ is:

$$\mathcal{S} = \frac{K}{\sum_{p \in \mathcal{H}} \sum_{i=0}^{M-1} b_i^{\gamma m} B_i} \tag{4.15}$$

where the quantities $B_i = \int_0^1 B_i(t) dt$ can be pre-computed. In many situations the filters are normalized so that $\int_a^b F(t - t_o) = K = 1$. In this case the scale factor is simply

$$\mathcal{S} = \frac{1}{\sum_{m \in \mathcal{H}}^{\gamma} \sum_{i=0}^{M-1} b_i^{\gamma m} B_i} \tag{4.16}$$

4-4.2 Weighting the levels of the NIL maps



Approximation hierarchy at two levels

Figure 4.22

Another normalization problem is introduced when we use a hierarchy of NIL cells to approximate the filter. In Figure 4.22 we illustrate an approximation to a filter with three

NIL cells. Consider the situation when we set the filter $F(t) = 1$ over the approximation region and the texture $T(t) = 1$. The approximation to this filter is then computed by

$$I = \frac{\sum_{i=0}^{M-1} \overset{2}{b}_i^0 \overset{2}{C}_i^0 + \sum_{i=0}^{M-1} \overset{1}{b}_i^2 \overset{2}{C}_i^2 + \sum_{i=0}^{M-1} \overset{1}{b}_i^3 \overset{2}{C}_i^3}{\sum_{i=0}^{M-1} \overset{2}{b}_i^0 \overset{2}{B}_i^0 + \sum_{i=0}^{M-1} \overset{1}{b}_i^2 \overset{2}{B}_i^2 + \sum_{i=0}^{M-1} \overset{1}{b}_i^3 \overset{2}{B}_i^3}$$

But since $T(t) = 1$ we have:

$$I = \frac{\sum_{i=0}^{M-1} \overset{2}{b}_i^0 \overset{2}{B}_i^0 + \sum_{i=0}^{M-1} \overset{1}{b}_i^2 \overset{2}{B}_i^2 \sum_{i=0}^{M-1} \overset{1}{b}_i^3 \overset{2}{B}_i^3}{\sum_{i=0}^{M-1} \overset{2}{b}_i^0 \overset{2}{B}_i^0 + \sum_{i=0}^{M-1} \overset{1}{b}_i^2 \overset{2}{B}_i^2 \sum_{i=0}^{M-1} \overset{1}{b}_i^3 \overset{2}{B}_i^3} = 1$$

At first glance this appears to produce the correct result. However, the weights that are assigned to the different portions of the filter approximation are equal. This means that the contribution of the NIL cell that spans half the approximation ($\overset{2}{C}_i^0$) is weighted equally to those that span a quarter of the approximation ($\overset{2}{C}_i^2$ and $\overset{2}{C}_i^3$). This weighting of the NIL cells can cause a problem. A simple example is a texture that is zero over the interval $[0, \frac{1}{2})$ and one elsewhere:

$$T(t) = \begin{cases} t \in [0, \frac{1}{2}), T(t) = 1 \\ t \in [\frac{1}{2}, 1], T(t) = 0 \end{cases}$$

In this case we have

$$I = \frac{\sum_{i=0}^{M-1} \overset{2}{b}_i^0 \overset{2}{B}_i^0 + \sum_{i=0}^{M-1} \overset{1}{b}_i^2 0 + \sum_{i=0}^{M-1} \overset{1}{b}_i^3 0}{\sum_{i=0}^{M-1} \overset{2}{b}_i^0 \overset{2}{B}_i^0 + \sum_{i=0}^{M-1} \overset{1}{b}_i^2 \overset{2}{B}_i^2 + \sum_{i=0}^{M-1} \overset{1}{b}_i^3 \overset{2}{B}_i^3} = \frac{1}{3} \neq \frac{1}{2}$$

The solution is to weight the contributions of the NIL cells according to their height in the NIL map; the quantity $2^{\gamma_{max} - \gamma}$ accomplishes this. The integral approximation now becomes:

$$I = \frac{\frac{1}{2}\sum_{i=0}^{M-1} \overset{2}{b}_i^0 \overset{2}{B}_i^0 + \frac{1}{4}\sum_{i=0}^{M-1} \overset{1}{b}_i^2 0 + \frac{1}{4}\sum_{i=0}^{M-1} \overset{1}{b}_i^2 0}{\frac{1}{2}\sum_{i=0}^{M-1} \overset{2}{b}_i^0 \overset{2}{B}_i^0 + \frac{1}{4}\sum_{i=0}^{M-1} \overset{1}{b}_i^2 \overset{2}{B}_i^2 + \frac{1}{4}\sum_{i=0}^{M-1} \overset{1}{b}_i^3 \overset{2}{B}_i^3} = \frac{1}{2}$$

4-4.3 Transfer functions

Manipulation of the transfer functions allows us to generate different views of the model that we are displaying. This is particularly true in volume rendering. By altering the transfer functions we can highlight or hide different aspects of the data. In this section we show how the re-initialization of NIL maps can be avoided by using a class of transfer functions defined by a basis set.
Equation 4.9 defines the NIL cells:

$$C_i^{\gamma m} = \int_0^1 B_i(t) T(2^\gamma m + 2^\gamma t) dt. \tag{4.17}$$

Where $T()$ is the texture function. Consider the case where we wish to evaluate the NIL map for a transformed representation of this texture. In the following discussion we will use the $\mathcal{T}_\beta()$ to indicate a transfer function that maps a scalar-valued texture $T()$ into another scalar-valued texture $\mathcal{T}_\beta(T())$. The NIL map entries are then computed using a modified version of Equation 4.17:

$$C_{\beta i}^{\gamma m} = \int_0^1 B_i(t) \mathcal{T}_\beta(T(2^\gamma m + 2^\gamma t)) dt \tag{4.18}$$

Following the pattern of NIL maps we can select a set of basis functions that will be used to generate our transfer functions $\mathcal{T}_\beta()$[4]. Using this notation the transfer function is defined by

$$\mathcal{T}_\beta(t) = \sum_{l=0}^{N-1} b_l B_l(t) \tag{4.19}$$

Substituting equation 4.19 into equation 4.18 we get:

$$C_{\beta_i}^{\gamma\ m} = \int_0^1 B_i(t) \sum_{l=0}^{N-1} b_l(\beta) B_l(T(2^\gamma m + 2^\gamma t)) dt \tag{4.20}$$

[4]This does not need to be a basis set, it can be any function set. The space spanned by these functions then defines the transfer functions available to the user.

Noticing that the b_l coefficients are independent of t:

$$\overset{\gamma}{C}{}_{\beta_i}^{\ m} = \sum_{l=0}^{N-1} b_{\beta l} \int_0^1 B_i(t) B_l(T(2^\gamma m + 2^\gamma t)) dt \tag{4.21}$$

These integrals can be pre-computed since they are independent of the transfer function chosen.

$$\overset{\gamma}{D}{}_{il}^{\ m} = \int_0^1 B_i(t) B_l(T(2^\gamma m + 2^\gamma t)) dt \tag{4.22}$$

Remembering the original NIL map equation for a set $\mathcal{S}$ of selected segments

$$I = \sum_{\mathcal{S}} \sum_{i=0}^{M-1} b_i \overset{\gamma}{C}{}_i \tag{4.23}$$

This becomes

$$I_\beta = \sum_{\mathcal{S}} \sum_{i=0}^{M-1} b_i \sum_{l=0}^{N-1} b_{\beta l} \overset{\gamma}{D}{}_{il} \tag{4.24}$$

for the display using transfer function $\mathcal{T}_\beta$

Laur and Hanrahan [Laur91] used a three-dimensional MIP map to speed up the display of volumetric data. Because they compute a MIP map for each of the red, green, blue, and alpha channels they have to recompute the MIP map each time the transfer functions are edited. Using the above technique with M = 1 and N = 3 would require the same amount of memory as the MIP map proposed by Laur and Hanrahan. This corresponds to the use of a one-dimensional basis to approximate the filter and a three-dimensional basis to model the transfer functions.

4-4.4 Procedural textures

NIL maps can be extended to allow the filtering of procedural texture. Consider a class of textures defined by some basis functions $X_0(t), X_1(t), ..., X_{R-1}(t)$. A texture T is then defined by

$$T(t) = \sum_{r=0}^{R-1} x_r X_r(t) \tag{4.25}$$

Recall that

$$\begin{aligned} C_i &= \int_0^1 T(t)B_i(t)dt \\ &= \int_0^1 \sum_{r=0}^{R-1} x_r X_r(t)B_i(t)dt \\ &= \sum_{r=0}^{R-1} x_r \int_0^1 X_r(t)B_i(t)dt \end{aligned}$$

Defining

$$\mathcal{X}_{ri} = \int_0^1 X_r(t)B_i(t)dt$$

The integral I is approximated by

$$I = \frac{\sum_{i=0}^{M-1} b_i \sum_{r=0}^{R-1} x_r \mathcal{X}_{ri}}{S}$$

$$\overset{\gamma}{\mathcal{X}}_{ri} = \int_0^1 X_r(2^\gamma t)B_i(t) \tag{4.26}$$

If the signal T is periodic

$$T(t+1) = T(t)$$

and we need only store one NIL cell per positive level of the NIL map. If negative levels are required then these can be computed as needed.

4-4.5 Cosine textures

In order to show the application of NIL maps to a procedural texture we have chosen to implement a set of procedural textures defined by a truncated cosine series. These textures serve to illustrate the application of NIL maps to procedural textures and also give us a texture set that can be filtered by all four filtering techniques presented in this chapter.

The texture $T(u,v,w)$ is a periodic texture with unit period such that $T(u+n, v+m, w+o) = T(u,v,w)$, with n,m,o being integers. Given a set of texture coefficients

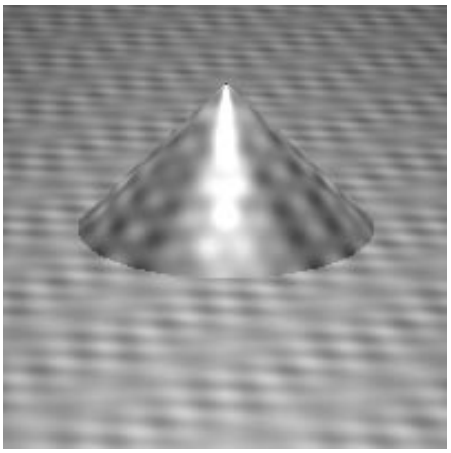
T_{ijk} the texture at a point (u, v, w) is given by

$$T(u,v,z) = S \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} \sum_{k=0}^{K-1} T_{ijk} \cos(2\pi iu) \cos(2\pi jv) \cos(2\pi kw) + 0.5$$

The scale factor S is defined by

$$S = \frac{1}{\sum_{i=0}^{I-1} \sum_{j=0}^{J-1} \sum_{k=0}^{K-1} T_{ijk}}$$

the addition of 0.5 is to ensure that the texture values lie in $[0, 1]$. If these texture coefficients T_{ijk} are obtained by applying the cosine transform on a real data set this 0.5 term in not required. An example of a texture generated in this manner is seen in plate 4.2.



Texture defined by cosine series with I = 4, J = 4, and K = 4

Plate 4.2

4-4.6 Cosine textures with NIL maps

Using these cosine textures with NIL maps results in a succinct analytical definition of the NIL maps. Consider the integral of a texture basis function $\cos(2\pi rt)$ with a polynomial

basis function for the filter $B_i(t) = a_3t^3 + a_2t^2 + a_1t + a_o$. The integral of the product of these two functions $\mathcal{X}_{ri}$ is then

$$\begin{aligned}
\mathcal{X}_{ri} &= \int_0^1 \cos(2\pi rt)\left[a_3t^3 + a_2t^2 + a_1t + a_o\right]dt \\
&= a_3\int_0^1 t^3\cos(2\pi rt)dt + a_2\int_0^1 t^2\cos(2\pi rt)dt + a_1\int_0^1 t\cos(2\pi rt)dt \\
&\quad + a_0\int_0^1 \cos(2\pi rt)dt \\
&= a_3\left[\frac{(12\pi^2r^2t^2-6)\cos(2\pi rt)}{16\pi^4r^4} + \frac{(4\pi^2r^2t^3-6t)\sin(2\pi rt)}{8\pi^3r^3}\right]_0^1 \\
&\quad + a_2\left[\frac{2t\cos(2\pi rt)}{4\pi^2r^2} + \frac{(4\pi^2r^2t^2-2)\sin(2\pi rt)}{8\pi^3r^3}\right]_0^1 \\
&\quad + a_1\left[\frac{\cos(2\pi rt)}{4\pi^2r^2} + \frac{t\sin(2\pi rt)}{2\pi r}\right]_0^1 \\
&\quad + a_0\left[\frac{\sin(2\pi rt)}{2\pi r}\right]_0^1 \\
&= a_3\frac{3}{4\pi^2r^2} + a_2\frac{1}{2\pi^2r^2}
\end{aligned} \tag{4.27}$$

In our implementation of NIL maps we use Lagrange polynomials as the basis of our filters. We chose these polynomials because the points they interpolate are all in the interval $[0, 1]$.

$$\begin{aligned}
B_0(t) &= \frac{-9}{2}t^3 + 9t^2 - \frac{11}{2}t + 1 \\
B_1(t) &= \frac{27}{2}t^3 - \frac{45}{2}t^2 + 9t \\
B_2(t) &= \frac{-27}{2}t^3 + 18t^2 - \frac{9}{2}t \\
B_3(t) &= \frac{9}{2}t^3 - \frac{9}{2}t^2 + t.
\end{aligned} \tag{4.28}$$

With this set of basis functions the NIL map values are:

$$\mathcal{X}_{r0} = \frac{9}{8\pi^2r^2}$$

$$\begin{aligned}\mathcal{X}_{r1} &= \frac{-9}{8\pi^2 r^2} \\ \mathcal{X}_{r2} &= \frac{-9}{8\pi^2 r^2} \\ \mathcal{X}_{r3} &= \frac{9}{8\pi^2 r^2}\end{aligned}$$

Incorporating the 2^γ factor into the above equations gives a definition of the positive levels of the NIL map.

$$\begin{aligned}\overset{\gamma}{\mathcal{X}}_{r0} &= \frac{9}{8\pi^2 (2^\gamma r)^2} \\ \overset{\gamma}{\mathcal{X}}_{r1} &= \frac{-9}{8\pi^2 (2^\gamma r)^2} \\ \overset{\gamma}{\mathcal{X}}_{r2} &= \frac{-9}{8\pi^2 (2^\gamma r)^2} \\ \overset{\gamma}{\mathcal{X}}_{r3} &= \frac{9}{8\pi^2 (2^\gamma r)^2}\end{aligned}$$

Other basis functions can be used for NIL maps, in particular the Catmull-Rom cubic basis integrated with the cosine basis yields:

$$\begin{aligned}\overset{\gamma}{\mathcal{X}}_{r0} &= \frac{1}{8\pi^2 (2^\gamma r)^2} \\ \overset{\gamma}{\mathcal{X}}_{r1} &= \frac{-1}{8\pi^2 (2^\gamma r)^2} \\ \overset{\gamma}{\mathcal{X}}_{r2} &= \frac{-1}{8\pi^2 (2^\gamma r)^2} \\ \overset{\gamma}{\mathcal{X}}_{r3} &= \frac{1}{8\pi^2 (2^\gamma r)^2}\end{aligned}$$

For a particular texture defined by $\{X_0, X_1, X_2, ..., X_{R-1}\}$ we can compute the NIL map $\overset{\gamma}{C}_i$. The periodicity of this texture ensures that we only have to store a single NIL cell per level. For most applications we have found that a NIL map with depth $\gamma_{\max} = 10$ is sufficient. If any filter covers more than 2^{10} texels then using the DC, or average, component of the texture is appropriate.

The negative NIL levels for these textures are also defined analytically. Define for

convenience the following functions

$$\begin{aligned}
\overset{\gamma}{F}{}_{3}^{m}(t) &= \int t^3 \cos(2\pi\,(r2^\gamma)\,(t+m))dt \\
&= \frac{(12\pi^2\,(r2^\gamma)^2\,t^2-6)\cos(2\pi\,(r2^\gamma)\,(t+m))}{16\pi^4\,(r2^\gamma)^4} \\
&\quad + \frac{(4\pi^2\,(r2^\gamma)^2\,t^3-6t)\sin(2\pi\,(r2^\gamma)\,(t+m))}{8\pi^3\,(r2^\gamma)^3} \\
\overset{\gamma}{F}{}_{2}^{m}(t) &= \int t^2 \cos(2\pi\,(r2^\gamma)\,(t+m))dt \\
&= \frac{2t\cos(2\pi\,(r2^\gamma)\,(t+m))}{4\pi^2\,(r2^\gamma)^2} + \frac{(4\pi^2\,(r2^\gamma)^2\,t^2-2)\sin(2\pi\,(r2^\gamma)\,(t+m))}{8\pi^3\,(r2^\gamma)^3} \\
\overset{\gamma}{F}{}_{1}^{m}(t) &= \int t\cos(2\pi\,(r2^\gamma)\,(t+m))dt \\
&= \frac{\cos(2\pi\,(r2^\gamma)\,(t+m))}{4\pi^2\,(r2^\gamma)^2} + \frac{t\sin(2\pi\,(r2^\gamma)\,(t+m))}{2\pi\,(r2^\gamma)} \\
\overset{\gamma}{F}{}_{0}^{m}(t) &= \int \cos(2\pi\,(r2^\gamma)\,(t+m))dt \\
&= \frac{\sin(2\pi\,(r2^\gamma)\,(t+m))}{2\pi\,(r2^\gamma)}
\end{aligned}$$

The negative levels of the nil map are then defined by

$$\overset{\gamma}{\mathcal{X}}{}_{ri}^{m} = a_i[\overset{\gamma}{F}{}_{i}^{m}(t)]_0^1$$

This allows us to define arbitrarily deep NIL maps. Pre-computation of these levels incurs a heavy memory cost, but because they are procedurally defined and the cost of evaluating these NIL cells is roughly comparable to the cost of evaluating the filter basis functions it makes sense to evaluate these NIL cells as they are needed. If enough memory is available, these may be stored in case they are needed later on.

Using this definition for NIL maps we can define a general NIL map for a class of textures defined by a basis set $\{X_i\}$. Once a particular texture has been defined the generality of these NIL maps is no longer needed. Thus evaluating the $\overset{\gamma}{C}{}_{i}$ coefficients can be done using the $\overset{\gamma}{\mathcal{X}}{}_{ri}^{m}$ coefficients.

4-4.7 Three dimensions

The extension of NIL maps to three dimensions is straightforward. The resulting NIL cells are $\overset{\gamma}{C}{}_{ijkl}^{mno}$, $\overset{\gamma}{D}{}_{ijkl}^{mno}$, $\overset{\gamma}{\mathcal{X}}{}_{r_u r_v r_w ijk}$, and S.

$$\overset{\gamma}{C}{}_{ijk}^{mno} = \int_0^1 \int_0^1 \int_0^1 B_i(x)B_j(y)B_k(z)T(2^\gamma(m+x), 2^\gamma(n+y), 2^\gamma(o+z))dxdydz \quad (4.29)$$

$$\overset{\gamma}{D}{}_{ijkl}^{mno} = \int_0^1 \int_0^1 \int_0^1 B_i(x)B_j(y)B_k(z)B_l(\mathcal{T}(2^\gamma(m+x), 2^\gamma(n+y), 2^\gamma(o+z)))dxdydz \quad (4.30)$$

$$\overset{\gamma}{\mathcal{X}}{}_{r_u r_v r_w ijk} = \int_0^1 \int_0^1 \int_0^1 X_{r_u}(2^\gamma u)X_{r_v}(2^\gamma v)X_{r_w}(2^\gamma w)B_i(u)B_j(v)B_k(w)dudvdw \quad (4.31)$$

or

$$\begin{aligned} \overset{\gamma}{\mathcal{X}}{}_{r_u r_v r_w ijk} &= \int_0^1 X_{r_u}(2^\gamma u)B_i(u)du \int_0^1 X_{r_v}(2^\gamma v)B_j(v)dv \int_0^1 X_{r_w}(2^\gamma w)B_k(w)dw \\ &= \overset{\gamma}{\mathcal{X}}{}_{r_u i}\, \overset{\gamma}{\mathcal{X}}{}_{r_v j}\, \overset{\gamma}{\mathcal{X}}{}_{r_w k} \end{aligned}$$

The scaling variable S is

$$S = \sum_{abc} \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} \sum_{k=0}^{M-1} 2^{\gamma_{max}-\gamma} b_{ijk}^{abc} B_{ijk} \quad (4.32)$$

with

$$B_{ijk} = \int_0^1 \int_0^1 \int_0^1 B_i(u)B_j(v)B_k(w)dudvdw \quad (4.33)$$

4-4.8 Trigger points

As discussed previously, the placement of trigger points is a critical part of the NIL map technique. This is particularly true when we are considering issues of practical efficiency. In order to study the usefulness of NIL maps in both volume rendering and texture mapping we have chosen to use a fairly simple algorithm for laying down the trigger

points. In a particular application where we have a more restricted set of filters that we wish to approximate we can use the properties of these filters to guide the placement of the trigger points. This possibility was pointed out by Fournier and Fiume and was explored by Lansdale in his M.Sc. thesis [Lans91].

Trigger points for texture mapping

The direct evaluation filter that we presented earlier provides a volume in which to place the trigger points. The user controls the number of trigger points that are uniformly distributed throughout this volume. We have found that using a large number of trigger points (on the order of $64 = 4^3$) and setting a fairly large tolerance for subdivision (on the order of 4-8) produces a better hierarchy for approximating the filter than when we use a small number of trigger points and a low tolerance for subdivision. In the next chapter we illustrate the flexibility of NIL maps by showing how to model a filter that simulates motion blur by using NIL maps. This motion blur is implemented by modifying the trigger placement code and by defining the appropriate filter. The code for both the trigger placement and the filter function definition are presented in Appendix B.

Trigger points for volume rendering

The distribution of trigger points throughout the volume extruded from the pixel uses two user defined variables, width and steps. The width parameter determines the number of trigger points to be placed per slice. The step parameter determines the number of slices to be placed through a unit length of the volume. Thus the number of trigger points placed for a particular pixel is $\text{width}^2 \times \text{steps} \times \text{length}$ and the maximum number of trigger points placed is $\text{width}^2 \times \text{steps} \times \sqrt{3}$.

Again we have found that using a larger number of trigger points and a higher value for the subdivision tolerance induces a better hierarchy with which to approximate the filter.

4-5 Wrapup

In this chapter we have presented four filtering techniques, clamping, direct evaluation, EWA, and NIL maps. Each of these techniques has its strengths and weaknesses. An evaluation of their characteristics is presented in the next chapter.

Chapter 5

Comparison and application of the techniques

Si operatur, operatur.
Bob Lewis

The four filter evaluation techniques presented in the previous chapter have applications to three-dimensional texture mapping. Each of these techniques has its strengths and weaknesses. In order to use a particular technique we must be able to assess its usefulness. Because these techniques are quite different it would be overly simplistic to use a single evaluation criterion. In this chapter we present an evaluation scheme consisting of several criteria. It is our hope that the results will allow the best filter approximating technique to be chosen.

5-1 Three-dimensional texture filter comparison

Mitchell and Netravali [Mitc88] presented a study of a class of cubic filters defined by two parameters. Their evaluation criteria used visual characteristics (ringing,blurring, anisotropy) to measure the performance of this family of reconstruction filters. They mentioned that that it is difficult to come up with objective measures for filters. Rather, they suggested that the properties of filters should be analyzed, documented, and made available so that the user can select a filter depending on the application. In some sense the same can be said about the evaluation of filter approximating techniques. In order to choose a filter technique we must be familiar with its strengths and weaknesses. We can use image fidelity measures to evaluate how well the technique approximates a particular filter but there are other non-quantitative measures that must also be considered. For the evaluation and comparison of the filter techniques we use the following criteria.

5-2 Filter technique evaluation criteria

- Class of filters:
  This criterion is intended to provide a measure of the application of the technique. There are three main issues that will be addressed by this criterion:
  - What kind of filters can approximated with the technique? e.g. A Gaussian filter projected into texture space, box filter projected into texture space.
  - What control over the approximation quality is provided by the technique?
  - Does the technique handle the scaling of filters? Most of the filters used in computer graphics have a finite support. How does the performance of the filter depend on the number of voxels or texels that are covered by the filter?
- Class of textures:
  This criterion studies the textures to which the technique is applicable. There are three classes of three-dimensional textures that we wish to check the techniques against.
  - General procedural:
    A procedural texture with no restrictions on the functions from which it is constructed.
  - Procedural defined by a basis:
    The basis may be further restricted to a frequency basis. Any restrictions imposed on this basis by the technique are also explored.
  - Discrete:
    A texture defined by a volumetric sample set. These textures may be empirically or procedurally defined.
- Fidelity measures:
  The image compression literature uses several fidelity measures for image data. The two main measures used are the mean square error (MSE), and the signal to noise ratio (SNR). In an overview of the compression literature Jain [Jain81] defines these two terms. The MSE of two images $\{u_{i,j}\}$ and $\{u^*_{i,j}\}$ of resolution $N \times M$ is

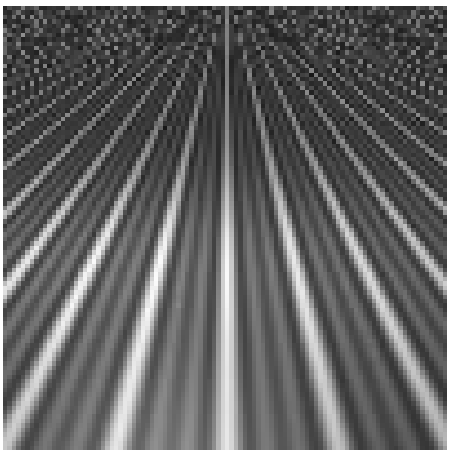
defined as

$$e_{ms}^2 = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}(u_{i,j} - u_{i,j}^*)^2.$$

The signal to noise ratio for an image whose pixel values range from 0 to 255 is then defined in decibels by

$$SNR = 10\log_{10}\frac{(255)^2}{e_{ms}^2}(db).$$

A quick scan of recent literature in the field of image compression [Wu91, Mark91, Xue91, Pent91, Tilt91] confirms that these two error metrics are still being used. Slight variations of these measures can be found in [Prat78].



Point sampled rendering of the cosine texture used for evaluation purposes. This image exhibits a large amount of aliasing in the upper portion.

Plate 5.1

We will attempt to evaluate the performance of the clamping, EWA, and NIL map techniques using the direct evaluation filters as a comparison. Clamping will be compared with the direct evaluation of a box filter, EWA filters and NIL maps will be compared with the direct evaluation of a Gaussian filter. The image that will be

used for these comparisons is presented in Plate 5.1. The resolution of the image was chosen to be 100 by 100 because this increases the pixel size relative to the scene, thus increasing the aliasing artifacts.

- Visual quality:
  Unfortunately the computational metrics used above are not directly related to our perception of the images so we must also perform a subjective evaluation. We will perform this evaluation using the following criteria.

  - Directional artifacts:
    Does the technique introduce any directional artifacts?
  - Discontinuities:
    Does the technique introduce any discontinuities to the texture?

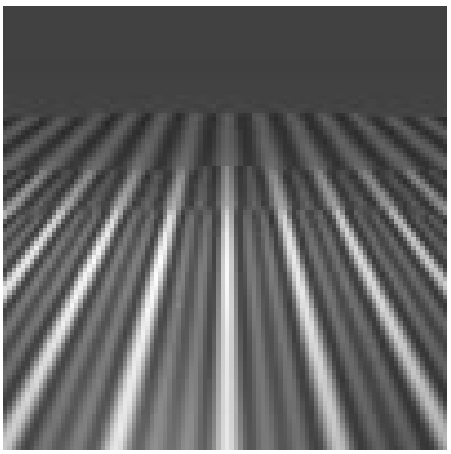
- Cost of the technique:

  - Pre-processing cost: The cost of any required pre-processing.
  - Evaluation cost: The cost of evaluating one filter application.

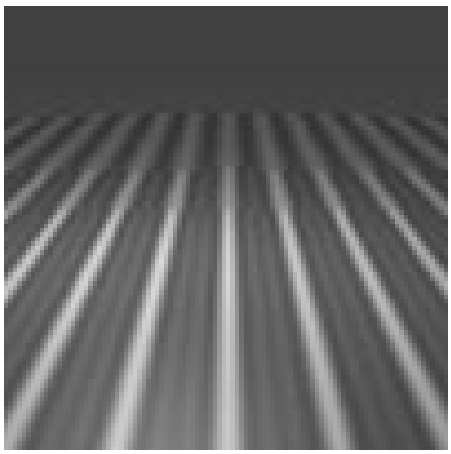
5-3 Clamping

- Class of filters:

The original clamping technique [Nort82] focused on finding a quick approximation to a box filter. They achieved this by requiring that the frequency components of the texture be known. Once the frequency components for a texture are known the class of filters that can be approximated with this technique is quite large. The only requirement placed on the filters to be approximated by such a clamping technique is that they have a well defined Fourier transform. The filtering quality is then controlled by the order of the polynomial that is used for the approximation of the transformed filter.



Step function used for clamping.
Plate 5.2



Quadratic function used for clamping.
Plate 5.3

- Class of textures:

  Procedural textures for which the frequency distribution of the signal is known. This does not mean that we must know the exact Fourier distribution of the signal, but we must have a good idea of the frequency distribution of the signal and how these frequency components are put together. This was the method used by Perlin [Perl85] for anti-aliasing his turbulence function. He modeled turbulence by adding successively higher frequency band-limited noise functions to the signal. By using the size of the projection of the pixel the computation could be stopped when it was determined that the next band-limited noise function would cause aliasing.

- Fidelity measures:
  The clamping methods we evaluated were compared to a direct evaluation of the box filter. The box filter was computed using the direct evaluation technique with an error tolerance of $0.004 \approx \frac{1}{256}$.

- Visual quality:
  Two particular implementations of this technique were studied, the first a direct

extension of the original quadratic clamping technique as suggested by Norton et al. [Nort82], the second a step function as implemented by Perlin [Perl85]. As can be seen from the image in Plate 5.2 the use of a step function for clamping results in visible discontinuities in the image. Using the quadratic function (Plate 5.3) reduces the effect of the discontinuity, however, the discontinuities are still visible.

- Cost of the technique:

  - Pre-processing cost:
    There is no pre-processing required if the frequency information of the texture is known. If this information is not available the cost of pre-processing is the cost of finding the frequency information. For a discrete texture of size N^3 this would incur a cost of $O(N^3 \log N)$ if a fast Fourier transform or some other related transform is used.

  - Evaluation cost:
    The computation of a texture defined by its frequency spectrum requires the weighed sum of the appropriate basis functions. In the case of the cosine textures the computation is

$$T(u,v,z) = S \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} \sum_{k=0}^{K-1} T_{ijk} X_i(u) X_j(v) X_k(w)$$

    The introduction of the above computation to include the clamping function results in

$$T(u,v,z) = S \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} \sum_{k=0}^{K-1} T_{ijk} C(i,j,k) T_{ijk} X_i(u) X_j(v) X_k(w)$$

    where $C(i,j,k)$ is the clamping function. The additional cost incurred by the technique is then $I \times J \times K$ invocations of the clamping function. Thus the total cost for evaluating the clamped texture is

$$I \times J \times K(\text{cost}(X_i) + \text{cost}(clamp)).$$

When the cost of evaluating the texture basis functions is higher than the cost of evaluating the clamping function the cost incurred by clamping is low. For cosine textures the ratio of the cost of the cosine function to the cost of the clamping function ranges between 26:1 and 6.6:1[1]. If the clamping function is known to be zero above a certain frequency then this can be used to ensure that no unnecessary basis functions are evaluated. In most situations this removal of higher frequency components results in a decrease in computation cost that outweighs the cost of evaluating the clamping function. In Table 5.1 we present the timings for evaluating the test image using point sampling, step clamping and quadratic clamping. The third and fourth columns of this table contain the MSE andSNR of the image relative to the image that results from a direct evaluation of the box filter.

Even though we could argue that the quadratic clamping function produces a better picture, its MSE is higher than that of the image produced by the step clamping function. This is not surprising because the quadratic filter starts clamping down on a frequency component long before it needs to be filtered out. This excessive filtering causes the rise in the MSE.

If the textures are defined in terms of their Fourier spectra and the filtering required is simple, anti-aliasing this technique should be considered. The discontinuities presented in this example illustrate the problems of choosing an inadequate clamping function. By tailoring a clamping function to a particular application it is possible to use this technique quite effectively.

5-4 Direct evaluation

- Class of filters:

  Arbitrary filters defined over rectangular parallelepipeds[2]. This restriction is in

[1] On a Silicon Graphics 300 series workstation the ratio is 20:1. On a SUN SPARCstation SLC the ratio is 26:1. On a IBM RS/6000 560 the ratio is 6.6:1.

[2] It is possible to use quadrature rules to evaluate integrals over non-rectangular volumes. Details of this can be found in [Burd81]. The construction of these non-rectangular integration volumes might require that specialized code be developed for different filters.

Technique	Time (min:sec)	MSE	SNR
Point sampled	0:11	115	27.5
Step clamped	0:8	115	27.5
Quadratic clamped	0:9	224	24.6
Direct evaluation	13:29	–	–

MSE and SNR of clamping methods

Table 5.1

place because of the quadrature methods used. Because there is no restriction on the size of these rectangular parallelepipeds it is possible to compute complex filters by extending the volume of integration until it encloses all of the *interesting* parts of the filter. Even though this approach is costly it does provide us with a method for evaluating these filters with some degree of confidence. The images computed in this manner can then be used to test the accuracy of other filtering techniques.

- Class of textures:
  The direct evaluation of filters using quadrature rules imposes no restriction on the class of textures that can be filtered.

- Fidelity measures:
  Because the images computed with the adaptive quadrature rule are being used as a measure of the goodness of the other techniques, we computed these images using a tolerance of $0.004 \approx \frac{1}{255}$. Using this tolerance and Simpson's quadrature rule we computed the image using the box filter and the Gaussian filter. In addition to this, one image was computed using the Gaussian quadrature rule with a Gaussian filter.

  In Table 5.2 we present a comprehensive set of data resulting from the application of the different filtering techniques to our test image.

- Visual quality:

	Technique	Time (min:sec)	MSE	SNR (dB)
1	Point sampled	0:11	115(4)	27.5
2	Step clamped	0:8	115(4)	27.5
3	Quadratic clamped	0:9	224	24.6
4	Direct Simpson's (box)	13:29	–	–
5	Direct Gaussian (box)	3:51	2.18(4)	44.7
6	Direct Simpson's (Gauss)	13:31	–	–
7	NIL, M=1,Tol=10	1:04	6416(6)	10.2
8	NIL, M=1,Tol=5	1:38	255(6)	24.0
9	NIL, M=1,Tol=2	1:12	165(6)	26.0
10	NIL, M=4,Tol=10	5:57	238 (6)	24.3
11	NIL, M=4,Tol=5	13:35	138 (6)	26.7
12	NIL, M=4,Tol=2	17:42	117 (6)	27.5
13	EWA 3d (3)	1:33	196(6)	25.2
14	EWA 3d (5)	4:47	193(6)	25.3
15	EWA 3d (7)	8:38	193(6)	25.3
16	EWA 3d (11)	24:18	192(6)	25.3

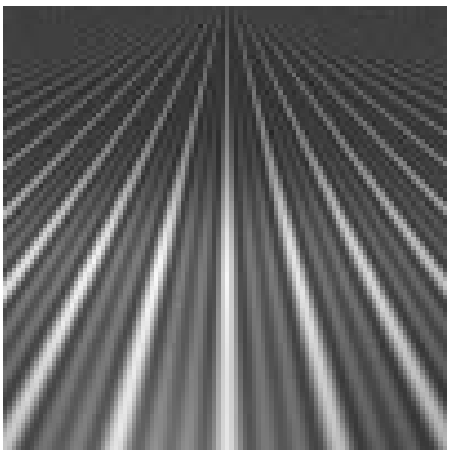
Comparison of run times for filtering techniques. The parenthesized numbers in the fourth column indicate which technique was used to compute the MSE and SNR.

Table 5.2

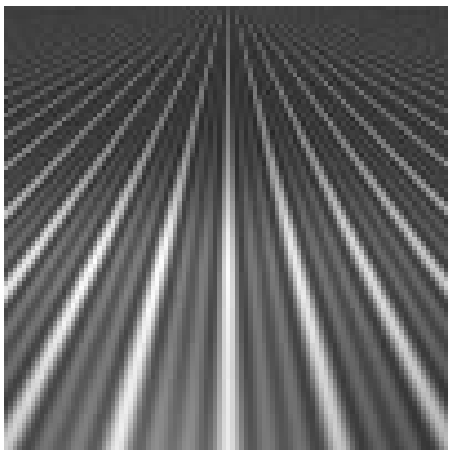
By using an adaptive quadrature rule we can directly evaluate the effects of using different filters on a particular image. Plates 5.5 and 5.4 show the results of using a box filter and a Gaussian filterfilter!Gaussian on our test image.

In Plate 5.6 we present the results of three classes of filters being applied to a marble block. The first column shows the results obtained with box filters aligned with the surface of the object. In the second column Bartlett filters are applied to the same object. The lower images were generated with increasingly wider filters. The last column shows the results of using a box filter aligned with the ray. In this column it is the depth of the filter that is being increased.

- Cost of the technique:



Direct evaluation of Box filter
Plate 5.4



Direct evaluation of Gaussian filter
Plate 5.5

– Pre-processing cost:

  The technique itself does not require any pre-processing of the texture. If the quadrature rule chosen requires any pre-processing this will be the total pre-processing cost required. In both the case of Simpson's adaptive quadrature and Gaussian quadrature no pre-processing is required.

– Evaluation cost:

  The evaluation cost is dependent on the quadrature rule chosen.

  With Simpson's adaptive quadrature rule used in all three dimensions the minimum number of texture evaluations is 125 ($5\times5\times5$). The number of actual evaluations performed depends on the fourth derivative of the product of the filter with the texture. This dependence exists because Simpson's adaptive rule stops when its estimate of the fourth derivative falls below a predetermined tolerance.

  Gaussian quadrature provides the best approximation to the integral for a fixed number of function evaluations[3]. In our implementation we used the cubic

[3]Gaussian quadrature optimizes the evaluation of the integral by carefully choosing the position of the sample points. The Gaussian quadrature rule chosen here is the one which results from using the Legendre polynomials

Gaussian quadrature rule. In this case the number of texture evaluations is 27 $= (3 \times 3 \times 3)$. A direct comparison of the run times of the various algorithms is presented in Table 5.2.

This technique allows the accurate computation of arbitrary filters. By using an adaptive quadrature rule we ensure that the computation of the filter is performed to a pre-determined tolerance. This expensive computation allows us to generate high quality images when expense is not a concern. In the study of other filtering techniques we have found this technique useful as a means of comparison. It is also possible to use a fixed cost quadrature rule for the evaluation of these filters. Because in some sense the Gaussian quadrature rule is *optimal* it makes sense to use this *super-sampling* technique rather than other super-sampling techniques.

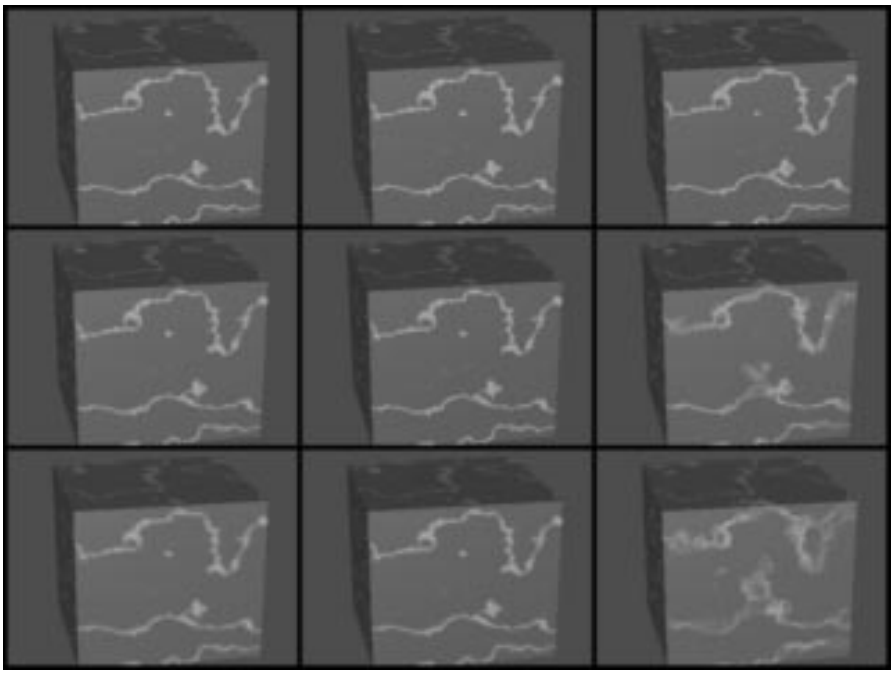
5-5 EWA filters

- Class of filters:
  Discrete approximations to truncated two-dimensional and three-dimensional radially symmetric filters. Greene and Heckbert [Gree86] used the Gaussian filter in their presentation of the EWA technique. The extensions presented here approximate a two-dimensional Gaussian filter applied on the tangent plane of the object, and a three-dimensional Gaussian applied in a volume near the surface of the object.

- Class of textures:
  All textures, whether procedural or discrete.

- Fidelity measures:
  In Table 5.4 we present the cost and resulting quality of using the EWA filtering technique. In order to illustrate the cost of this technique we present the results of using this technique with a maximum of 3^3,5^3,7^3, and 11^3 sample points. As can be seen from the results in Table 5.4, increasing the sampling of this technique does not significantly improve its performance.

[Burd81].



Marble blocks, with box, Bartlett, and volume filters.

Plate 5.6

Box filter	Bartlett filter	Volume filter
$\epsilon = 0.01$ $spread = 1.0$	$\epsilon = 0.01$ $spread = 1.0$	$\epsilon = 0.025$ $spread = 1.0$
$\epsilon = 0.01$ $spread = 2.0$	$\epsilon = 0.01$ $spread = 2.0$	$\epsilon = 0.100$ $spread = 1.0$
$\epsilon = 0.01$ $spread = 4.0$	$\epsilon = 0.01$ $spread = 4.0$	$\epsilon = 0.200$ $spread = 1.0$

Parameters for the displays of the marble block in Plate 5.6

Table 5.3

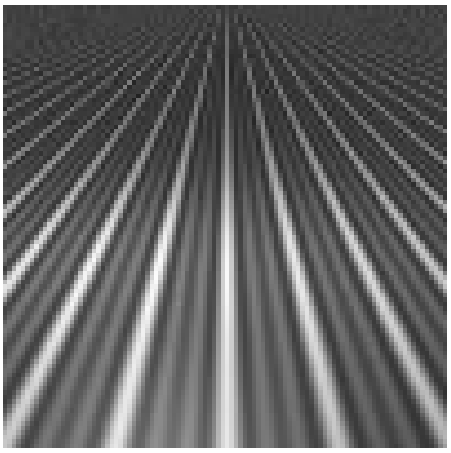
Technique	Time (min:sec)	MSE	SNR (dB)
Point sampled	0:11	115(5)	27.5
EWA 3d (3)	1:33	196(6)	25.2
EWA 3d (5)	4:47	193(6)	25.3
EWA 3d (7)	8:38	193(6)	25.3
EWA 3d (11)	24:18	192(6)	25.3
Direct Simpson's (box)	13:29	–	–

EWA times and performances

Table 5.4

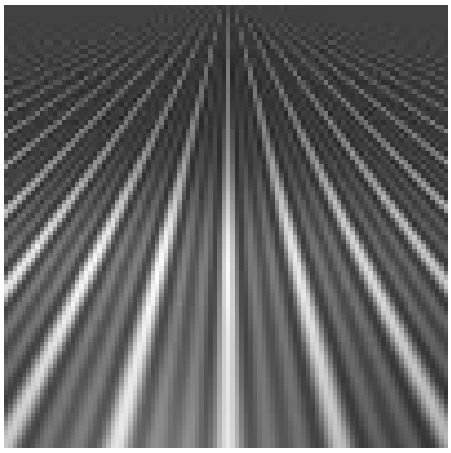
- Visual quality:

  In areas of the image where the sampling rate is insufficient there is still some aliasing.



EWA filter. Max samples = 3

Plate 5.7



EWA filter. Max samples = 11

Plate 5.8

- Cost of the technique:

  – Pre-processing cost:

    Independent of texture size or class. The cost of the pre-processing is n evaluations of the filter and n memory locations for the results, where n is the size of the linear array being used to store the pre-computed filter values.

Technique	Pre-processing time (min:sec)
NIL M=4	0:6.8
NIL M=1	0:0.1

NIL map pre-processing times for sample cosine texture ($4 \times 4 \times 4$).

Table 5.5

– Evaluation cost:

The cost of evaluating an EWA sample is $\Omega(n^3)$, where n is the number of samples taken along one of the axes of the ellipsoid.

Given a box which encloses the ellipsoid centered at p_o we compute the number of samples, n^3, which must be taken in this box. The result is then computed as a weighted sum of the texture over these n^3 points. Thus the cost of a single evaluation of an EWA filter is $\Omega(n^3)$.

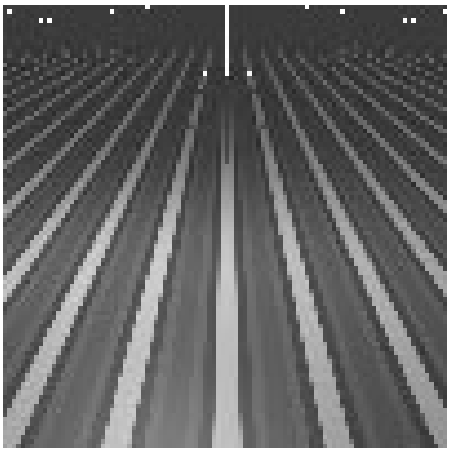
The use of a pre-computed filter for the evaluation of these radially symmetric functions removes the cost of approximating the filters at each step. This approximation of the filter performs well when the volume over which the filter is being evaluated is small. When the filter volume is large the number of samples required makes this approximation very costly. This high cost requires us to limit the number of samples taken per filter. The simplicity of the technique makes it attractive, however, this simplicity comes at the price of restricting the class of filters that can be used.

5-6 NIL maps

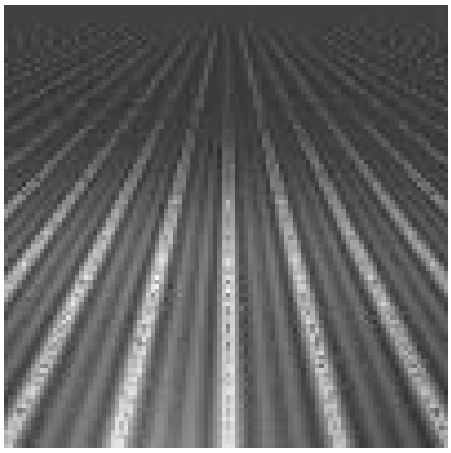
- Class of filters:
  Arbitrary filters. Approximation quality depends both on the order of the approximating patches and the number of patches chosen to represent the filter.

- Class of textures:
  Procedural textures defined by a set of basis functions or discrete textures.

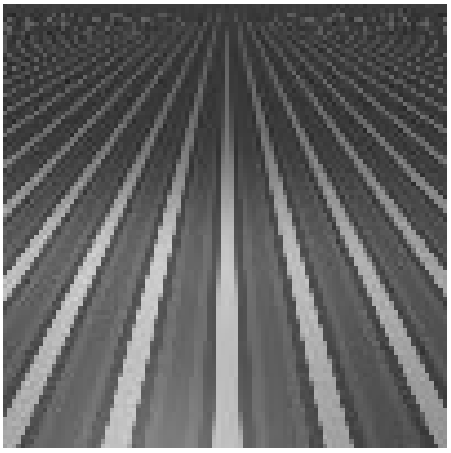
- Fidelity measures:
  The results of approximating a Gaussian filter with NIL maps are presented in Table 5.6. In order to give an overview of the performance of the performance of NIL maps we present several images. The first three were computed using constant patch NIL maps (M=1), and the remaining three images were computed with tricubic patches (M=4). The number of trigger points for all of the images is constant (5^3). The number of trigger points allowed per NIL cell is shown in column 2.



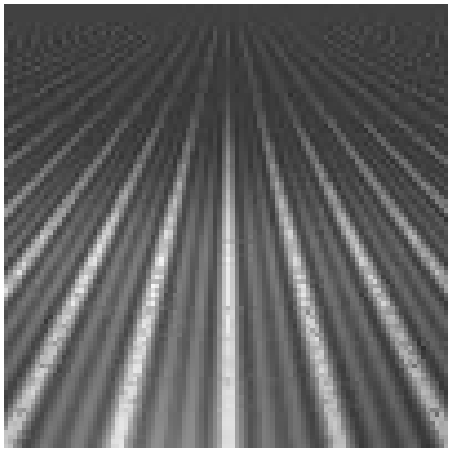
NIL map filter. M=1, tol=5
Plate 5.9



NIL map filter. M=4, tol=5
Plate 5.10



NIL map filter. M=1, tol=2
Plate 5.11



NIL map filter. M=4, tol=2
Plate 5.12

Technique	Tolerance	Time (min:sec)	MSE	SNR (dB)
Point sampled	–	0:15.2		
NIL, M=1	10	1:04	6416(4)	10.2
NIL, M=1	5	1:38	255(4)	24.0
NIL, M=1	2	1:12	165(4)	26.0
NIL, M=4	10	5:57	238 (6)	24.3
NIL, M=4	5	13:35	138 (6)	26.7
NIL, M=4	2	17:42	117 (6)	27.5
Direct Simpson's (Gauss)	–	13:31	–	–

NIL times and comparisons.
Table 5.6

- Visual quality:

  A number of anomalies were seen in the images computed using NIL maps. Many of these are attributable to an inadequate sampling of the filter function. When many of the control point weights are small the scaling value is also small. Small variations in this value can cause large variations in the computed value because the scaling factor is the denominator. An extreme example of this effect can be seen in Plate 5.9, where the scaling factor evaluated to zero in a few spots (the white pixels). By forcing the subdivision to proceed further these gross artifacts are removed. This is shown in Plate 5.11.

  When the one-dimensional, or constant, NIL maps are used a blockiness is introduced. This is not completely unexpected because we are approximating the filter with constant patches.

- Technique Cost:

  - Pre-processing cost:

    For textures defined by a set of basis functions the storage cost for each positive level is M^3R^3 . For a particular texture defined this can be reduced to M^3 by computing the $\overset{\gamma}{C}_{ijk}$ terms.

For a discrete texture with resolution $x = y = z = 2^{\gamma_{max}}$ the storage cost per positive level of the NIL map is $2^{3(\gamma_{max} - gamma)} M^3$. Thus the total cost for storing the positive levels of the NIL map is $M^3 2^{3(\gamma_{max})} \sum_{i=0}^{\gamma_{max}} \frac{1}{2^{3i}}$.

– Evaluation cost:

In order to approximate a filter with NIL maps two steps must be followed. The first generated the hierarchy and the second evaluates the approximation to the filter by computing the weights for the control points of the patches.

The number of patches which can be generated for an approximation depends on the number of trigger points N_t, the tolerance tol, and the minimum depth γ_{min}. In fact the cost of generating the hierarchy is $O\left(\frac{N_t}{tol+1}\right)$ and the size of the hierarchy is also $O\left(\frac{N_t}{tol+1}\right)$.

The proof of this is a simple extension of the proof presented by Fournier and Fiume [Four88a] in their original NIL map paper.

The smallest number of patches for an approximating hierarchy will be generated when $tol > N_t$ and all of the trigger points lie in one NIL cell.

The largest number of patches which can be produced occurs when the subdivision starts at γ_{max} and proceeds to the lowest level γ_{min}. In order for this to occur we must have $tol + 1$ trigger points clustered together in such a way that they force a subdivision at each level. Because each subdivision caused by this of cluster trigger points removes one patch and adds eight patches the number of patches generated by this control patch is at most $7(\gamma_{max} - \gamma_{min})$. Notice that no other set of $tol + 1$ trigger points can add as many patches to the approximating hierarchy, because the first subdivision is already attributed to the first $tol + 1$ trigger points. We can thus use $7(\gamma_{max} - \gamma_{min})$ as a coarse upper bound for the number of patches which each $tol + 1$ trigger points will generate. This results in an upper bound of

$$7(\gamma_{max} - \gamma_{min}) \left(\frac{N_t}{tol + 1}\right)$$

patches for the approximating hierarchy. Thus the number of patches approximating a filter is $O\left(\frac{N_t}{tol+1}\right)$.

The cost of constructing this hierarchy is also $O\left(\frac{N_t}{tol+1}\right)$. This follows because each patch produced in the above process is only executed once.

The approximation of the filter is then computed using the following sum.

$$I = \sum_{patch} \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} \sum_{k=0}^{M-1} b_{ijk}^{patch} C_{ijk}^{patch}$$

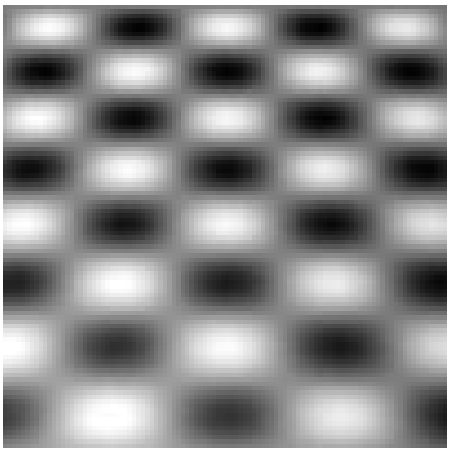
The cost of evaluating the filter is then $O\left(\frac{N_t}{tol+1} M^3\right)$.

Thus we see that the cost of evaluating a single filter with NIL maps in not dependent on the filter's size, position, or shape, but rather is dependent on tolerance parameters set by the user.

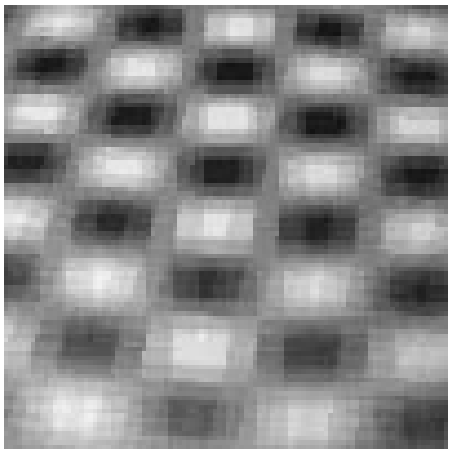
One could argue that the use of NIL maps for the approximation of a Gaussian filter is somewhat excessive. A better example of the power of NIL maps is the case of motion blur filters.

First let us assume that a Gaussian filter is being used as the filter over the area on the viewing plane which maps onto a pixel. If the texture is moving across the screen in a linear fashion then the required filter is a Gaussian extruded along a line. If the texture is on a plane which is rotating about the origin then the filter can be approximated by a Gaussian stretched along the corresponding arc. In order to model this filter with NIL maps we lay down the trigger near this arc. The hierarchy is found using these trigger points. The weights for the control points are taken from the rotated filter. The implementation of this filter required that less than 50 lines of C code be written. Appendix B shows the code written for this example.

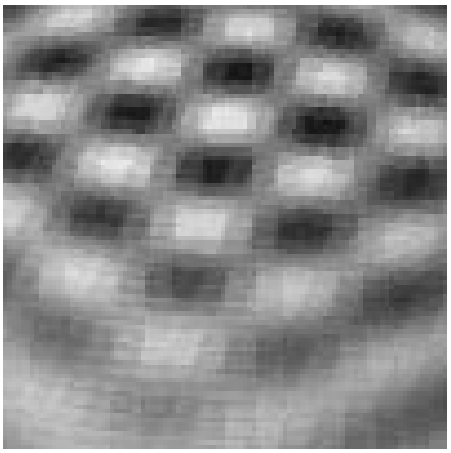
In Plate 5.13 we present a point-sampled view of a textured plane. If this texture is rotated about the origin and the exposure time of a camera is large enough motion blur will occur. This motion blur occurs because the texture moves relative to the viewing plane. Motion blur can be modeled by spreading a filter over the area of the texture which passed in front of a pixel area. Using NIL maps we were able to quickly model and implement such a motion blur filter. In Plates 5.14, 5.15, and 5.16 we see the plane displayed with motion blur filters. The rotation used in these Plates is $\pi/24$, $\pi/12$, and $\pi/6$ respectively.



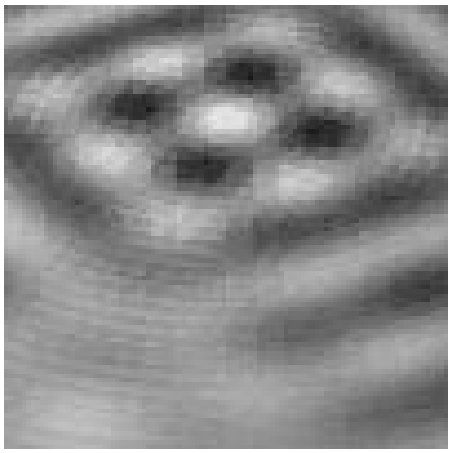
Texture for motion blur example
Plate 5.13



Motion blur caused by a rotation of $\pi/24$
Plate 5.14



Motion blur caused by a rotation of $\pi/12$
Plate 5.15



Motion blur caused by a rotation of $\pi/6$
Plate 5.16

5-7 Wrapup

The four filtering techniques compared here allow the evaluation of three-dimensional filters. Each of these techniques has its strengths and weaknesses. In Tables 5.7 - 5.11 we present a summary of our evaluation.

Clamping provides the lowest cost method for the removal of aliasing frequencies from an image. The artifacts which this technique can introduce and the requirement that the texture be expressed in terms of its frequency spectrum make this technique somewhat limited in application. EWA filters allow the evaluation of radially symmetric filters without the overhead of computing an expensive filter. When this technique is used with expensive textures the high cost of evaluating large filters can force the user to place an upper bound on the number of samples taken. If this is the case, EWA filters will not perform well when the filters grow large. By using a pyramidal data structure NIL maps provide a constant cost (per pixel) filter evaluation technique. The quality of the approximation is controlled by the order of the approximating basis and the number of patches generated to approximate the filter. Even though the per-pixel cost of NIL maps is fixed it is still quite high. This makes NIL maps impractical for the evaluation of simple filters such as elliptical Gaussians. However, the ease with which new filters can be developed makes NIL maps a powerful tool in the study of filters for three-dimensional texture mapping. As expected the direct evaluation of the filters using adaptive quadrature rules yield the best results. The evaluation of odd shaped filters (such as motion blur) may required the evaluation of the integrals to be over large volumes or over irregular volumes. Computing these integrals is expensive, but the control over the quality afforded by this technique may make it the only option in some applications.

Technique	Filter class
Clamping	Polynomial approximation in Fourier space.
Direct	Arbitrary filters over rectangular volumes.
EWA-2 fixed EWA-2 adaptive EWA-3 fixed EWA-3 adaptive	Radially symmetric filters, centered in ellipse or ellipsoid.
NIL Maps	Arbitrary filters approximated by patches.

Filters approximated

Table 5.7

Technique	Texture class
Clamping	Procedural with frequency information. Discrete with frequency information.
Direct	Procedural and discrete.
EWA-2 fixed EWA-2 adaptive EWA-3 fixed EWA-3 adaptive	Procedural and discrete.
NIL Maps	Procedural defined by a set of basis functions.

Texture class allowed

Table 5.8

Technique	Pre-processing cost
Clamping	Negligible if frequency known, otherwise cost of finding frequency information.
Direct	None.
EWA	Evaluation of n samples of one-dimensional filter. Storage is n floating point numbers. (This is why radial symmetry required.)
NIL Maps	O($M^3x^3log_2(x)$) x: Resolution of data set

Pre-processing cost

Table 5.9

Technique	Evaluation cost
Clamping	Fixed
Direct Fixed Direct adaptive	Fixed o(F)
EWA-3	$O(n^3)$ $(n \times n \times n)$ samples in box
NIL Maps	Storage $O\left(\frac{N_t}{tol+1}M^3\right)$ Storage $O\left(\frac{N_t}{tol+1}M^3\right)$ (M: Order of approximating patches.) (N_t:Number of trigger points.) (tol: Tolerance.)

Evaluation cost

Table 5.10

Technique	Visual evaluation.
Clamping	Abrupt changes possible
Direct	Depends on filter used
EWA	Aliasing still possible when sampling restricted.
NIL Maps	Truncated filter can lead to normalization problems. (Scale factor goes to zero) For N=1, Discrete blocks aligned with axis.

Visual evaluation
Table 5.11

Chapter 6
Filters for volume rendering

The goal of volume rendering is to display objects and structures that are not normally visible. Because of this it is impossible to apply a reality measure to the resulting images. This means we are free to choose an arbitrary display model, such as a medium that absorbs light, or a medium that absorbs and emits light for instance. Once this display model is chosen then we must ensure that we display the data set accurately according to the model chosen. By choosing new display models and display methods new tools can be developed to display volumetric data.[1]

As we have shown in Chapter 3, the evaluation of these volumetric filters is quite costly. Not only is this evaluation costly, it may also be the case that the implementation of a filter requires a large amount of specialized code to be written. In this chapter we three examples to show that by using NIL maps we were able to prototype volume rendering filters. The first set of examples shows how NIL maps can be used to approximate traditional volume rendering techniques. The second example shows how slices of the data can be extracted using a Gaussian filter approximated by NIL maps. The third example shows how NIL maps were used to find a technique for highlighting the wind-passage in our example MRI data set. This example uses a modification of a current volume rendering technique [Sabe88] to select a sample position. Once the point of interest is found a filter is evaluated in its neighbourhood. The value resulting from the application of this filter is used as the pixel intensity. In particular, if a surface-detection filter is used, the surface of the wind-passage is more clearly displayed.

[1] The relationship between the display model and the display technique for volume rendering is similar to the relationship that exists between illumination models and shading techniques. The illumination model is used to model the interaction of light with a surface element and the shading technique is used to evaluate the illumination model. Usually the shading technique introduces a simplification to the illumination model. It is in this sense that the display technique in volume rendering applications is used to evaluate the display model.

6-0.1 Data set for examples

The examples in this chapter all use a $128 \times 128 \times 21$ sub-sampled version of a $256 \times 256 \times 21$ 8 bit per voxel MRI (Magnetic Resonance Imaging) data set. This smaller subset of the data was used due to the high memory overhead of the NIL map filtering technique.

This data set was obtained as part of a sleep apnea study at the UBC faculty of dentistry. Hannam[2] defines sleep apnea as follows:

> Sleep apnea is a disorder caused by an upper airway obstruction during sleep. It is not evident in the waking state. The soft tissues of the upper airway change shape and result in a constriction of the airway. Diagnosis of the disorder is currently made by a combination of bio-medical imaging and polysomographic readings made in a sleep disorder clinic. Treatment includes surgical correction of soft tissues, the use of intra-oral appliances to control tongue posture during sleep, and positive airway maintenance with continuous airflow.

Image resolution

The images in Plates 6.1, 6.2, and 6.15 were computed at a resolution of 640×480. The remaining plates were computed at a resolution of 320×240.

6-1 Conventional display

Previously we showed (in Chapter 3) that the display of volumetric data under a particular display model can be formulated

$$\mathcal{I} = \int_{u_0}^{u_1} \int_{v_0}^{v_1} I du dv = \int_{u_0}^{u_1} \int_{v_0}^{v_1} \int_{t_a}^{t_b} e^{-\tau(t-t_o)} \rho(x(t), y(t), z(t)) dt du dv, \qquad (6.1)$$

where τ is a user parameter that controls the strength of the scattering effect. This equation models a medium where the scattering effect is only dependent on the optical depth. As we noted this is the evaluation of the application of a filter to the volumetric data. In this case the filter is defined as the product of an exponential decay filter

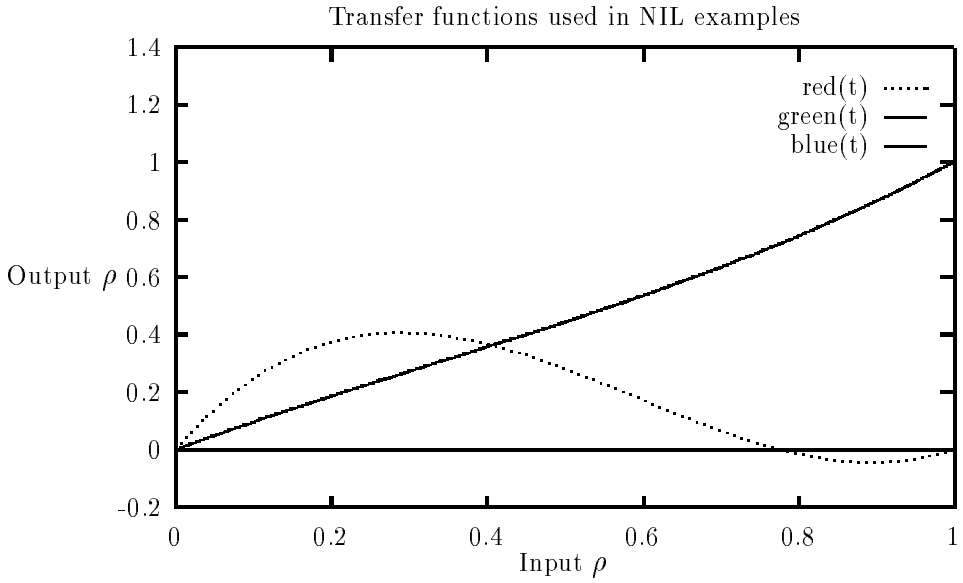
[2]Dr Alan Hannam, Professor of Oral Biology, Faculty of Dentistry, University of British Columbia, Personal communication.

($e^{-\tau(t-t_o)}$) and a box filter applied over the pixel area on the viewing screen. By rewriting this equation as,

$$\mathcal{I} = \int_{u_0}^{u_1} \int_{v_0}^{v_1} \int_{t_a}^{t_b} F(u,v) e^{-\tau(t-t_o)} \rho(x(u,v,t), y(u,v,t), z(u,v,t)) dt du dv,$$

we see how this three-dimensional filter can be defined as the product of a two-dimensional filter and a one-dimensional filter.

In the following examples we use NIL maps to display this data set using a Gaussian screen filter and an exponential decay filter along the line of sight. In Plates 6.4, 6.5, and 6.6 we use the transfer functions illustrated in figure 6.23.



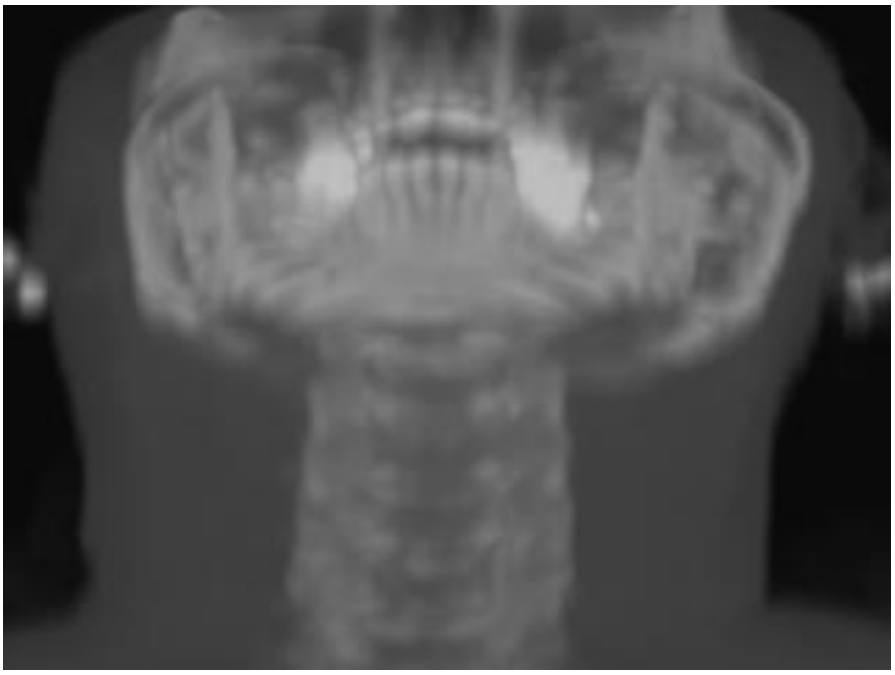
Transfer functions used in NIL map examples. These transfer functions are defined in terms of Lagrange polynomials.

Figure 6.23

In Plates 6.1 and 6.2 we see the image displayed using Sabella's [Sabe88] method using a super-sampling rate of 16 rays per pixel and 256 sample points per ray. The 320 $\times$ 240 versions of these images required 15 minutes of CPU time. In Plate 6.3 we present the display of the data from the same point of view using a constant patch

Side view of the data set used in examples. This $128 \times 128 \times 21$ 8 bit MRI data set is used throughout the chapter. This image was computed using Sabella's [Sabe88] technique. The value for the τ parameter in this image is 3. The red channel contains the computed intensity, and the green channel contains the max value encountered along the ray.

Plate 6.1



Front view of data set used in examples.

Plate 6.2

Patch order Trigger points	2	4	8	16
0	2	3	5	10
1	4	7	13	32
2	13	20	40	–

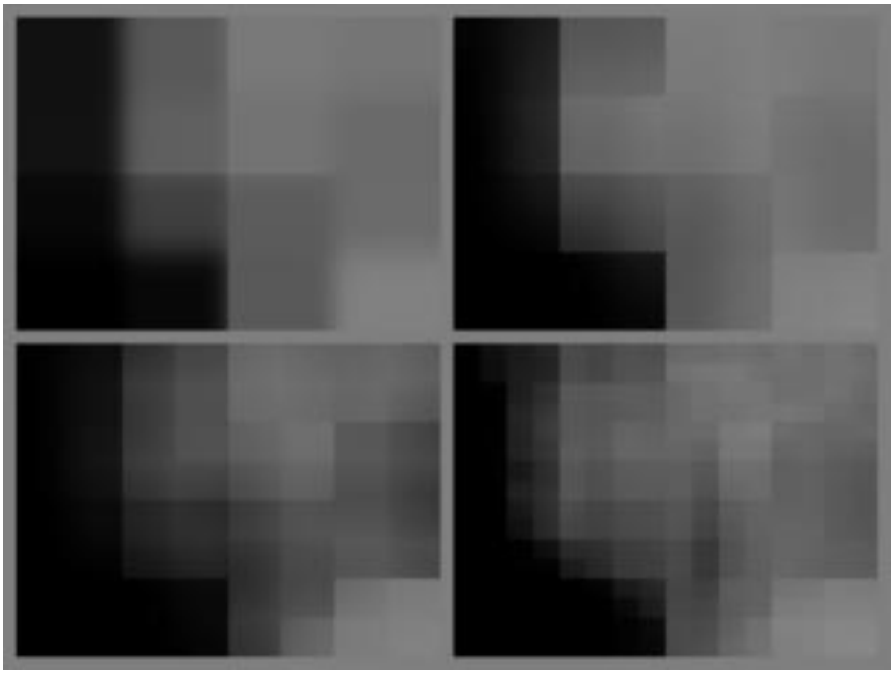
NIL map volume rendering timings (Min).

Table 6.1

approximation. The tolerance for subdivision in these images was set to 2 and the number of trigger points used was 4, 8, 16, and 32. This means that the level of approximation was $2 \times 2 \times 2$, $4 \times 4 \times 4$, $8 \times 8 \times 8$, and $16 \times 16 \times 16$, respectively. Plate 6.4 presents the display using a linear patch approximation. These two sets of images (Plates 6.3 and 6.4) illustrate the results which can be generated using a very coarse approximation to the filters with NIL maps. The high cost of evaluating these images discouraged further investigation in this direction. The techniques which directly evaluate the display filters using ray-marching are much better suited to the task than NIL maps are.

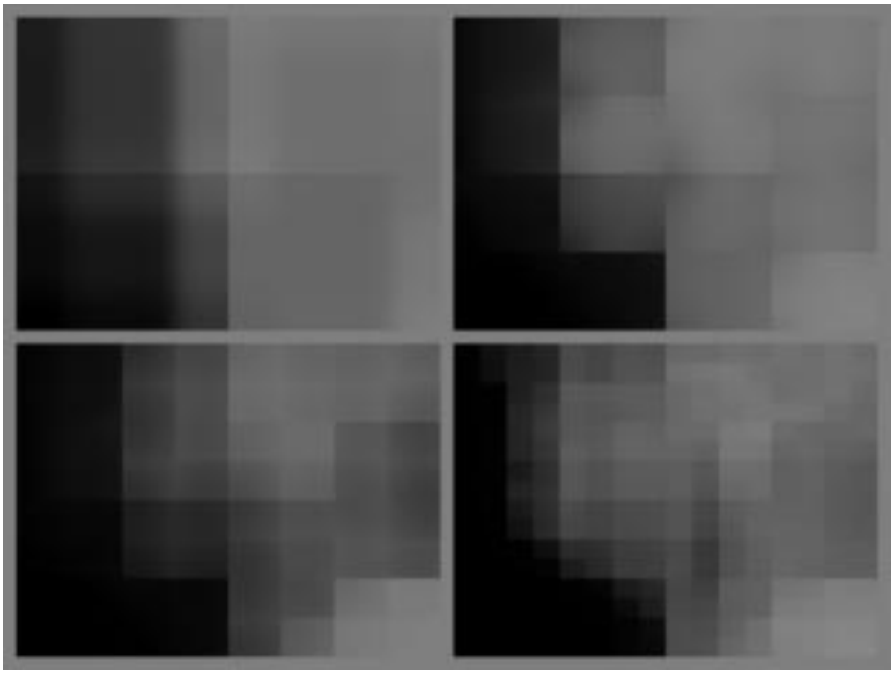
In Plate 6.5 we show the data set displayed using a quadratic patch approximation to the filter. Notice that for a $2 \times 2 \times 2$ representation of the data there is quite a bit of detail visible. When we contrast this with the display of the data using direct display (Sabella) with reduced sampling we see that NIL maps allows the use of a low resolution version of the data. In contrast lowering the sampling rate on the direct evaluation technique would yield poor results. The display of the data using 2, 4, 8, and 16 sample points with direct evaluation is presented in Plate 6.7. Even though the image in Plate 6.5 provides a good abstract display of the data for a $2 \times 2 \times 2$ data set. It is hard to see where such a simple display of a volumetric data set would prove useful. In table 6.1 we present the time required to produce the NIL map renderings. By increasing the number of approximating patches a fairly good display (Plate 6.6) of this data set was generated.

As expected NIL maps do not perform particularly well in this application. The high overhead cost associated with finding the approximation to the filter is the main cause of



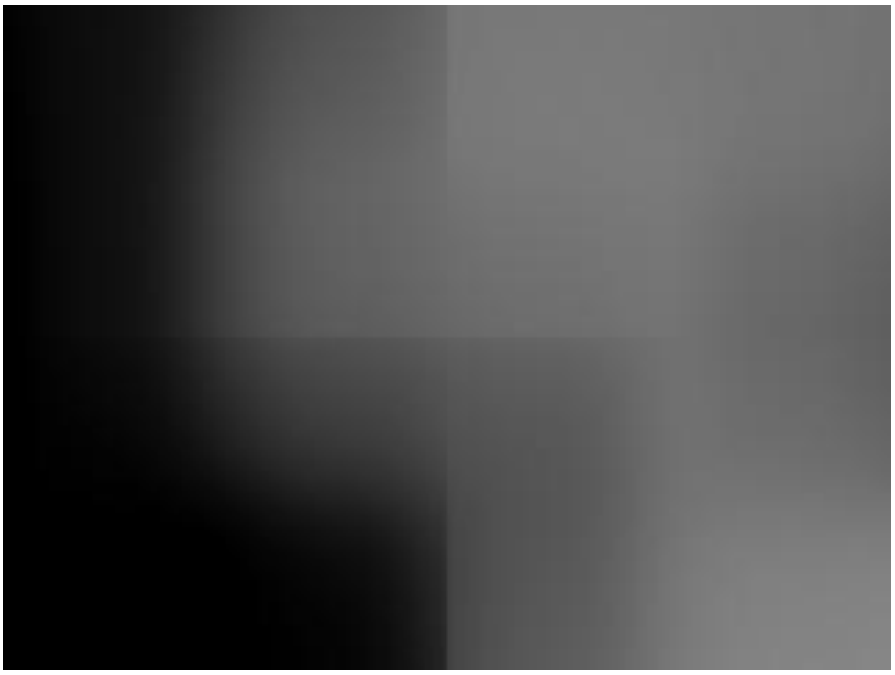
Constant patch approximation using tolerance = 2, and trigger 4, 8, 16, 32. The images are ordered left to right and top to bottom.

Plate 6.3



Linear patch approximation using tolerance = 2, and trigger = 4, 8, 16, 32. The images are ordered left to right and top to bottom.

Plate 6.4

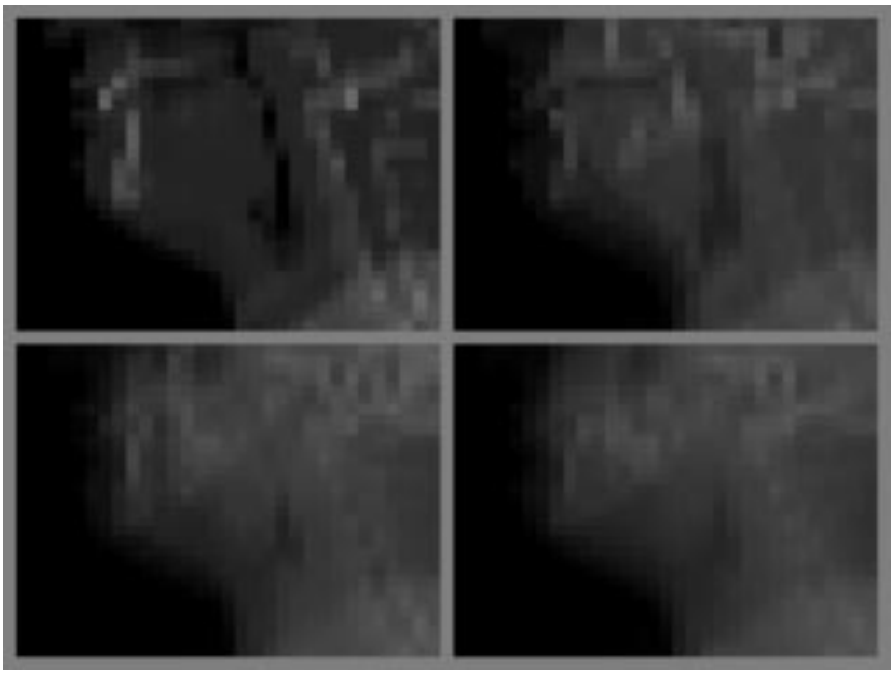


Quadratic patch approximation using the $2 \times 2 \times 2$ level of the NIL map

Plate 6.5

Linear patch approximation of general filter for $128 \times 128 \times 21$ data set. The number of trigger points is 512. The tolerance is 3.

Plate 6.6



Direct rendering with 2,4,8,16 samples per ray.

Plate 6.7

this. If a smaller representation of the data is required then the use of the top levels of the NIL map will provide the required abstraction. This property of NIL maps is more useful in texture mapping applications than it is in volume rendering applications.

6-2 Slicing the data

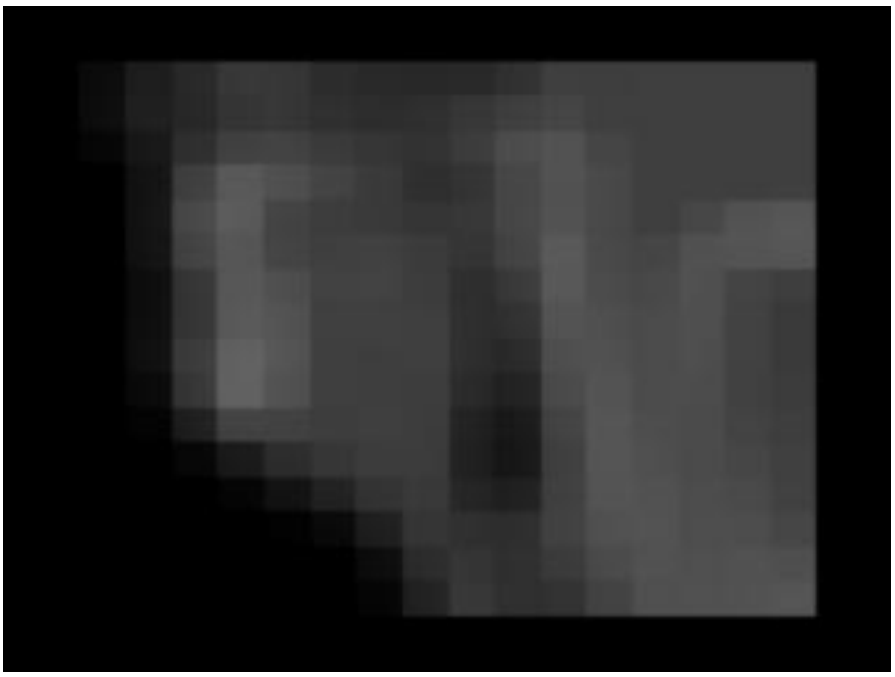
Often a slice of the data is required. This slice is a two-dimensional sample set taken from the original data set. The position and orientation of these slices is arbitrary. By using NIL maps we show how a filtered slice of the data can be constructed. The slice is specified by specifying a distance from the eye. For each pixel a Gaussian reconstruction filter is placed at the pre-defined point along the ray that goes through the centre of the pixel. The resulting image is the slice. By varying the width of the filter we can obtain slices that represent different widths of the data. In Plates 6.8 and 6.9 we illustrate slices obtained with a slice width of 0.1 and 0.01 respectively.

Another possible application is the extraction of the slices as required in the Fourier domain rendering of the data [Tots93, Levo92, Malz93]. In this approach a slice of the data must be extracted from the Fourier representation of the volume. The possibility of using NIL maps for this slice extraction is currently being investigated. The uniform size of the reconstruction filter seems to indicate that the pyramidal aspect of NIL maps will not be as important as when variable size filters are being approximated.

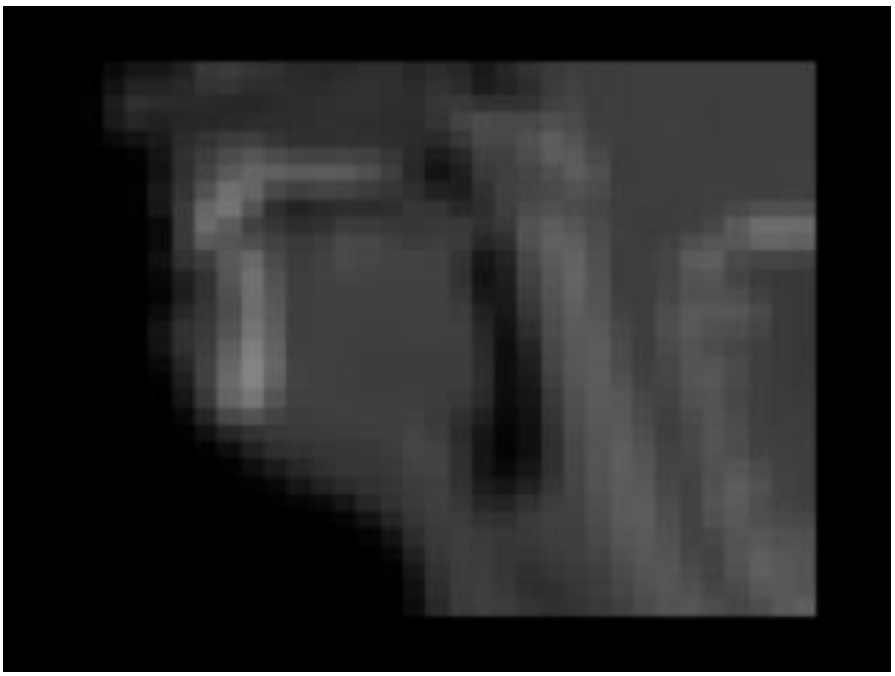
6-3 Maximum revisited

The initial display of the data with NIL maps did not really display the wind-passage as well as expected. Using the slice method we could extract a variety of slices to highlight the constriction in the wind-passage. In this section we show how the combination of a simple search technique with a surface-detection filter can be used to highlight the wind-passage.

One of the simplest tools for the display of volumetric data is to compute the maximum along a line of sight and use this as the intensity of the resulting pixel [Sabe88]. Unfortunately this technique does not help us in the task of displaying the wind-passage



Thick slice of the data
Plate 6.8



Thin slice of the data
Plate 6.9

for the sleep apnea study as can be seen in Plates 6.1 and 6.2. We can use a variation of this technique to position a filter that highlights surfaces in the volumetric data. The density near the wind-passage lies roughly between 0.14 and 0.18[3].

Given a user defined search density we choose the position of the filter by searching along the ray for the point with the closest density to this search density. If we use these densities as the intensity values for the pixels[4] a image with almost uniform intensity is produced. If one looks at the image carefully the outline of the wind-passage is faintly visible. By positioning a difference of Gaussians (DOG) filter at this point we were able to clearly highlight the wind-passage. In Plates 6.11–6.14 we illustrate the results obtained using this combination of techniques.

The wind-passage outline is quite easy to find in the images produced with a search density of 0.14. In these Plates (6.11 and 6.12) there appears to be a complete constriction of the wind-passage. Fortunately for the patient this does not represent his/her wind-passage. By ranging the search density from 0.14 to 0.19 we were able to get a good idea of the shape of the wind-passage. This process of finding the correct search density can be interactively controlled. In Plates 6.13 and 6.14 we see a front and a side view of the data set corresponding to a search density of 0.18. If we compare this to the image in Plate 6.15[5] we see that we have a close match in the shape of the wind-passage. The irregularities in these images (Plates 6.11-6.14) are due to the simple search method and not to the filter. When our simple search method finds a location which is not near the wind-passage[6] the application of a DOG filter will produce a low response.

The images presented in this section are the original images that were produced during the initial implementation and experimentation step. Because the intention of this chapter is to illustrate how NIL maps can be used in the investigation of filters we did not spend any additional time trying to improve the pictures.

It is interesting to note that the compact shape of the filters allowed a much faster

[3]The original data is an 8 bit unsigned data set. In the current NIL map implementation this data is stored in a floating point representation with the density in the range [0,1].

[4]Either the density values or a function of the density values can be used. In either case the image contains little useful information

[5]This image was obtained using a Marching Cubes implementation[Lore87]. The iso-surface is defined by thresholding the data at a density of $0.18 \approx \frac{46}{255}$.

[6]Or the ear passages.

Search and display of volumetric data. The search density (ρ_o) is 0.14. The outline of the wind-passage is faintly visible.

Plate 6.10

Pre-processing	Subdivision	Control point evaluation	Placement of trigger points	Patch integration
3.79	64.73	23.22	2.11	1.59

Percentage of time spent by the different NIL map components
Table 6.2

approximation of the filter to be used. Typically nine NIL cells were used to approximate each filter. This means that the computation of each these images required about five minutes.

The examples in this Chapter illustrate how NIL maps can be used as a tool for the investigation of volume rendering filters. The three examples presented in this chapter were implemented by altering the trigger point placement code and the filter-weight evaluation code. In particular, the implementation of the last technique was done in the course of an afternoon.

6-4 Comments on NIL maps for volume rendering

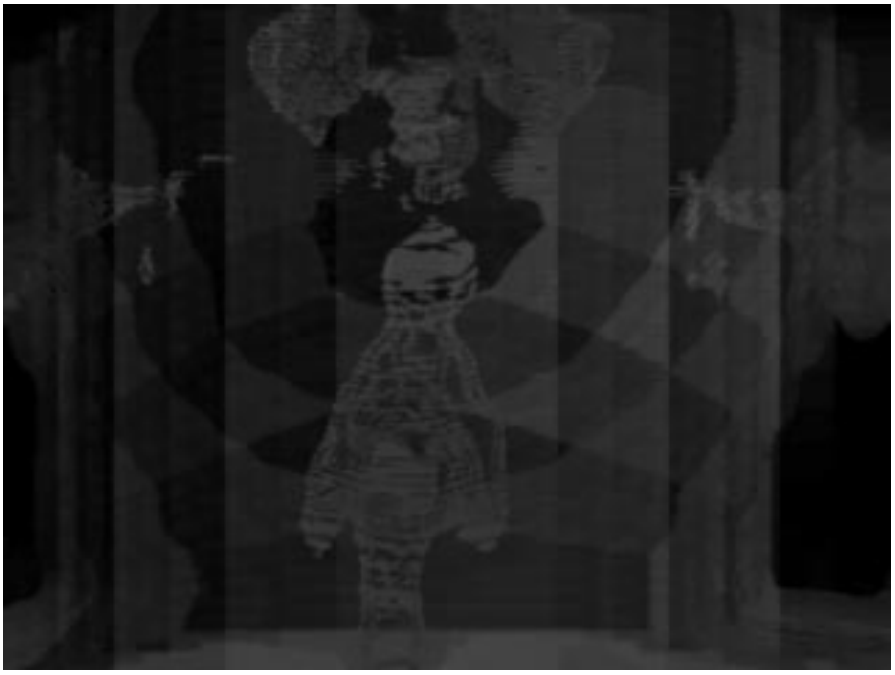
NIL maps have allowed us to implement quickly and experiment with a number of filters. The three examples presented here highlight the main disadvantages and advantages of NIL maps in the context of volume rendering. The cost of using NIL maps as a conventional volume rendering technique is high. This high cost is primarily due to the cost of evaluating the approximating hierarchy. In the current implementation a major portion of the execution time ($\approx 87\%$) is spent finding the approximating hierarchy and the weights of the control points. In table 6.2 we present a typical set of statistics gathered by a standard UNIX^{TM} profiling tools[7].

This problem is partially addressed if we store the filter-approximating hierarchy for each pixel. If a new display of the data set is required and the current hierarchies are still valid this will significantly reduce the cost of the display. In particular this is true

[7] Pixie and prof on a Silicon Graphics workstation.

Side view of data set. Display of the data using the sample and filter display. The search density (ρ_o) is 0.14, and the display filter is a difference of Gaussians (DOG) filter.

Plate 6.11

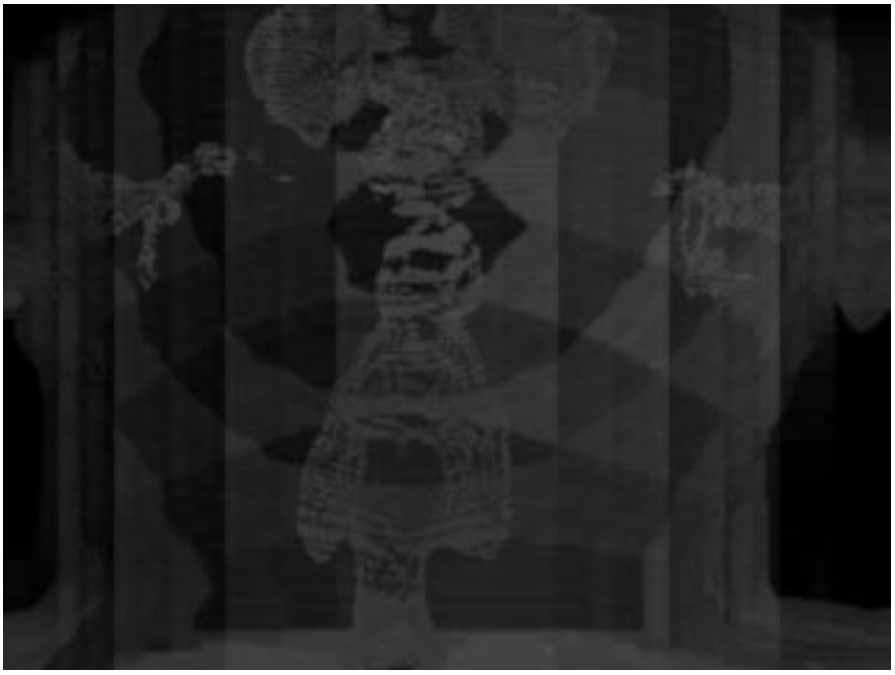


Front view of data set. The search density (ρ_o) is 0.14.

Plate 6.12

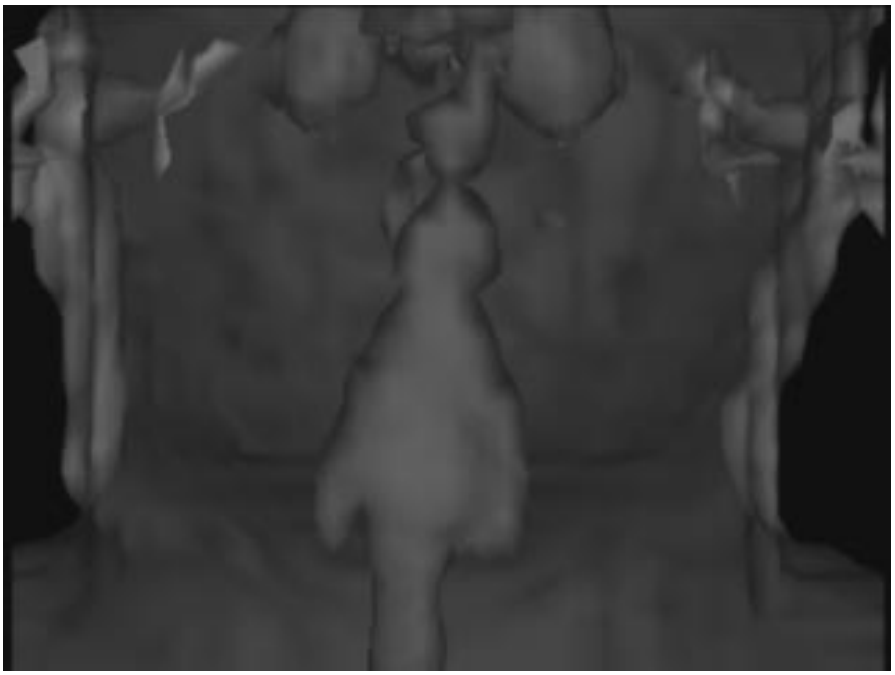
Side view of data set. The search density (ρ_o) is 0.18.

Plate 6.13



Front view of data set. The search density (ρ_o) is 0.18.

Plate 6.14



Iso-surface generated using Marching Cubes technique. The threshold used to generate this was 0.18.

Plate 6.15

when transfer functions are being altered.

If the view of a data set is changed then we have no choice but to recompute the hierarchy for each pixel. However, there are situations where this hierarchy need not be completely recomputed. These include transfer function manipulation, tolerance increase/decrease, and slight changes in viewing position. When either the number of trigger points or the tolerance that sets the subdivision is altered the hierarchy that approximates a filter can be modified by further subdividing it or by collapsing NIL cells into higher representations. In the case of a slight change in the point of view the old hierarchies may be examined in order to determine their validity.

There are a number of situations where we may wish to extract a slice from the data. NIL maps can be used to approximate the interpolation filters. This allows higher quality filters such as Gaussian filters to be used. The hierarchical aspect of NIL maps does not seem be so important for this application.

The main advantage of NIL maps is seen in the last example presented above. The ability to rapidly implement filters encourages research into alternate filters for volume rendering. Once an interesting filter is found it can be implemented in a more efficient manner. We believe this to be one of the more important contributions of NIL maps for volume rendering.

There is one small remaining problem with the current implementation of NIL maps. If we know that the filter can adequately be approximated using a linear approximation, but we have computed the NIL map using a higher order approximation we must either re-compute the NIL map using the appropriate basis or use a higher order polynomial to approximate the filter. Both of these alternatives can be avoided if we choose a polynomial basis set $\{B_i(t)\}$ with the property that the degree of the polynomial $B_i(t)$ is i. Two examples of such a basis are the power basis

$$\begin{aligned} B_0(t) &= 1 \\ B_1(t) &= t \\ B_2(t) &= t^2 \\ B_3(t) &= t^3 \end{aligned}$$

and the Chebyshev polynomials

$$\begin{aligned} B_0(t) &= 1 \\ B_1(t) &= t \\ B_2(t) &= 2t^2 - 1 \\ B_3(t) &= 4t^3 - 3t. \end{aligned}$$

When a basis with the above property is chosen we can evaluate a lower order approximation without having to compute the M control points. If such a basis is chosen for the NIL map representation we must perform a control point transformation between the approximating basis functions, and the internal basis functions.

6-5 Wrapup

The display of volumetric data using filters is and will continue to be an important tool for the study of these data-sets. By developing new filters as display tools we are able to provide more sophisticated tools for this study. In order to develop these display filters we need a tool with which filters can be quickly developed and tested. NIL maps is such a tool. The use of NIL maps for the evaluation of these filters allows preliminary study into the use of filters for volume rendering with out requiring the generation of large amounts of 'specialized' filter code to be written. Considered in this way the use of NIL maps is a middle step towards the development of more complex filters for the display of volumetric data. The flexibility of the NIL maps technique comes with a high cost. This suggests that filters found using NIL maps as an exploratory tool need to be implemented directly if they are to be used in a production system. The use of NIL maps allows us to delay this specialized code development step until we are sure of the usefulness of a particular filter.

Chapter 7
Conclusions

The end

This dissertation studied the display of volumetric data in computer graphics. In both texture mapping and volume rendering this display can benefit from the application of three-dimensional filters. These filters range from simple anti-aliasing filters to more complex filters designed to produce a particular effect. The evaluation of these filters requires the computation of an expensive triple integral. This high cost motivated the search for approximating techniques for filter evaluation.

Four filter evaluation techniques were presented, direct evaluation, NIL maps, EWA filters and clamping. Direct evaluation uses numerical quadrature rules to evaluate the integrals. NIL maps and EWA filters are extensions of their two-dimensional counterparts proposed by Fournier and Fiume [Four88a] and Greene and Heckbert [Gree86] respectively. The use of clamping [Nort82] for anti-aliasing three-dimensional textures was proposed by Perlin [Perl85].

7-0.1 Three-dimensional texture map filtering

The implementation and application of these four filtering techniques allowed us to compare their relative performance for texture mapping. The evaluation criteria included a variety of objective and subjective items. Based on this evaluation it is difficult to present any of these techniques as the '*best*' one. Each of these techniques has clear advantages and disadvantages. One possible ranking is based on the generality of the techniques.

Direct evaluation places no constraints on the class of textures that can be used. The class of filters that can be evaluated using direct evaluation is limited only by the quadrature rule chosen. In most cases this means that the filter is evaluated over a rectangular parallelepiped. Even though this approach is quite costly, it does provide us with the ability to evaluate these filters within a pre-determined tolerance.

The development and study of new filters with NIL maps is easily done. This flexibility

allows the quick prototyping of filters for both texture mapping and volume rendering. Once interesting filters are developed it is probably wise to implement them using more efficient methods. It is also possible to use a different basis set for approximating the filters, as was done in two dimensions for Gaussian filters by Gotsman [Gots93].

EWA filters are restricted to the evaluation of radially symmetric filters over ellipsoidal volumes. This restriction allows the use of a pre-computed one-dimensional sample of the filter. This technique is useful when the cost of evaluating the filter function is high compared to the texture evaluation cost. When EWA filters are being considered two issues must be considered. First, EWA filters approximate a filter centered in the ellipsoid that, as we showed, can introduce quite a large error when the perspective distortion is large. Second, the fixed point Gaussian quadrature rule provided a better approximation than EWA filters do. If EWA filters are being used for procedural textures then the sample points can be placed at the same positions as the sample points generated by the fixed point quadrature rule. This is not an option when discrete textures are being used unless a reconstruction filter is being used to generate a continuous signal. In this case the reconstructed signal can be treated in the same manner as a procedural texture.

The least general of these techniques is clamping. It requires some information about the Fourier spectra of the texture. The filter is approximated by a low order polynomial approximation to its Fourier transform. In most of the applications of clamping that we have encountered the cost of evaluating a filtered sample is lower than that of evaluating an un-filtered sample. If simple anti-aliasing is required and the cost of more sophisticated filtering cannot be afforded, then clamping is an option that should be considered.

7-0.2 Filters for volume rendering

The display of volumetric data allows us to see that which is not normally visible. In other areas of computer graphics we can apply some sort of *reality measure*; this is not an option for volume rendering. Rather than striving to produce pictures of a certain realistic quality, volume rendering research has concentrated on producing a variety of display tools with which volumetric data sets can be studied. The use of filters for volume rendering extends this set of tools in a significant way.

A popular display model for volumetric data is to consider the data measurements as

the local density of a volume of gas that either absorbs or emits light. By simplifying this model it is possible to formulate the display of these data sets as a filtering operation. This investigation was prompted by the similarities between the display of volumetric textures near the surfaces of objects and the direct display of volumetric data. The initial investigation of filters for volume rendering was done using NIL maps as the filter evaluation method. By using this method we were able to quickly study different filters for volume rendering.

This research lead to the incorporation of filters with a more conventional display technique. In this hybrid method a ray marching method is used to locate a point of interest. The display of this point does not provide much information, however, the neighborhood of the point may be interesting. By placing a filter at this point and using the result of its application a more interesting display was generated.

The simplicity of this approach and the results achieved thus far highlight the potential of this approach. The properties of the data that need to be highlighted differ between volume rendering applications. By allowing the display filters to be designed by the users we hope to provide a more sophisticated set of tools for the display, study, and analysis of volumetric data sets.

7-0.3 NIL maps

NIL maps provide a general filter evaluating tool for volume rendering and texture mapping. The overhead of the technique is quite high, but it allows for the rapid prototyping of three-dimensional filters. In texture mapping it allows a wide range of procedural textures to be used. The only limitation is that the textures be defined by a finite (hopefully small) set of functions. In volume rendering NIL maps incorporate transfer functions. These transfer functions are also restricted to be defined by a set of basis functions.

The ease with which filters can be implemented and studied using this technique makes it an attractive one for the preliminary study of filters in both of these applications. Once a particular filter is selected for a task it is probably the case that a more efficient evaluation scheme can be found.

7-1 Contributions

The contributions of this thesis are:

- An overview of the computer graphics literature relevant to the filtering of volumetric data. This included an overview of the two-dimensional filtering literature from which two techniques were selected for extension to three dimensions.
- It was shown that the display of three-dimensional textures, the anti-aliasing of three-dimensional textures, and the display of volumetric data can all be formulated as filtering operations.
- Three new filtering techniques for three-dimensional textures were developed:
  - Direct evaluation:
    The development of a three-dimensional filter evaluation technique based on Simpson's adaptive quadrature rule. Allows the accurate evaluation of three-dimensional filters to within a pre-defined tolerance.
  - NIL maps:
    This technique is an extension of the two-dimensional version. The extension includes methods for procedural textures and transfer functions. The flexibility of this technique makes it a good method for the rapid prototyping of filters both in the context of three-dimensional texture mapping and volume rendering.
  - EWA filters:
    Again, an extension of the two-dimensional version. This technique is easy to implement but is restricted in the class of filters that it can approximate. The performance of this technique is easily matched by a fixed point quadrature rule. If the cost of evaluating the filter is high we could still use the EWA technique to reduce the cost of computing the filter-weight function.
- These three filter evaluation techniques were evaluated in the context of three-dimensional texture mapping. A comprehensive criteria was used that allows the correct technique to be chosen depending on the application.

- Examples of the application of filters to volume rendering were presented. In particular the combination of filtering and other display techniques was shown to produce a innovative display of the data.

7-2 Future work

In this thesis we have provided a set of tools for approximating the evaluation of three-dimensional filters when they are applied to volumetric data. A number of issues arise out of this research:

- If we wish to use filters for volume rendering it is reasonable to allow the user to tailor these filters for a particular task. The definition and manipulation of these filters is not an intuitive task. Several questions arise:
  - In what ways can these filters be defined and controlled?
    The mathematical definition of the filter does not provide an intuitive feel for the function[1] of the filter. Other methods for describing or defining these filters must be found.
  - How can these filters be displayed? In one and two dimensions we can display by plotting their function, this is not so easy for three-dimensional filters because their display requires the display of a surface in four dimensions.
  - What queries of the data can be formulated as a filter operation? By combining a ray marching search method with a filter display we were able to display more information about the neighborhood of a point of interest. The application of filters to both the search step and the display step merit investigation. There are many properties of the data, such as surfaces defined by a sudden density change, whose location can be determined by the application of a filter.
- The range of textures that we can generate using the procedural textures is far richer than the colour textures we have studied. As we indicated earlier there is research on the filtering of such textures in two dimensions [Four93]. The extension

[1] No pun intended

of these techniques to three-dimensional textures should prove to be an interesting and challenging study.

- We have shown how many volumetric data display techniques can be formulated as a filtering operation. A study of the various uses and/or sources of volumetric data may highlight other situations where a filtering approach will help in the display of this data.

- There are a variety of properties of the data that are used in volume rendering that cannot be expressed as a linear filtering operation. One such example is the maximum density along a viewing ray. This property has been used extensively for the display of volumetric data. An investigation of such problems from the point of view of hierarchical data structures, and/or pyramidal data structures may yield interesting results.

Notation

$B_i(t)$ One-dimensional basis function.

$\overset{\gamma}{C}{}_{i}^{m}$ NIL map cell from γ level derived from basis function $B_i(t)$. The super-script m indicates the cell's offset in the γ level of the NIL map.

$\overset{\gamma}{D}{}_{il}^{m}$ NIL map cell from γ level which incorporates a transfer basis function. The l sub-script indicates which transfer function basis function ($B_l(t)$) the NIL cell is derived from. The super-script m indicates the cell's offset in the γ level of the NIL map.

I Place holder for the integral result.

$T(t)$ Texture function.

$\mathcal{T}_\beta(t)$ Transfer function, the β sub-script indicates what channel the data is mapped into. In the context of NIL maps this transfer function is defined in terms of a basis set $b_l(t)$.

$$\mathcal{T}_\beta(t) = \sum_{l=0}^{N-1} b_l B_l(t)$$

$\overline{T}()$ Signal resulting from a reconstruction of a discrete data set.

$X_r(t)$ Procedural texture basis function. The texture function is then defined by

$$T(t) = \sum_{r=0}^{R-1} x_r X_r(t)$$

u, v, w Texture space coordinates.

$\overset{\gamma}{\mathcal{X}}{}_{ri}$ NIL map cell from γ level which incorporates a procedural texture basis function. The r sub-script indicates which texture basis function ($X_r(t)$) the NIL cell is derived from.

x, y, z Object space coordinates.

Glossary

Aliasing:
: Introducing a signal component that should not be there.

CAT:
: Computer Aided Tomography.

DC:
: Constant term in a signal. In a Fourier transform of a signal the DC component is the coefficient of the zero$^{\text{th}}$ term.

Discrete texture:
: A discrete representation (usually sampled) of a texture.

EWA:
: Elliptical Weighted Average filters. A filter approximation technique designed to accelerate the computation of radially symmetric filters. Primarily of interest when a Gaussian is being used.

Filter:
: A weighting function.

Modelling:
: Building an abstract (usually mathematical) representation.

MRI:
: Magnetic Resonance Imaging.

NIL map:
: A pyramidal data structure used for storing pre-integrated basis functions. NIL stands for *nodus in largo*. Roughly translated this means knot large.

Pixel:
: Picture element, or frame buffer element.

Procedural texture:
: A continuously defined texture implemented in some programming language.

Reconstruction:
: Generating a continuously defined signal from a discrete sample set.

Rendering:
: Using a model and a display technique to assign values to pixels in a frame buffer.

Sampling:
: Taking a finite number of samples of a signal.

Texture:
: A high frequency (usually) surface characteristic.

Texel:
: Texture element

Transfer function:
: A function $\mathcal{T}(t)$ which maps a scalar data-set or scalar function into another domain.

Voxel:
: Volume element.

Bibliography

[Adel85] Edward. H. Adelson and James R. Bergen. "Spatiotemporal energy models for the perception of motion". *J. Opt. Soc. Am. A*, Vol. 2, No. 2, 1985.

[Artz79a] E. Artzy. "Display of three-dimensional information in computed tomography". *Computer Graphics and Image Processing*, Vol. 9, pp. 196–198, February 1979.

[Artz79b] E. Artzy, G. Frieder, G.T. Herman, and H.K. Liu. "A system for three-dimensional dynamic display of organs from computed tomograms". *Proc. The Sixth Conference on Computer Applications in Radiology and Computer-Aided Analysis of Radiological Images*, June 1979.

[Artz80] E. Artzy, G. Frieder, and G.T. Herman. "The theory, design, implementation, and evaluation of a three-dimensional surface detection algorithm". *Computer Graphics (SIGGRAPH '80 Proceedings)*, Vol. 14, No. 3, pp. 2–9, July 1980.

[Artz81a] E. Artzy, G. Frieder, and G.T. Herman. "The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm". *Computer Graphics and Image Processing*, Vol. 15, pp. 1–24, January 1981.

[Artz81b] E. Artzy and G.T. Herman. "Boundary detection in 3-dimensions with a medical application". *Computer Graphics*, Vol. 15, No. 2, pp. 92–123, March 1981.

[Benn89] Chakib Bennis and Andre Gagalowicz. "2D macroscopic texture synthesis". *Computer Graphics Forum*, Vol. 8, No. 4, pp. 291–300, December 1989.

[Blin78a] J.F. Blinn. *Computer Display of Curved Surfaces*. Ph.D. Thesis, University of Utah, 1978.

[Blin78b] J.F. Blinn. "A scan line algorithm for displaying parametrically defined surfaces". *Computer Graphics (Special SIGGRAPH '78 Issue, preliminary papers)*, pp. 1–7, August 1978.

[Blin78c] J.F. Blinn. "Simulation of wrinkled surfaces". *Computer Graphics (SIGGRAPH '78 Proceedings)*, Vol. 12, No. 3, pp. 286–292, August 1978.

[Bovi87] A. C. Bovik, M. Clark, and W. S. Geisler. "Computational texture analysis using localized spatial filtering". *IEEE Workshop on Computer Vision*, November 1987.

[Bren70] B. Brennan and W.R. Bandeen. "Anisotropic reflectance characteristics of natural earth surfaces". *Applied Optics*, Vol. 9, No. 2, February 1970.

[Buch91] John Buchanan. "The filtering of 3d textures". *Proceedings of Graphics Interface '91*, pp. 53–60, June 1991.

[Burd81] R. Burden, J. Faires, and A. Reynolds. *Numerical Analysis, Second edition.* PWS Publishers, Boston, Massachusetts, 1981.

[Cann86] J. Canny. "A computational approach to edge detection". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 8, pp. 679–698, 1986.

[Carp84] Loren Carpenter. "The A-buffer, an antialiased hidden surface method". *Computer Graphics (SIGGRAPH '84 Proceedings)*, Vol. 18, No. 3, pp. 103–108, July 1984.

[Catm74] Edwin E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* Ph.D. Thesis, University of Utah, December 1974.

[Catm75] Edwin E. Catmull. "Computer display of curved surfaces". *Proceedings of the IEEE Conference on Computer Graphics, Pattern Recognition, and Data Structure*, pp. 11–17, May 1975.

[Chen85] Lih-Shyang Chen, G.T. Herman, R.A. Raynolds, and J.K. Udupa. "Surface shading in the cuberille environment". *IEEE Computer Graphics and Applications*, Vol. 5, No. 12, pp. 33–43, December 1985.

[Chri78] H.N. Christiansen and T.W. Sederberg. "Conversion of complex contour line definitions into polygonal element mosaics". *Computer Graphics (SIGGRAPH '78 Proceedings)*, Vol. 12, No. 3, pp. 187–192, August 1978.

[Clin88] Harvey E. Cline, William E. Lorensen, Sigwalt Ludke, Carl R. Crawford, and Bruce C. Teeter. "Two algorithms for the reconstruction of surfaces from tomograms". *Medical Physics*, Vol. 15, No. 3, pp. 320–327, June 1988.

[Crow84] F.C. Crow. "Summed-area tables for texture mapping". *Computer Graphics (SIGGRAPH '84 Proceedings)*, Vol. 18, No. 3, pp. 207–212, July 1984.

[Dreb88] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. "Volume rendering". *Computer Graphics (SIGGRAPH '88 Proceedings)*, Vol. 22, No. 4, pp. 65–74, August 1988.

[Dung78] W. Dungan, A. Stenger, and G. Sutty. "Texture tiles consideration for raster graphics". *Computer Graphics (SIGGRAPH '78 Proceedings)*, Vol. 12, No. 3, pp. 130–134, August 1978.

[Eber90] David S. Ebert and Richard E. Parent. "Rendering and animation of gaseous phenomena by combining fast volume and scanline A-buffer techniques". *Computer Graphics (SIGGRAPH '90 Proceedings)*, Vol. 24, No. 4, pp. 357–366, August 1990.

[Feib80a] E. Feibush and D.P. Greenberg. "Texture rendering system for architectural design". *Computer-Aided Design*, Vol. 12, pp. 67–71, March 1980.

[Feib80b] E.A. Feibush, M. Levoy, and R.L. Cook. "Synthetic texturing using digital filters". *Computer Graphics (SIGGRAPH '80 Proceedings)*, Vol. 14, No. 3, pp. 294–301, July 1980.

[Ferr84] Leonard A. Ferrari and Jack Sklansky. "A fast recursive algorithm for binary-valued two-dimensional filters". *Computer Vision, Graphics, and Image Processing*, Vol. 26, No. 3, June 1984.

[Ferr85] Leonard A. Ferrari and Jack Sklansky. "A note on duhamel integrals and running average filters". *Computer Vision, Graphics, and Image Processing*, Vol. 29, March 1985.

[Fium83a] E. Fiume, A. Fournier, and L. Rudolph. "A parallel scan conversion algorithm with anti-aliasing for a general purpose ultracomputer". *Computer Graphics (SIGGRAPH '83 Proceedings)*, Vol. 17, No. 3, pp. 141–150, July 1983.

[Fium83b] E. Fiume, Alain Fournier, and Larry Rudolph. "A parallel scan conversion algorithm with anti-aliasing for a general-purpose ultracomputer: Preliminary report". *Proceedings of Graphics Interface '83*, pp. 11–21, May 1983.

[Fium87] E. Fiume, A. Fournier, and V. Canale. "Conformal texture mapping". *Eurographics '87*, pp. 53–64, August 1987.

[Fole90] J.D. Foley, A. van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Publishing Company, second edition, 1990.

[Four86] Alain Fournier and David A. Grindal. "The stochastic modelling of trees". *Proceedings of Graphics Interface '86*, pp. 164–172, May 1986.

[Four88a] Alain Fournier and Eugene Fiume. "Constant-time filtering with space-variant kernels". *Computer Graphics (SIGGRAPH '88 Proceedings)*, Vol. 22, No. 4, pp. 229–238, August 1988.

[Four88b] Alain Fournier and Donald Fussell. "On the power of the frame buffer". *ACM Transactions on Graphics*, Vol. 7, No. 2, pp. 103–128, April 1988.

[Four93] A. Fournier. "Filtering normal maps and creating multiple surfaces". *Work in progress*, 1993.

[Fuch77] H. Fuchs, Z.M. Kedem, and S.P. Uselton. "Optimal surface reconstruction from planar contours". *Communications of the ACM*, Vol. 20, pp. 693–702, October 1977.

[Gaga83] Andre Gagalowicz. *Vers un Modele de Textures.* Ph.D. Thesis, Universite Pierre et Marie Curie, Paris VI, 1983.

[Gaga86] A. Gagalowicz and Song-Di-Ma. "Model driven synthesis of natural textures for 3-D scenes". *Computers and Graphics*, Vol. 10, No. 2, pp. 161–170, 1986.

[Gaga87] Andre Gagalowicz. "Texture modelling applications". *The Visual Computer*, Vol. 3, No. 4, pp. 186–200, December 1987.

[Gaga88] Andre Gagalowicz and Song D. Ma. "Animation of stochastic model-based 3d textures". *Eurographics '88*, pp. 313–326, September 1988.

[Galy91] Tinsley A. Galyean and John F. Hughes. "Sculpting: An interactive volumetric modeling technique". *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, No. 4, pp. 267–274, July 1991.

[Gang82] M. Gangnet, D. Perny, and P. Coueignoux. "Perspective mapping of planar textures". *Eurographics '82*, pp. 57–70, 1982.

[Gard84] Geoffrey Y. Gardner. "Simulation of natural scenes using textured quadric surfaces". *Computer Graphics (SIGGRAPH '84 Proceedings)*, Vol. 18, No. 3, pp. 11–20, July 1984.

[Gard85] G.Y. Gardner. "Visual simulation of clouds". *Computer Graphics (SIGGRAPH '85 Proceedings)*, Vol. 19, No. 3, pp. 297–303, July 1985.

[Glas86] A. Glassner. "Adaptive precision in texture mapping". *Computer Graphics (SIGGRAPH '86 Proceedings)*, Vol. 20, No. 4, pp. 297–306, August 1986.

[Gots93] Craig Gotsman. "Constant-time filtering by singular value decomposition". *Eurographics Workshop on Rendering*, pp. 145–155, 1993.

[Gree86] N. Greene and P. S. Heckbert. "Creating raster omnimax images from multiple perspectives views using the elliptical weighted average filter". *IEEE Computer Graphics and Applications*, Vol. 6, No. 6, pp. 21–27, June 1986.

[Gree89] Ned Greene. "Voxel space automata: Modeling with stochastic growth processes in voxel space". *Computer Graphics (SIGGRAPH '89 Proceedings)*, Vol. 23, No. 3, pp. 175–184, July 1989.

[Grin84] D. Grindal. "The Stochastic Creation of Tree Images". M.Sc. Thesis, Department of Computer Science, University of Toronto, 1984.

[Heck86] P. S. Heckbert. "Filtering by repeated integration". *Computer Graphics (SIGGRAPH '86 Proceedings)*, Vol. 20, No. 4, pp. 315–321, August 1986.

[Heeg87] D. J. Heeger. "Model for the extraction of image flow". *J. Opt. Soc. Am. A*, Vol. 4, No. 8, 1987.

[Heeg88] D. J. Heeger. "Optical flow using spatiotemporal filters". *International Journal of Computer Vision*, 1988.

[Herm80] G.T. Herman. "Surfaces of objects in discrete three-dimensional space". *Computer Graphics 80, Proc. of a Conference at Brighton*, pp. 287–300, August 1980.

[Herm82] G.T. Herman, R.A. Reynolds, and J.K. Udupa. "Computer techniques for the representation of three-dimensional data on a two-dimensional display". *Proc. SPIE Int. Soc. Opt. Eng.*, Vol. 367, pp. 3–14, 1982.

[Herm83] G.T. Herman and J.K. Udupa. "Display of 3-D digital images: computational foundations and medical applications". *IEEE Computer Graphics and Applications*, Vol. 3, No. 5, pp. 39–46, August 1983.

[Jain81] Anil K. Jain. "Image data compression: A review". *Proceedings of the IEEE*, Vol. 69, No. 3, pp. 349–389, March 1981.

[Kaji84] James T. Kajiya and Brian P. Von Herzen. "Ray tracing volume densities". *Computer Graphics (SIGGRAPH '84 Proceedings)*, Vol. 18, No. 3, pp. 165–174, July 1984.

[Kaji85] J.T. Kajiya. "Anisotropic reflection models". *Computer Graphics (SIGGRAPH '85 Proceedings)*, Vol. 19, No. 3, pp. 15–21, July 1985.

[Kaji89] James T. Kajiya and Timothy L. Kay. "Rendering fur with three dimensional textures". *Computer Graphics (SIGGRAPH '89 Proceedings)*, Vol. 23, No. 3, pp. 271–280, July 1989.

[Lans91] Robert C. Lansdale. "Texture Mapping and Resampling for Computer Graphics". M.Sc. Thesis, Department of Computer Science, University of Toronto, January 1991.

[Laur91] David Laur and Pat Hanrahan. "Hierarchical splatting: A progressive refinement algorithm for volume rendering". *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, No. 4, pp. 285–288, July 1991.

[Levo88] Marc Levoy. "Display of surfaces from volume data". *IEEE Computer Graphics and Applications*, Vol. 8, No. 3, pp. 29–37, May 1988.

[Levo90a] Marc Levoy. "Efficient ray tracing of volume data". *ACM Transactions on Graphics*, Vol. 9, No. 3, pp. 245–261, July 1990.

[Levo90b] Marc Levoy. "A hybrid ray tracer for rendering polygon and volume data". *IEEE Computer Graphics and Applications*, Vol. 10, No. 2, pp. 33–40, March 1990.

[Levo90c] Marc Levoy. "Volume rendering by adaptive refinement". *The Visual Computer*, Vol. 6, No. 1, pp. 2–7, February 1990.

[Levo90d] Marc Levoy and Ross Whitaker. "Gaze-directed volume rendering". *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, Vol. 24, No. 2, pp. 217–223, March 1990.

[Levo92] Marc Levoy. "Volume rendering using the Fourier projection-slice theorem". *Proceedings of Graphics Interface '92*, pp. 61–69, May 1992.

[Lewi89] J. P. Lewis. "Algorithms for solid noise synthesis". *Computer Graphics (SIGGRAPH '89 Proceedings)*, Vol. 23, No. 3, pp. 263–270, July 1989.

[Lore87] William E. Lorensen and Harvey E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm". *Computer Graphics (SIGGRAPH '87 Proceedings)*, Vol. 21, No. 4, pp. 163–169, July 1987.

[Lyon89] Kic P. Lyons and Joyce E. Farrel. "Linear systems analysis of CRT displays". *Society for Information Display 89 Digest*, Vol. 20, pp. 220–223, 1989.

[Ma86] Song De Ma and A. Gagalowicz. "Determination of local coordinate systems for texture synthesis on 3-D surfaces". *Computers and Graphics*, Vol. 10, No. 2, pp. 171–176, 1986.

[Malz93] Tom Malzbender. "Fourier volume rendering". *ACM Transactions on Graphics*, Vol. 12, No. 3, July 1993.

[Mark91] Tassos Markas and John Reif. "Image compression methods with distortion controlled capabilities". *Data Compression Conference*, pp. 93–102, April 8-11 1991.

[Max90] Nelson Max, Pat Hanrahan, and Roger Crawfis. "Area and volume coherence for efficient visualization of 3D scalar functions". *Computer Graphics (San Diego Workshop on Volume Visualization)*, Vol. 24, No. 5, pp. 27–33, November 1990.

[Mitc88] Don P. Mitchell and Aru N. Netravali. "Reconstruction filters in computer graphics". *Computer Graphics (SIGGRAPH '88 Proceedings)*, Vol. 22, No. 4, pp. 221–228, August 1988.

[Naim89] Avi C. Naiman and Walter Makous. "Information transmission for grayscale edges". *Society for Information Display 91 Digest*, Vol. 22, pp. 109–112, May 1989.

[Newm79] William M. Newman and Robert F. Sproull. *Principles of Interactive Computer Graphics*. McGraw-Hill, second edition, 1979.

[Nort82] A. Norton, A.P. Rockwood, and P.T. Skolmoski. "Clamping: A method of antialiasing textured surfaces by bandwidth limiting in object space". *Computer Graphics (SIGGRAPH '82 Proceedings)*, Vol. 16, No. 3, pp. 1–8, July 1982.

[Novi90] Kevin L. Novins, Francois X. Sillion, and Donald P. Greenberg. "An efficient method for volume rendering using perspective projection". *Computer Graphics (San Diego Workshop on Volume Visualization)*, Vol. 24, No. 5, pp. 95–102, November 1990.

[Peac85] D.R. Peachey. "Solid texturing of complex surfaces". *Computer Graphics (SIGGRAPH '85 Proceedings)*, Vol. 19, No. 3, pp. 279–286, July 1985.

[Pent91] Alex Pentland and Bradley Horowitz. "A practical approach to fractal-based image compression". *Data Compression Conference*, pp. 176–185, April 8-11 1991.

[Perl85] K. Perlin. "An image synthesizer". *Computer Graphics (SIGGRAPH '85 Proceedings)*, Vol. 19, No. 3, pp. 287–296, July 1985.

[Perl89] Ken Perlin and Eric M. Hoffert. "Hypertexture". *Computer Graphics (SIGGRAPH '89 Proceedings)*, Vol. 23, No. 3, pp. 253–262, July 1989.

[Pern82] D. Perny, M. Gangnet, and P. Coueignoux. "Perspective mapping of planar textures". *Computer Graphics*, Vol. 16, No. 1, pp. 70–100, May 1982.

[Poul89] Pierre Poulin. "Anisotropic Reflection Models". M.Sc. Thesis, Department of Computer Science, University of Toronto, January 1989.

[Poul90] Pierre Poulin and Alain Fournier. "A model for anisotropic reflection". *Computer Graphics (SIGGRAPH '90 Proceedings)*, Vol. 24, No. 4, pp. 273–282, August 1990.

[Prat78] W.K. Pratt. *Digital Image Processing*. Wiley-Interscience, 1978.

[Rose75] A Rosenfeld. *Multiresolution Image Processing and Analysis*. Springer-Verlag, Berlin, 1975.

[Rose76] A. Rosenfeld and A. C. Kak. *Digital Picture Processing*. Computer Science and Applied Mathematics. Academic Press, 1976.

[Sabe88] Paolo Sabella. "A rendering algorithm for visualizing 3D scalar fields". *Computer Graphics (SIGGRAPH '88 Proceedings)*, Vol. 22, No. 4, pp. 51–58, August 1988.

[Schw83] D. Schweitzer. "Artificial texturing: An aid to surface visualization". *Computer Graphics (SIGGRAPH '83 Proceedings)*, Vol. 17, No. 3, pp. 23–29, July 1983.

[Shir90] Peter Shirley and Allan Tuchman. "A polygonal approximation to direct scalar volume rendering". *Computer Graphics (San Diego Workshop on Volume Visualization)*, Vol. 24, No. 5, pp. 63–70, November 1990.

[Tani75] S.L. Tanimoto and Theo Pavlidis. "A hiearchical data structure for picture processing". *Computer Vision, Graphics, and Image Processing*, Vol. 4, No. 2, June 1975.

[Tilt91] James C. Tilton, Deasso Han, and M Manohar. "Compression experiments with AVHRR data". *Data Compression Conference*, pp. 411–420, April 8-11 1991.

[Tots93] Takashi Totsuka and Marc Levoy. "Frequency domain volume rendering". *Computer Graphics (SIGGRAPH '93 Proceedings)*, pp. 271–278, August 1993.

[Trou87] Yves Trousset and Francis Schmitt. "Active ray tracing for 3d medical imaging". *Eurographics '87*, pp. 139–150, August 1987.

[Turk91] Greg Turk. "Generating textures on arbitrary surfaces using reaction-diffusion". *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, No. 4, pp. 289–298, July 1991.

[Upso88] Craig Upson and Michael Keeler. "V-BUFFER: Visible volume rendering". *Computer Graphics (SIGGRAPH '88 Proceedings)*, Vol. 22, No. 4, pp. 59–64, August 1988.

[Ward92] Gregory J. Ward. "Measuring and modeling anisotropic reflection". *Computer Graphics (SIGGRAPH '92 Proceedings)*, Vol. 26, No. 2, pp. 265–272, July 1992.

[West90] Lee Westover. "Footprint evaluation for volume rendering". *Computer Graphics (SIGGRAPH '90 Proceedings)*, Vol. 24, No. 4, pp. 367–376, August 1990.

[Wijk91] Jarke J. van Wijk. "Spot noise". *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, No. 4, pp. 309–318, July 1991.

[Wilh90a] Jane Wilhelms and Allen Van Gelder. "Octrees for faster isosurface generation extended abstract". *Computer Graphics (San Diego Workshop on Volume Visualization)*, Vol. 24, No. 5, pp. 57–62, November 1990.

[Wilh90b] Jane Wilhelms and Allen Van Gelder. "Topological considerations in isosurface generation extended abstract". *Computer Graphics (San Diego Workshop on Volume Visualization)*, Vol. 24, No. 5, pp. 79–86, November 1990.

[Wilh91] Jane Wilhelms and Allen Van Gelder. "A coherent projection approach for direct volume rendering". *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, No. 4, pp. 275–284, July 1991.

[Will83] L. Williams. "Pyramidal parametrics". *Computer Graphics (SIGGRAPH '83 Proceedings)*, Vol. 17, No. 3, pp. 1–11, July 1983.

[Witk91] Andrew Witkin and Michael Kass. "Reaction-diffusion textures". *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, No. 4, pp. 299–308, July 1991.

[Wu91] Xiaolin Wu and Chengfu Yao. "Image coding by adaptive tree-structured segmentation". *Data Compression Conference*, pp. 73–82, April 8-11 1991.

[Xue91] Kefu Xue and James M. Criseey. "An iteratively interpolative vector quantization algorithm for image data compression". *Data Compression Conference*, pp. 139–148, April 8-11 1991.

Appendix A
NIL innards

Get it running!

In Chapter 4 we overviewed the NIL map filter approximating technique. This overview was followed by a discussion of the extension of NIL maps to allow the use of Transfer functions, and procedural textures defined by a texture basis function set.

In one dimension we showed that the evaluation of the convolution of a filter $F(t)$ positioned at t_o and a texture signal $T(t)$

$$I(t_o) = \int_{-\infty}^{\infty} F(t - t_o)T(t)dt \tag{A.1}$$

could be approximated by first approximating the filter by a set of curves. This set of curves is piece-wise defined by basis functions $B_i(t)$. If these basis functions are pre-integrated with the texture function,

$$C_i^m = \int_0^1 B_i(t)T(m + t)dt \tag{A.2}$$

the above convolution can be approximated by

$$I(t_o) = \sum_{m=0}^{K-1} \sum_{i=0}^{M-1} b_i^m C_i^m. \tag{A.3}$$

The C_i^m coefficients are known as the NIL map cell entries. By spreading the basis functions and integrating these with successively larger intervals of the texture a pyramidal NIL map can be constructed. In this case the NIL map cells are defined by

$$C_i^{\gamma m} = \int_0^1 B_i(t)T(2^\gamma m + 2^\gamma t)dt, \tag{A.4}$$

where γ indicated the level of the pyramid to which the NIL cell belongs. In this appendix we discuss some of the optimizations which were employed in our implementation of NIL maps.

A-1 Speeding up the computation of C_{ijk} for discrete textures

The process that we are trying to approximate is that of the convolution of a three-dimensional filter with a continuous three-dimensional signal. Unfortunately, in many of our applications we do not have a continuous three-dimensional signal, instead we have a sampled version of the signal. Before we can apply any filters to this signal we must reconstruct the signal. There are many reconstruction filters that could be chosen, however, in practice one of two reconstruction methods are used: sample and hold or tri-linear interpolation. When we know the reconstruction filter we can speed up the computation of the NIL maps.

A-1.1 Sample and hold

The sample and hold reconstruction is computationally the simplest of the reconstruction filters available. The computation of the value of a particular point is found by applying the floor function to each of the coordinates. In the following discussion we will use $\overline{T}(x,y,z)$ to denote the reconstructed texture and $T(x,y,z)$ to denote the discrete texture.

$$\overline{T}(x,y,z) = T(\lfloor x \rfloor, \lfloor y \rfloor, \lfloor z \rfloor)$$

Using this reconstruction filter affords us two optimizations. The first is in the computation of the different levels of the NIL maps. The second optimization allows the inclusion of an arbitrary number of negative levels to the NIL map with out any storage cost.

Speeding up the computation of $\overset{\gamma}{C}{}^{mno}_{ijk}$

First lets look at this problem in one dimension. The texture $\overline{T}$ is sampled at $\Delta t = 1$ intervals [1] so the equation for $\overset{0}{C}{}^{m}_{i}$ is

$$\overset{0}{C}{}^{m}_{i} = \int_0^1 \overline{T}(a+t)B_i(t)dt \tag{A.5}$$

[1] We can assume WLOG $|\Delta t| = 1$, if this is not so a change of variables will suffice to ensure the condition.

since

$$\overline{T}(m+t)\Big|_{t\in[0,1)} \equiv T(m)$$

we rewrite A.5

$$\overset{0}{C}{}_i^m = \overline{T}(m)\int_0^1 B_i(t)dt$$

Now in general for any level γ we have

$$\int_0^1 B_i(t)\overline{T}(2^\gamma m + 2^\gamma t)dt = \sum_{a=0}^{2^\gamma-1}\int_{t=\frac{a}{2^\gamma}}^{\frac{a+1}{2^\gamma}} \overline{T}(2^\gamma m + a)B_i(t)dt \tag{A.6}$$

so that

$$\overset{\gamma}{C}{}_i^m = \sum_{a=0}^{2^\gamma-1}\int_{t=\frac{a}{2^\gamma}}^{\frac{a+1}{2^\gamma}} \overline{T}(2^\gamma m + a)B_i(t)dt \tag{A.7}$$

Since the texture function is now constant in every integral interval we rewrite

$$\overset{\gamma}{C}{}_i^m = \sum_{a=0}^{2^\gamma-1} \overline{T}(2^\gamma m + a)\int_{t=\frac{a}{2^\gamma}}^{\frac{a+1}{2^\gamma}} B_i(t)dt \tag{A.8}$$

Following this argument in three dimensions we find that for a texture $\overline{T}(u,v,w)$

$$\begin{aligned}\overset{\gamma}{C}{}_{ijk}^{mno} &= \sum_{a=0}^{2^\gamma-1}\sum_{b=0}^{2^\gamma-1}\sum_{c=0}^{2^\gamma-1}\int_{u=\frac{a}{2^\gamma}}^{\frac{a+1}{2^\gamma}}\int_{v=\frac{b}{2^\gamma}}^{\frac{b+1}{2^\gamma}}\int_{w=\frac{c}{2^\gamma}}^{\frac{c+1}{2^\gamma}} \overline{T}(2^\gamma m + a, 2^\gamma n + b, 2^\gamma o + c) \\ &\quad B_i(u)B_j(v)B_k(w)dudvdw\end{aligned} \tag{A.9}$$

When we examine equations A.9 and A.7 we notice that the integrals are always performed between constant sections of the texture and some fixed interval of the basis functions. This means that we can rewrite A.9 as follows:

$$\begin{aligned}\overset{\gamma}{C}{}_{ijk}^{mno} &= \sum_{a=0}^{2^\gamma-1}\sum_{b=0}^{2^\gamma-1}\sum_{c=0}^{2^\gamma-1} \overline{T}(2^\gamma m + a, 2^\gamma n + b, 2^\gamma o + c) \\ &\quad \int_{u=\frac{a}{2^\gamma}}^{\frac{a+1}{2^\gamma}}\int_{v=\frac{b}{2^\gamma}}^{\frac{b+1}{2^\gamma}}\int_{w=\frac{c}{2^\gamma}}^{\frac{c+1}{2^\gamma}} B_i(u)B_j(v)B_k(w)dudvdw\end{aligned} \tag{A.10}$$

Thus the computation of the C_{ijk} values should be done by precomputing for each

level γ the various integrals of the form

$$\overset{\gamma}{B}{}^{abc}_{ijk} = \int_{u=\frac{a}{2^\gamma}}^{\frac{a+1}{2^\gamma}} \int_{v=\frac{b}{2^\gamma}}^{\frac{b+1}{2^\gamma}} \int_{w=\frac{c}{2^\gamma}}^{\frac{c+1}{2^\gamma}} B_i(u)B_j(v)B_k(w)dudvdw \qquad \text{(A.11)}$$

$$\left| \begin{array}{l} a \in [0, 2^\gamma) \\ b \in [0, 2^\gamma) \\ c \in [0, 2^\gamma) \end{array} \right.$$

This is a separable integration so we can rewrite

$$\overset{\gamma}{B}{}^{abc}_{ijk} = \int_{u=\frac{a}{2^\gamma}}^{\frac{a+1}{2^\gamma}} B_i(u)du \int_{v=\frac{b}{2^\gamma}}^{\frac{b+1}{2^\gamma}} B_j(v)dv \int_{w=\frac{c}{2^\gamma}}^{\frac{c+1}{2^\gamma}} B_k(w)dw \qquad \text{(A.12)}$$

$$\left| \begin{array}{l} a \in [0, 2^\gamma) \\ b \in [0, 2^\gamma) \\ c \in [0, 2^\gamma) \end{array} \right.$$

Using these values equation A.10 becomes

$$\overset{\gamma}{C}{}^{mno}_{ijk} = \sum_{a=0}^{2^\gamma-1} \sum_{b=0}^{2^\gamma-1} \sum_{c=0}^{2^\gamma-1} \overline{T}(2^\gamma m + a, 2^\gamma n + b, 2^\gamma o + c)\ \overset{\gamma}{B}{}^{abc}_{ijk} \qquad \text{(A.13)}$$

From this discussion two methods suggest themselves for helping the computation of C_{ijk}, one memory intensive and the other less so.

The first method requires the computation of all the values for B^{abc}_{ijk} for each individual level γ. Since these arrays increase exponentially with the depth of the NIL map we must allocate enough memory to store the highest level of the B_{ijk} coefficients. The memory required at this level is

$$n^3 \times M^3$$

where the dimension of the data is $n \times n \times n$.
Noticing that these integrals are symmetric over any permutation of the indices a, b, c so we can make further computation and storage optimizations.

$$\frac{n(n+1)(2n+1)}{6} \times M^3$$

This can be shown by examining the levels of the array. If we set the first index to 1 then we see that $[1,b,c] = [1,c,b]$ is the symmetry that defines the first level. The storage required for this level is therefore

$$\frac{n(n+1)}{2}.$$

We no longer have to consider any triple that contains a 1 in it. Setting the first index to 2 and again using $[2,b,c] = [2,c,b]$ the storage required is

$$\frac{n(n+1)}{2} - 1 = \frac{n(n+1)}{2} - \frac{2(2-1)}{2}$$

Now setting the first level to i and noticing that we now do not consider any triple containing a number less than i the storage for level i then becomes

$$\frac{n(n+1)}{2} - \frac{(i-1)i}{2}.$$

Where the $\frac{(i-1)i}{2}$ factor accounts for those triples that contain a value less than i. The total required memory is the sum

$$\text{Storage} = \left[\sum_{i=1}^{n} \frac{n(n+1)}{2} - \frac{(i-1)(i)}{2}\right]$$

Using the identities

$$\begin{aligned}\sum_{i=1}^{n} i &= \frac{n(n+1)}{2} \\ \sum_{i=1}^{n} i^2 &= \frac{n(n+1)(2n+1)}{6}\end{aligned}$$

we have

$$\begin{aligned}\text{Storage} &= \left[\sum_{i=1}^{n} \frac{n(n+1)}{2} - \frac{(i-1)(i)}{2}\right] \\ &= \frac{n^2(n+1)}{2} - \frac{1}{2}\left[\frac{n(n+1)(2n+1)}{6} - \frac{n(n+1)}{2}\right]\end{aligned}$$

$$\begin{aligned} &= \frac{6n^2(n+1) - n(n+1)(2n+1) + 3n(n+1)}{12} \\ &= \frac{n(n+1)\left[6n - 2n - 1 + 3\right]}{12} \\ &= \frac{n(n+1)(4n+2)}{12} \\ &= \frac{n(n+1)(2n+1)}{6} \end{aligned} \tag{A.14}$$

The second method is a little bit more efficient in its memory usage. However, this efficiency is achieved at an increased computation cost. When we examine equation A.12 we notice that the basis integrals can be computed independently in the variables u, v, w, moreover because these integrals will be identical for each of u, v, and w. We need only store one table of integrals of the form.

$$\overset{\gamma}{B}{}_{i}^{a} \Big| \; a \in [0, 2^\gamma) = \int_{u=\frac{a}{2^\gamma}}^{\frac{a+1}{2^\gamma}} B_i(u)du \tag{A.15}$$

Using these variables the computation of $\overset{\gamma}{C}_{ijk}$ then becomes.

$$\overset{\gamma}{C}{}_{ijk}^{mno} = \sum_{a=0}^{2^\gamma-1} \sum_{b=0}^{2^\gamma-1} \sum_{c=0}^{2^\gamma-1} \overline{T}(2^\gamma m + a, 2^\gamma n + b, 2^\gamma o + c)\; \overset{\gamma}{B}{}_{i}^{a}\; \overset{\gamma}{B}{}_{j}^{b}\; \overset{\gamma}{B}{}_{k}^{c} \tag{A.16}$$

Negative levels for sample and hold

In many situations the required NIL map entry will be smaller than one of the voxels of data. Since we are using the sample and hold filter it is a simple matter to use negative levels for the NIL map. This is accomplished by noticing that the value of the texture over any of these negative NIL map entries is the same as that of the texture over the parent NIL map entry at level 0. Thus if we need to use one of these negative level NIL map entries we simply scale the value of the parent NIL map entry by the quantity.

$$\overset{-\gamma}{C}{}_{ijk}^{mno} = \text{parent}(\overset{-\gamma}{C}{}_{ijk}^{mno}) \tag{A.17}$$

A-1.2 Trilinear interpolation

When the reconstruction filter is a linear interpolation filter we can write the value of the texture

$$\overline{T}(m+t) = (1-t)T(m) + tT(m+1) \tag{A.18}$$

Using the above in equation A.5 we get.

$$\begin{aligned} \overset{0}{C}{}_i^m &= \int_0^1 \overline{T}(m+t)B_i(t)dt \\ &= \int_0^1 ((1-t)T(m) + tT(m+1))B_i(t)B_i(t)dt \end{aligned} \tag{A.19}$$

This integral can be separated.

$$\overset{0}{C}{}_i^m = \left(T(m)\int_0^1 B_i(t) - T(m)\int_0^1 tB_i(t) + T(m+1)\int_0^1 tB_i(t)dt\right)$$

Noticing that the integrals $\int_0^1 B_i(t)dt$ and $\int_0^1 tB(t)dt$ are independent of the sampled data, we define the quantities.

$$B_i = \int_0^1 B_i(t)dt \tag{A.20}$$

and

$${}^tB_i = \int_0^1 tB_i(t)dt \tag{A.21}$$

Extending this to fit into the NIL map scheme these quantities can be defined for arbitrary levels of γ.

$$\begin{aligned} \overset{\gamma}{B}{}_i^a &= \int_{t=\frac{a}{2^\gamma}}^{\frac{a+1}{2^\gamma}} B_i(t)dt \\ {}^t\overset{\gamma}{B}{}_i^a &= \int_{t=\frac{a}{2^\gamma}}^{\frac{a+1}{2^\gamma}} (2^\gamma t - a)B_i(t)dt \end{aligned} \tag{A.22}$$

So that the computation of C_i^m becomes

$$\overset{\gamma}{C}{}_i^m = \sum_{a=0}^{2^\gamma-1}\left[T(2^\gamma m + a)\ \overset{\gamma}{B}{}_i^a - T(2^\gamma m + a)\,{}^t\overset{\gamma}{B}{}_i^a + T(2^\gamma m + a + 1)\,{}^t\overset{\gamma}{B}{}_i^a\right]$$

A-1.3 Three dimensions

For a given texture T sampled on a grid we can express the reconstructed signal $\overline{T}$ as a weighted sum of the eight corners of the voxel.

$$\begin{aligned}\overline{T}(m+u,n+v,o+w) &= (1-w)[(1-v)[(1-u)T(m,n,o)+uT(m+1,n,o)]\\ &\quad +v[(1-u)T(m,n+1,o)+uT(m+1,n+1,o)]]\\ &\quad +w[(1-v)[(1-u)T(m,n,o+1)+uT(m+1,n,o+1)\\ &\quad +v[(1-u)T(m,n+1,o+1)+uT(m+1,n+1,o+1)]]\end{aligned}$$

Or expanded

$$\begin{aligned}&\overline{T}(m+u,n+v,o+w) =\\ &\quad T(m,n,o)+uT(m+1,n,o)+vT(m,n+1,o)+wT(m,n,o+1)\\ &\quad -uT(m,n,o)-vT(m,n,o)-wT(m,n,o)-uwT(m,n,o+1)\\ &\quad -uwT(m+1,n,o)-vwT(m,n,o+1)-vwT(m,n+1,o)-uvT(m+1,n,o)\\ &\quad -uvT(m,n+1,o)+vwT(m,n,o)+uvT(m,n,o)+uwT(m,n,o)\\ &\quad +uvT(m+1,n+1,o)+vwT(m,n+1,o+1)+uwT(m+1,n,o+1)\\ &\quad +uvwT(m+1,n,o)+uvwT(m,n+1,o)+uvwT(m,n,o+1)\\ &\quad -uvwT(m+1,n+1,o)-uvwT(m+1,n,o+1)-uvwT(m,n+1,o+1)\\ &\quad -uvwT(m,n,o)+uvwT(m+1,n+1,o+1)\end{aligned}$$

The calculation of C_{ijk}^{mno} then becomes

$$\begin{aligned}C_{ijk}^{mno} &= \int_0^1\int_0^1\int_0^1 T(m,n,o)B_i(u)B_j(v)B_k(w)dudvdw\\ &\quad +\int_0^1\int_0^1\int_0^1 uT(m+1,n,o)B_i(u)B_j(v)B_k(w)dudvdw\\ &\quad +\int_0^1\int_0^1\int_0^1 vT(m,n+1,o)B_i(u)B_j(v)B_k(w)dudvdw\end{aligned}$$

$$
\begin{aligned}
&+\int_0^1\int_0^1\int_0^1 wT(m,n,o+1)B_i(u)B_j(v)B_k(w)dudvdw\\
&-\int_0^1\int_0^1\int_0^1 uT(m,n,o)B_i(u)B_j(v)B_k(w)dudvdw\\
&-\int_0^1\int_0^1\int_0^1 vT(m,n,o)B_i(u)B_j(v)B_k(w)dudvdw\\
&-\int_0^1\int_0^1\int_0^1 wT(m,n,o)B_i(u)B_j(v)B_k(w)dudvdw\\
&-\int_0^1\int_0^1\int_0^1 uwT(m,n,o+1)B_i(u)B_j(v)B_k(w)dudvdw\\
&-\int_0^1\int_0^1\int_0^1 uwT(m+1,n,o)B_i(u)B_j(v)B_k(w)dudvdw\\
&-\int_0^1\int_0^1\int_0^1 vwT(m,n,o+1)B_i(u)B_j(v)B_k(w)dudvdw\\
&-\int_0^1\int_0^1\int_0^1 vwT(m,n+1,o)B_i(u)B_j(v)B_k(w)dudvdw\\
&-\int_0^1\int_0^1\int_0^1 uvT(m+1,n,o)B_i(u)B_j(v)B_k(w)dudvdw\\
&-\int_0^1\int_0^1\int_0^1 uvT(m,n+1,o)B_i(u)B_j(v)B_k(w)dudvdw\\
&+\int_0^1\int_0^1\int_0^1 vwT(m,n,o)B_i(u)B_j(v)B_k(w)dudvdw\\
&+\int_0^1\int_0^1\int_0^1 uvT(m,n,o)B_i(u)B_j(v)B_k(w)dudvdw\\
&+\int_0^1\int_0^1\int_0^1 uwT(m,n,o)B_i(u)B_j(v)B_k(w)dudvdw\\
&+\int_0^1\int_0^1\int_0^1 uvT(m+1,n+1,o)B_i(u)B_j(v)B_k(w)dudvdw\\
&+\int_0^1\int_0^1\int_0^1 vwT(m,n+1,o+1)B_i(u)B_j(v)B_k(w)dudvdw\\
&+\int_0^1\int_0^1\int_0^1 uwT(m+1,n,o+1)B_i(u)B_j(v)B_k(w)dudvdw\\
&+\int_0^1\int_0^1\int_0^1 uvwT(m+1,n,o)B_i(u)B_j(v)B_k(w)dudvdw\\
&+\int_0^1\int_0^1\int_0^1 uvwT(m,n+1,o)B_i(u)B_j(v)B_k(w)dudvdw\\
&+\int_0^1\int_0^1\int_0^1 uvwT(m,n,o+1)B_i(u)B_j(v)B_k(w)dudvdw\\
&-\int_0^1\int_0^1\int_0^1 uvwT(m+1,n+1,o)B_i(u)B_j(v)B_k(w)dudvdw
\end{aligned}
$$

$$
\begin{aligned}
&- \int_0^1 \int_0^1 \int_0^1 uvwT(m+1,n,o+1)B_i(u)B_j(v)B_k(w)dudvdw \\
&- \int_0^1 \int_0^1 \int_0^1 uvwT(m,n+1,o+1)B_i(u)B_j(v)B_k(w)dudvdw \\
&- \int_0^1 \int_0^1 \int_0^1 uvwT(m,n,o)B_i(u)B_j(v)B_k(w)dudvdw \\
&+ \int_0^1 \int_0^1 \int_0^1 uvwT(m+1,n+1,o+1)B_i(u)B_j(v)B_k(w)dudvdw
\end{aligned}
$$

Defining some short hand notations

$$
\begin{aligned}
B_{ijk} &= \int_0^1 \int_0^1 \int_0^1 B_i(u)B_j(v)B_k(w)dudvdw \\
{}^{u}B_{ijk} &= \int_0^1 \int_0^1 \int_0^1 uB_i(u)B_j(v)B_k(w)dudvdw \\
{}^{v}B_{ijk} &= \int_0^1 \int_0^1 \int_0^1 vB_i(u)B_j(v)B_k(w)dudvdw \\
{}^{w}B_{ijk} &= \int_0^1 \int_0^1 \int_0^1 wB_i(u)B_j(v)B_k(w)dudvdw \\
{}^{uv}B_{ijk} &= \int_0^1 \int_0^1 \int_0^1 uvB_i(u)B_j(v)B_k(w)dudvdw \\
{}^{uw}B_{ijk} &= \int_0^1 \int_0^1 \int_0^1 uwB_i(u)B_j(v)B_k(w)dudvdw \\
{}^{vw}B_{ijk} &= \int_0^1 \int_0^1 \int_0^1 vwB_i(u)B_j(v)B_k(w)dudvdw \\
{}^{uvw}B_{ijk} &= \int_0^1 \int_0^1 \int_0^1 uvwB_i(u)B_j(v)B_k(w)dudvdw
\end{aligned}
$$

and because the texture values are constant over the integrals we have

$$
\begin{aligned}
&C_{ijk}^{mno} = \\
&\quad T(m,n,o)B_{ijk} + T(m+1,n,o){}^{u}B_{ijk} + T(m,n+1,o){}^{v}B_{ijk} \\
&\quad + T(m,n,o+1){}^{w}B_{ijk} - T(m,n,o){}^{u}B_{ijk} - T(m,n,o){}^{v}B_{ijk} \\
&\quad - T(m,n,o){}^{w}B_{ijk} - T(m,n,o+1){}^{uw}B_{ijk} - T(m+1,n,o){}^{uw}B_{ijk} \\
&\quad - T(m,n,o+1){}^{vw}B_{ijk} - T(m,n+1,o){}^{vw}B_{ijk} - T(m+1,n,o){}^{uv}B_{ijk} \\
&\quad - T(m,n+1,o){}^{uv}B_{ijk} + T(m,n,o){}^{vw}B_{ijk} + T(m,n,o){}^{uv}B_{ijk} \\
&\quad + T(m,n,o){}^{uw}B_{ijk} + T(m+1,n+1,o){}^{uv}B_{ijk} + T(m,n+1,o+1){}^{vw}B_{ijk} \\
&\quad + T(m+1,n,o+1){}^{uw}B_{ijk} + T(m+1,n,o){}^{uvw}B_{ijk} + T(m,n+1,o){}^{uvw}B_{ijk}
\end{aligned}
$$

$$
\begin{aligned}
&+T(m,n,o+1)^{uvw}B_{ijk} - T(m+1,n+1,o)^{uvw}B_{ijk} - T(m+1,n,o+1)^{uvw}B_{ijk} \\
&-T(m,n+1,o+1)^{uvw}B_{ijk} - T(m,n,o)^{uvw}B_{ijk} \\
&+T(m+1,n+1,o+1)^{uvw}B_{ijk}
\end{aligned} \tag{A.23}
$$

It is simple to show that

$$
\begin{aligned}
{}^{u}B_{ijk} &= {}^{v}B_{jik} = {}^{w}B_{kij} \\
{}^{uv}B_{ijk} &= {}^{uw}B_{ikj} = {}^{vw}B_{kij}
\end{aligned} \tag{A.24}
$$

Using these identities equation A.23 becomes.

$$
\begin{aligned}
&C_{ijk}^{mno} = \\
&\quad T(m,n,o)B_{ijk} + T(m+1,n,o)^{u}B_{ijk} + T(m,n+1,o)^{u}B_{jik} \\
&\quad +T(m,n,o+1)^{u}B_{kij} - T(m,n,o)^{u}B_{ijk} - T(m,n,o)^{u}B_{jik} \\
&\quad -T(m,n,o)^{u}B_{kij} - T(m,n,o+1)^{uv}B_{ikj} - T(m+1,n,o)^{uv}B_{ikj} \\
&\quad -T(m,n,o+1)^{uv}B_{kij} - T(m,n+1,o)^{uv}B_{kij} - T(m+1,n,o)^{uv}B_{ijk} \\
&\quad -T(m,n+1,o)^{uv}B_{ijk} + T(m,n,o)^{uv}B_{kij} + T(m,n,o)^{uv}B_{ijk} \\
&\quad +T(m,n,o)^{uv}B_{ikj} + T(m+1,n+1,o)^{uv}B_{ijk} + T(m,n+1,o+1)^{uv}B_{kij} \\
&\quad +T(m+1,n,o+1)^{uv}B_{ikj} + T(m+1,n,o)^{uvw}B_{ijk} + T(m,n+1,o)^{uvw}B_{ijk} \\
&\quad +T(m,n,o+1)^{uvw}B_{ijk} - T(m+1,n+1,o)^{uvw}B_{ijk} - T(m+1,n,o+1)^{uvw}B_{ijk} \\
&\quad -T(m,n+1,o+1)^{uvw}B_{ijk} - T(m,n,o)^{uvw}B_{ijk} \\
&\quad +T(m+1,n+1,o+1)^{uvw}B_{ijk}
\end{aligned} \tag{A.25}
$$

Rearranging yields

$$
\begin{aligned}
C_{ijk}^{mno} &= T(m,n,o)\left[B_{ijk} - {}^{u}Bijk - {}^{u}B_{jik} - {}^{u}B_{kji} + {}^{uv}B_{kij} + {}^{uv}B_{ijk} + {}^{uv}B_{ikj}\right] \\
&\quad +T(m+1,n,o)\left[{}^{u}B_{ijk} - {}^{uv}B_{ikj} - {}^{uv}B_{ijk} + {}^{uvw}B_{ijk}\right] \\
&\quad +T(m,n+1,o)\left[{}^{u}B_{jik} - {}^{uv}B_{kij} - {}^{uv}B_{ijk} + {}^{uvw}B_{ijk}\right] \\
&\quad +T(m,n,o+1)\left[{}^{u}B_{kij} - {}^{uv}B_{ikj} - {}^{uv}B_{kij} + {}^{uvw}B_{ijk}\right] \\
&\quad +T(m+1,n+1,o)\left[{}^{uv}B_{ijk} - {}^{uvw}B_{ijk}\right] \\
&\quad +T(m+1,n,o+1)\left[{}^{uv}B_{ikj} - {}^{uvw}B_{ijk}\right] \\
&\quad +T(m,n+1,o+1)\left[{}^{uv}B_{kij} - {}^{uvw}B_{ijk}\right] \\
&\quad +T(m+1,n+1,o+1)\left[{}^{uvw}B_{ijk}\right]
\end{aligned} \tag{A.26}
$$

So defining

$$\begin{aligned}
{}^{000}B_{ijk} &= [B_{ijk} - {}^{u}B_{ijk} - {}^{u}B_{jik} - {}^{u}B_{kji} + {}^{uv}B_{kij} + {}^{uv}B_{ijk} + {}^{uv}B_{ikj}] \\
{}^{100}B_{ijk} &= [{}^{u}B_{ijk} - {}^{uv}B_{ikj} - {}^{uv}B_{ijk} + {}^{uvw}B_{ijk}] \\
{}^{010}B_{ijk} &= [{}^{u}B_{jik} - {}^{uv}B_{kij} - {}^{uv}B_{ijk} + {}^{uvw}B_{ijk}] \\
{}^{001}B_{ijk} &= [{}^{u}B_{kij} - {}^{uv}B_{ikj} - {}^{uv}B_{kij} + {}^{uvw}B_{ijk}] \\
{}^{110}B_{ijk} &= [{}^{uv}B_{ijk} - {}^{uvw}B_{ijk}] \\
{}^{101}B_{ijk} &= [{}^{uv}B_{ikj} - {}^{uvw}B_{ijk}] \\
{}^{011}B_{ijk} &= [{}^{uv}B_{kij} - {}^{uvw}B_{ijk}] \\
{}^{111}B_{ijk} &= [{}^{uvw}B_{ijk}]
\end{aligned} \tag{A.27}$$

The computation of C_{ijk}^{mno} becomes

$$C_{ijk}^{mno} = \sum_{x=0}^{1}\sum_{y=0}^{1}\sum_{z=0}^{1} {}^{xyz}B_{ijk}T(m+x, n+y, o+z) \tag{A.28}$$

These integrals ${}^{uvw}B_{ijk}$ can be precomputed. In the case of sample and hold we found that computing the quantities $\overset{\gamma}{B}{}_{ijk}^{abc}$ reduced the precomputation cost somewhat because the integrals of the basis functions were not being evaluated more than once. The memory cost associated with this was equivalent to the bottom most level of the NIL map. For tri-linear interpolation the memory cost for the precomputed integrals is going to be eight times that of the bottom most level of the NIL map so we evaluated the higher levels of the NIL maps using the following definition.

$$\begin{aligned}
\overset{\gamma}{C}{}_{ijk}^{mno} &= \sum_{a=0}^{2^\gamma-1}\sum_{b=0}^{2^\gamma-1}\sum_{c=0}^{2^\gamma-1}\int_{u=\frac{a}{2^\gamma}}^{\frac{a+1}{2^\gamma}}\int_{v=\frac{b}{2^\gamma}}^{\frac{b+1}{2^\gamma}}\int_{w=\frac{c}{2^\gamma}}^{\frac{c+1}{2^\gamma}} \overline{T}(2^\gamma m + a, 2^\gamma n + b, 2^\gamma o + c) \\
&\quad B_i(u)B_j(v)B_k(w)dudvdw
\end{aligned} \tag{A.29}$$

A-1.4 Negative levels for tri-linear interpolation

The negative levels for the NIL map in this case are computed by using an altered version of equation A.26. In the following equation $T(m^p, n^p, o^p)$ refers to the texture value in the lower corner of the parent cell, and the references to $T(m^p + x, n^p + x, o^p + x)$ refer to the interpolated texture value evaluated at $(m^p + x, n^p + x, o^p + x)$

$$
\begin{aligned}
&\bar{C}^{\gamma}_{ijk}{}^{mno} = \\
&\quad \overline{T}(m^p + \frac{m}{2^\gamma}, n^p + \frac{n}{2^\gamma}, o^p + \frac{o}{2^\gamma})\,[B_{ijk} - {}^{u}Bijk - {}^{u}B_{jik} - {}^{u}B_{kji} + {}^{uv}B_{kij} + {}^{uv}B_{ijk} + {}^{uv}B_{ikj}] \\
&\quad + \overline{T}(m^p + \frac{m+1}{2^\gamma}, n^p + \frac{n}{2^\gamma}, o^p + \frac{o}{2^\gamma})\,[{}^{u}B_{ijk} - {}^{uv}B_{ikj} - {}^{uv}B_{ijk} + {}^{uvw}B_{ijk}] \\
&\quad + \overline{T}(m^p + \frac{m}{2^\gamma}, n^p + \frac{n+1}{2^\gamma}, o^p + \frac{o}{2^\gamma})\,[{}^{u}B_{jik} - {}^{uv}B_{kij} - {}^{uv}B_{ijk} + {}^{uvw}B_{ijk}] \\
&\quad + \overline{T}(m^p + \frac{m}{2^\gamma}, n^p + \frac{n}{2^\gamma}, o^p + \frac{o+1}{2^\gamma})\,[{}^{u}B_{kij} - {}^{uv}B_{ikj} - {}^{uv}B_{kij} + {}^{uvw}B_{ijk}] \\
&\quad + \overline{T}(m^p + \frac{m+1}{2^\gamma}, n^p + \frac{n+1}{2^\gamma}, o^p + \frac{o+1}{2^\gamma})\,[{}^{uv}B_{ijk} - {}^{uvw}B_{ijk}] \\
&\quad + \overline{T}(m^p + \frac{m+1}{2^\gamma}, n^p + \frac{n}{2^\gamma}, o^p + \frac{o+1}{2^\gamma})\,[{}^{uv}B_{ikj} - {}^{uvw}B_{ijk}] \\
&\quad + \overline{T}(m^p + \frac{m}{2^\gamma}, n^p + \frac{n+1}{2^\gamma}, o^p + \frac{o+1}{2^\gamma})\,[{}^{uv}B_{kij} - {}^{uvw}B_{ijk}] \\
&\quad + \overline{T}(m^p + \frac{m+1}{2^\gamma}, n^p + \frac{n+1}{2^\gamma}, o^p + \frac{o+1}{2^\gamma})\,[{}^{uvw}B_{ijk}]
\end{aligned}
\tag{A.30}
$$

Using this definition of the negative levels we can compute them on the fly at the cost of 8 texture evaluations and the above sum.

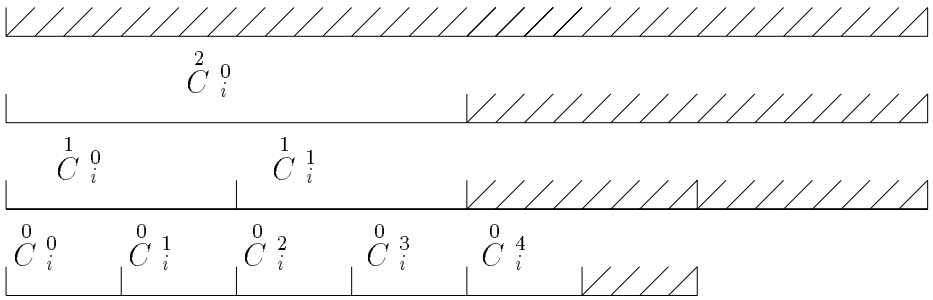
A-2 Non symmetric cases

In the preceding discussion we have assumed that the dimensions of the digitized data is of the form 2^p for some integer p. We have also assumed that in the three dimensional cases $x_{res} = y_{res} = z_{res}$. In this section we present solutions for these cases. First, we present a solution to the problem of not having the dimension of the data as an integral power of 2. Second, we extend this solution to solve the problem of not having three dimensional data of equal dimensions.

A-2.1 Non integral powers of 2 in one dimension

Since the algorithm that we are using depends on halving the dimensions of the data for each level of the NIL maps it becomes a problem when we try to use this technique on data that has been sampled inadequately. One of the solutions is to reconstruct the signal

and resample it. If we do not wish to do this we must extend the NIL map technique to handle arbitrarily sized data as efficiently as possible. The technique proposed here allows the use of the NIL map technique on such data. We first extend the data structure that we used for storing the NIL map values, by adding a bit value, *valid*, that indicates whether the current entry in the NIL map is valid or not.



Building NIL maps with samples where $n \neq 2^p$

Figure A.24

The construction of the NIL maps now requires a parity check on the size of each level. If the level is odd we add a NIL map cell and mark it as invalid. The dimension of the next level is then half of the size of this extended level. Each of the NIL cells on this level is evaluated unless one (or both) of its children are invalid. If this is the case then we simply mark this cell as invalid. The parity of this level is then checked and an additional cell is added if necessary. This process continues until we have constructed a complete NIL map. Such a NIL map is illustrated in Figure A.24. The NIL map in this figure corresponds to a NIL map constructed from five initial samples. The lowest level ($\gamma = 0$) has had one NIL cell added to it. The second level ($\gamma = 1$) has two invalid cells. The leftmost invalid cell is invalid because one of its children is invalid. The rightmost invalid cell is invalid because it was added to ensure that this level had an even number of cells. The third level has one invalid cell because both of its children are invalid, and the fourth level has one invalid cell because one of its children is invalid.

The extra storage cost incurred by the above alterations is at most $\lceil \log_2(n) \rceil$, because at each level we can only add one additional cell and there are at most $\lceil \log_2(n) \rceil$ levels

in this NIL map.

The process of generating the approximating hierarchy must now be altered slightly. In the subdivision process that we described earlier we subdivided a NIL cell if there were more trigger points in it that allowed by the tolerance. The new subdivision process is similar to this except that when an invalid patch is encountered it is automatically subdivided regardless of how many trigger points it contains. In this manner we ensure that the subdivision produces a hierarchy of valid patches. This process will require some additional subdivisions to be performed.

The upper bound on the number of additional subdivisions is easily derived. Consider the case where a single trigger point is positioned so that it lies in an invalid NIL cell for each level of the NIL map except for the lowest level. This trigger point will cause at most $\lceil \log_2(n) \rceil$ subdivisions. No other trigger point will cause these subdivisions. This implies that using the above extension to NIL maps will require at most $\lceil \log_2(n) \rceil$ additional subdivisions.

A-2.2 Rectangular three-dimensional textures

In this section we will explore the case of a texture which has dimensions $x_{res} = 2^{p_x} < y_{res} = 2^{p_y} \leq z_{res} = 2^{p_z}$. We wish to develop a NIL map representation of these textures such that for some level γ_m there is only one entry in the corresponding level of the NIL map. If we blindly apply the NIL map algorithm we will find that at level $\gamma = p_x$ the number of NIL map entries in the x direction is one but because $p_x < p_y \leq p_z$ we have $2^{p_y - p_x}$ NIL map entries in the y and z direction.

We wish to continue building the NIL map until there is 1 NIL map entry in the top level of the NIL map. This means we must continue building for $p_z - p_x$ levels. This is simply a matter of keeping track of these *special* levels. We must however modify the equations we use for the calculation of the $\overset{\gamma}{B}{}^{\,abc}_{ijk}$, $\overset{\gamma}{C}{}^{\,mno}_{ijk}$, and $\overset{\gamma}{D}{}^{\,mno}_{ijkl}$ coefficients. The calculation of $\overset{\gamma}{B}{}^{\,abc}_{ijk}$, detailed in equation A.12, becomes

$$\overset{\gamma}{B}{}^{\,abc}_{ijk} = \left(\frac{1}{2^{(p_x - \min(p_x,\gamma))}}\right) \int_{u=\frac{a}{2^\gamma}}^{\frac{a+1}{2^\gamma}} B_i(u)du \left(\frac{1}{2^{(p_y - \min(p_y,\gamma))}}\right) \int_{v=\frac{b}{2^\gamma}}^{\frac{b+1}{2^\gamma}} B_j(v)dv$$

$$\left(\frac{1}{2^{(p_z - min(p_y,\gamma))}}\right) \int_{w=\frac{c}{2^\gamma}}^{\frac{c+1}{2^\gamma}} B_k(w)dw \tag{A.31}$$
$$a \in [0, 2^{\min(p_x,\gamma)}), b \in [0, 2^{\min(p_y,\gamma)}), c \in [0, 2^{\min(p_z,\gamma)})$$

Equation A.13 then becomes

$$\overset{\gamma}{C}{}_{ijk}^{mno} = \sum_{a=0}^{2^{\min(p_x,\gamma)}-1} \sum_{b=0}^{2^{\min(p_y,\gamma)}-1} \sum_{c=0}^{2^{\min(p_z,\gamma)}-1} \overline{T}(2^{\min(p_x,\gamma)}m + a, 2^{\min(p_y,\gamma)}n + b, 2^{\min(p_z,\gamma)}o + c)\ \overset{\gamma}{B}{}_{ijk}^{abc} \tag{A.32}$$

similarly the computation of $\overset{\gamma}{D}{}_{ijkl}^{mno}$ becomes

$$\overset{\gamma}{D}{}_{ijkl}^{mno} = \sum_{a=0}^{2^{\min(p_x,\gamma)}-1} \sum_{b=0}^{2^{\min(p_y,\gamma)}-1} \sum_{c=0}^{2^{\min(p_z,\gamma)}-1} B_l(\overline{T}(2^{\min(p_x,\gamma)}m + a, 2^{\min(p_y,\gamma)}n + b, 2^{\min(p_z,\gamma)}o + c))\ \overset{\gamma}{B}{}_{ijk}^{abc} \tag{A.33}$$

Appendix B
High level view of NIL code

In this appendix we present the listing of the loop from our NIL map implementation for volume rendering. Much of the book-keeping code has been removed to present a cleaner view of the code. This code listing shows the inner-loops which compute the pixel intensities.

```
/*
 * For each pixel on the screen calculate the ray which it
 * generates.   Use this ray to lay the trigger points and/or find
 * the closest density
 */

for (pix_j = bottomY; pix_j <= topY; pix_j++)                    for
        {
        for (pix_i = leftX; pix_i<= rightX; pix_i++) /* for I */
                {                                                10
                ray = GenerateRay ((float)pix_i,
                                   (float)pix_j, &up, &right);

                /*
                 * Get the entry and exit points for the volume Use the
                 * voxel traversal stuff for going through the data and
                 * collect the 4 parameters.
                 */
                hits = get_entry_exit_points(ray,&v1,&v2, &t1, &t2);
                if (hits == 1)                                   20
                        {
                        hits = get_entry_exit_points(ray,&v1,&v2, &t1, &t2);
                        hits = 0;
                        }
                /* Hit detected */
                if (hits != 0)
```

```
    {
    trigger_points = lay_down_the_sample_points(v1,v2,
            t1,t2,up,right,ray,selected_trigger_points,
            slices_required,samples_per_slice,length);
    box = find_the_box_which_encloses_the_points(
              selected_trigger_points,trigger_points);

    number_of_patches = find_the_required_patches(
            selected_patches,box,
            selected_trigger_points,trigger_points,
            max_gamma);

/*
 * First find the control weights of the control points
 * for all the patches.  Use the sum of their weights to
 * normalize the display
 */

    for(i=0; i< number_of_patches; i++)
            {
            scale += find_the_weight_of_the_control_points(
                &selected_patches[i],max_gamma,x,y,
                pix_i,pix_j,volume_bottom, volume_dx,volume_dy,
                volume_dz,center);
            }

    for(i=0; i< number_of_patches; i++)
            {
            patch_total = find_the_integral_under_the_patch(
                &selected_patches[i]) ;
            total += patch_total;
            }

    total = total/scale;
    pixel=UNIT_TO_CHAR(total_red);

    (*current_frame_buffer->setpixel)(pix_i,pix_j,pixel);
    } /* Hits != 0*/
```

```
            } /* for I */
} /* for J */
```

70

Appendix C

Motion blur filter

C code

The addition of motion blur to the texture map filtering code required the addition of the following code.

```
/*
 *      Return the value of the filter at point.
 *      input:
 *      point                   Vector containing the control point.
 *      Box_center_length       Radial distance of centre of filter.
 *      Box_width_for_rotate    Width of filter.
 *
 */
double Filter(vector point, double Box_center_length, double Box_width_for_rotate)
        {                                                                    10
        double l = sqrt(point.x*point.x + point.y*point.y);
        double Q = (Box_center_length -l)*(Box_center_length -l);
        return(exp(-Q/Box_width_for_rotate));
        }

/*
 *      Place points around the arc near which the filter is defined.
 *      input:
 *              cB              Texture coordinate for centre of pixel projection.
 *              a               Long axis of projected ellipse                  20
 *              points[]        Array of trigger points.
 *              nil_samples     User parameter indicating how many samples.
 *              rotation        Angle of rotation in radians
 */
int Lay_Down_Rotate_Points(vector cB, double a, vector points[],
                           int nil_samples,double rotation)
        {
        double rot_delta=rotation/(double)nil_samples;
        double angle;
```

```
double start_angle;
double tmp_length;
double offset = rot_delta/nil_samples;
int number_of_points;

start_angle = atan2(cB.y,cB.x);
Box_center_length = sqrt(cB.y*cB.y+cB.x*cB.x);
Box_width_for_rotate = a;
for(i=0; i< nil_samples; i++)
        {
        angle = start_angle + i*rot_delta;
        for(j=0; j< nil_samples; j++)
                {
                double tmp_length = Box_center_length −a
                                + 2.0*a/nil_samples *j;
                X.x = tmp_length * cos(angle+j*offset);
                X.y = tmp_length * sin(angle+j*offset);

                X.z = −0.01;
                points[number_of_points] = X;
                number_of_points++;
                X.z = 0.0;
                points[number_of_points] = X;
                number_of_points++;
                X.z = 0.01;
                points[number_of_points] = X;
                number_of_points++;
                }
        }
return(number_of_points);
}
```

Index