c 1

Multi-resolution Surface Approximation for Animation

By

LIFENG WANG

B.Sc., University of Science and Technology of China

China, 1990

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(DEPARTMENT OF COMPUTER SCIENCE)

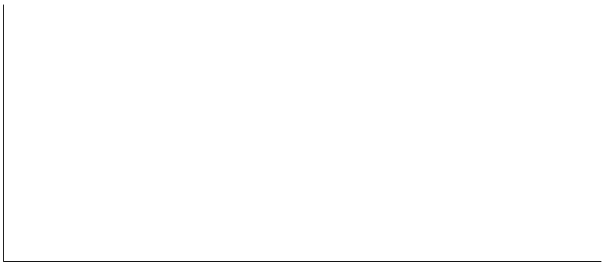
We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

April, 1993

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

(Signature) 

Department of Computer Science

The University of British Columbia
Vancouver, Canada

Date April 28th, 1993

DE-6 (2/88)

Abstract

This thesis addresses the problem of approximating a set of gridded data points obtained from a three-dimensional digitizing system to create a representation with a hierarchical bicubic B-spline surface that is suitable for further manipulation and animation. Chord length parameterization is obtained using 2-D deformation technique.

A full multigrid (FMG) numerical method is used to solve the surface approximation and the multi-resolution elements created are used directly to define the overlays in a hierarchical B-spline surface. The direct use of FMG multi-resolution data offers reasonable surface shape behaviour, but the number of non-zero offsets is large. Storage cost is reduced either by eliminating offsets whose magnitude is below a certain tolerance or by reducing all offsets in a given level by a user specified amount. The resulting spline surface is modifiable, both locally and globally while retaining surface details of the digitized data.

An interactive system based on these methods was created and the results of approximating two large data sets are presented.

Contents

List of Figures

Acknowledgements

I would like to thank Dr. David R. Forsey, my thesis supervisor, for his guidance and encouragement throughout my work on this thesis. The valuable ideas and suggestions from him improved the content and presentation of this thesis significantly.

I would also like to thank the second reader of my thesis, Dr.Peter Cahoon for reading through the draft of this thesis, his comments and help made the thesis complete.

Karen Kuder read the first draft of my thesis and spent a lot of time changing my grammar and spelling error. I feel much appreciated for her very great care on reading my thesis. Dr. Pierre Poulin volunteered to read the draft paper presented on GI93 which is based on this thesis. His comments have helped a lot on improving my work. Many other people in the department offered time and effort my work. It is difficult to list all the individuals who has been involved with me. But those people who have provided help know that they deserved a note of THANKS.

If this is the chance, I would like to direct the most important acknowledgement to my parents, who have been providing priceless love and support over the past. Without their encouragement and comforts, most of the opportunities I enjoyed would not have been possible, including the opportunity to come to Canada to complete the work I present here.

Chapter 1

Introduction

1.1 Overall Goal

This research is a part of a long term project to create a system for the animation and rendering of human animal forms. An integral part of this system is a mechanism whereby digitized surface information is used to determine the surface characteristics (shape and colour) of an animated figure. The underlying surface formulation for this system is hierarchical B-spline [17] [20], a multi-resolution approach to modelling with tensor product surfaces. Thus the mechanisms for approximating an arbitrary surface definition with a hierarchical B-spline surface are required.

This thesis takes the first step toward this goal by addressing the sub-problem of approximating regular gridded data with a hierarchical spline. The contribution of this work is the use of a multi-resolution approach to solving system of linear equations (the full multigrid method, i.e. FMG) to directly generate the multi-resolution surface of hierarchical B-spline. Surface approximation for animation adds one additional complication. Because the approximating surface may only be a part of a much larger surface, the question of what part of the surface approximates what part of the data must also be addressed.

1.2 Surface Approximation

1.2.1 Mathematical Formulation

We are interested in fitting a tensor-product B-spline surface

$$S(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} V_{i,j} B_{i,k}(u) B_{j,l}(v) \tag{1.1}$$

to a given set of data points.

To generate the equations for surface approximation, each data point $(x,y,z) = D_\lambda \in R^3$ is associated with a domain point $(u,v) = \delta_\lambda \in R^2$ of the spline. This forms the equation:

$$\sum_{i=0}^{m}\sum_{j=0}^{n} V_{i,j} B_{i,k}(u_i) B_{j,l}(v_j) = D_\lambda. \tag{1.2}$$

Data is *gridded* if

$$\begin{aligned} u &\in \{u_0, \cdots, u_M\} \\ v &\in \{v_0, \cdots, v_N\} \end{aligned} \tag{1.3}$$

and if the δs consist of all points in $\{u_0, \cdots, u_M\} \times \{v_0, \cdots, v_N\}$ the surface approximation equations become

$$\sum_{i=0}^{m}\sum_{j=0}^{n} V_{i,j} B_{i,k}(u_s) B_{j,l}(v_t) = D_{q,r}. \tag{1.4}$$

for $s = 0, \cdots, M$ and $t = 0, \cdots, N$. Thus the number of equations is equal to the number of data points.

Hierarchical splines are a multi-resolution approach to splines for use in interactive creation of free-form surfaces [17] [40]. The problem of applying these equations to approximate a digitized surface with a hierarchical spline is complicated by two factors: an unusual parameterization, and the multi-resolution nature of the hierarchical formulation itself.

Readers who are familiar with parameterization and surface fitting can skip the following sections and go directly to Chapter 3.

1.2.2 Discrete Data

Systematic measuring devices such as laser scanners, imaging systems and optical scanners are used to measure the positions of points on an object's surface with a resolution high enough

to provide an accurate representation of the original object. These devices work in three dimensional space and typically produce a 2-D array of data points. A medical imaging system like CT (Computed Tomography) records both exterior and interior information, producing a regularly spaced 3-D lattice.

Raw data sets for human figures are usually too large to manipulate in real-time on general purpose graphics workstations. Techniques used to approximate the raw data must be very efficient both in terms of computing time and storage complexity.

1.2.3 Interpolation and Approximation

A common technique for recreating the smooth surface from a set of discrete digitized data points discrete data set as a continuous shape is a graph. Other techniques include fitting surfaces (interpolation and approximation), volume rendering and various visualization techniques.

There are basically two ways to reconstruct the continuous model from digitized data: interpolation and approximation. Interpolation requires that the surface passes through every data point. Piecewise parametric polynomial interpolation is a common technique [14] and there are various other mathematical formulations which meet different requirements [4] [16] [24]. Most interpolation methods are simple in concept, have unique solutions and provide an accurate representation of the original data.

Approximation is designed to fairly represent the data without necessarily interpolating the individual data points and are often useful for data sets comprised of measured values including errors where their exact interpolation is not desired. Approximate algorithms can be adaptive or non-adaptive. Adaptive algorithms begin with a simple approximation defined by a small number of coefficients. The number of degrees of freedom in the approximation is adaptively increased in areas of high error until the surface is within a given tolerance of the data. Adaptive algorithms are efficient because some sub-regions of data have less gradient variation than other regions. The smoother regions can be approximated with coarse meshes without exceeding certain error tolerances, reducing the storage cost. The bumpy regions should be approximated with fine meshes.

There are numerous approaches to both approximation and interpolation. Each mathematical representation will produce different approximations for the same set of data. Piecewise parametric surfaces are one family of tools for approximating shapes [14]. Notable examples of such formulations include Bézier curves [14], Coons patches [11], and B-splines [3]. These representations have been used extensively throughout the CAGD field to describe shape in computer systems. Some mathematical background concerning these representations will be given in Chapter 2. More details can be found in books by Farin [14] and by Bartels et. al. [3].

1.3 Problem Definition

1.3.1 Test Data

The data set chosen for use in this thesis is representative of the size and detail of the models used in animation, the bust of Victor Hugo (courtesy F. Schmidt, see Plate A.1) obtained from a commercial three-dimensional digitizing system developed by the Laboratoire IMAGE of the Ecole Nationale Supérieure des Télécommunications [34]. In this integrated device, the microprocessor controls a video-laser data acquisition system capable of measuring approximately 200,000 points per minute. The rotation axis (Y axis in Figure 1.1) is usually perpendicular to the beam and along the center line of the object. Figure 1.1 shows the topology of the data obtained from these kinds of devices.

The resulting data are represented as a 2-D array of radii; that is, any non-boundary data point is connected to exactly four neighbouring points, forming a cylindrical contour(Figure 1.1). The device measures the distance from the surface point to the central axis, but before approximating the surface, the radius values r_{ij} are converted to the coordinate triple (x_{ij}, y_{ij}, z_{ij}) in R^3 as follows:

$$\begin{aligned} x_{ij} &= r_{ij} \times cos(j), \\ y_{ij} &= M - i, \\ z_{ij} &= -r_{ij} \times sin(j), \end{aligned} \tag{1.5}$$

where M is the number of rows of points. In the Victor Hugo data set, there are 264 rows of 361 data points (95304 points in all). For notational purposes, the data is indexed by row number $\{i, 0 \leq i \leq 263\}$ and by column number $\{j, 0 \leq j \leq 360\}$.

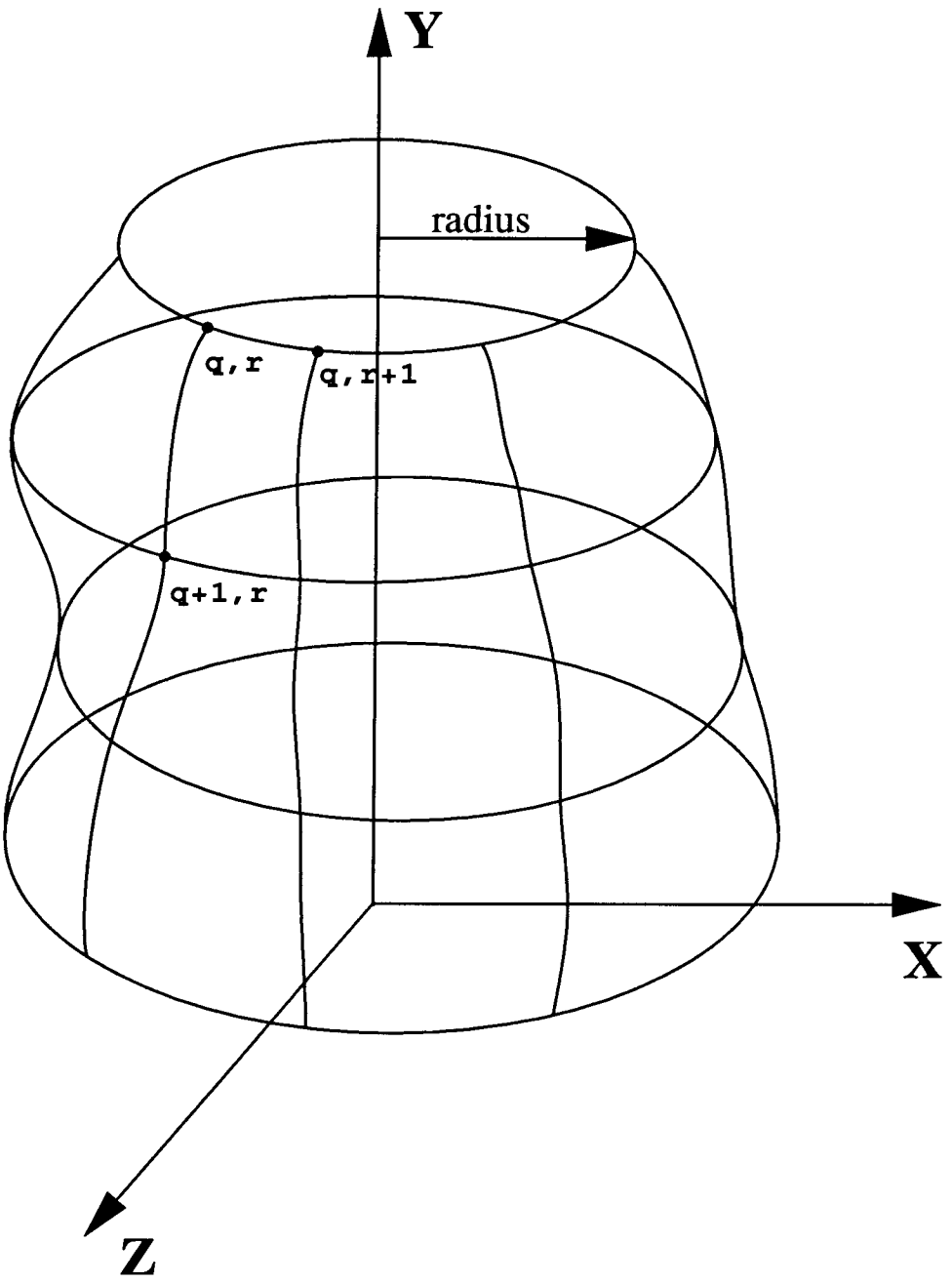


Plate 1.1: Digitized Data Topology.

1.3.2 Data Parameterization

For a cylindrical data set, the natural choice for an approximating surface is also cylindrical. This simplifies the problem of finding the correspondence between the data and the spline surface [1].

For this particular application, the digitized data is used to specify the shape of a portion of an existing spline figure with its own notion of which part of the surface is what (head, foot, etc.) For example, the animated figures created with hierarchical B-splines are typically a single parametric surface encasing the entire body (This is part of the mechanism used to control the deformation of the surface around the articulations.). The particular portion of this surface dedicated to the head is not a cylinder, but a rectangular sub-region of the surface where the edges of the region correspond to the opening at the neck and the interior of the region that encompasses the rest of the head.

In this case, parameterization that maps a data point in R^3 to a point in parametric space (in R^2), is not straightforward. Chapter 3 describes the parameterization method used to approximate cylindrical data.

1.4 Related Work

The literature on surface fitting and approximation is vast. This section will not attempt to survey the field but will briefly examine work on surface approximation for animation. Chapter 2 will also cover specific issues of data parameterization, numerical methods, and parametric surface formulations.

The work of Nahas et al. [30] addresses the same problem of using digitized data to create an animated human figure. A raw digitized mesh of data points is used without further processing as the control vertices of a bicubic B-spline surface. For animation, broad-scale changes to the surface shape are accomplished by embedding the control vertices of the model in a coarser spline mesh, created by selecting a relatively few *characteristic points* from the original B-spline

[1] The operation that identifies which part of the data set matches which part of the approximate surface is termed *parameterization*.

model of the surface. When the characteristic points are moved, they displace the vertices of the model itself. The data set used was sparse and no attempt was made to reduce its size.

The paper by Schmitt et al. [33] presented an adaptive process to fit surface data with a geometrically continuous collection of rectangular bicubic Bézier patches. Bézier patch basis functions do not automatically ensure the continuity between adjacent patches, thus during the approximation process, extra constraints must be imposed to ensure at least geometric continuity (collinear tangents) between patches. The adaptive behaviour is implemented by fitting a portion of the data with a single patch, testing the fit for errors within a given tolerance, and then subdividing the patch if the tolerance is exceeded. Geometric continuity is controlled by using a constrained least squares approximation method where the constraints are imposed by the continuity conditions. A given number of constraints per patch are required to piece the Bézier patches together to form a continuous composite surface. Because of the properties of the basis functions,a Bézier surface will have more control vertices (9 times) than that of the equivalent B-spline surface. (Note, however, that with Bézier patch, it has considerably easier to control the continuity between patches.) Because each Bézier patch is essentially independent, each patch in the surface can be split into two or more subpatches (refinement) independently. The broad class of tensor-product spline surfaces, e.g., B-spline, Beta-splines and their rational counterparts, which provide continuity without the explicit imposition of constraints, must refine an entire row or column of patches to increase the number of patches in the surface.

A hierarchical spline [17] [20] allows local refinement of uniform B-splines or Beta-splines and their rational counterparts (A brief summary of this approach is provided in Section 2.3). In the paper by Forsey and Bartels [19], this property was exploited to provide an adaptive approach to fitting regular (gridded) data in three dimensions, with a hierarchical bicubic B-spline, and generalized to multivariate B-splines in any dimension [18]. This approach utilizes a least squares to fit a coarse spline surface (i.e., on the order of 1-16 patches) of the appropriate topology. Regions that are out of tolerance, i.e., more than a given distance from the data, are refined and the fitting process is repeated for the refined surface. Eventually, regions of

inadequate fit become separated and surrounded by regions within tolerance. These regions are individually accommodated using local refinement and constrained least squares approximation within each separable region. This approach results in a multi-resolution surface formulation in a hierarchical format.

The hierarchical approach provides a further contribution by enabling a certain economy of representation for the final composite surface and produces reasonable results. However, the resulting surface suffers from oscillations when presented with data with high frequency components. The oscillations were reduced considerably through the use of non-uniform hierarchical B-splines and an improved parameterization procedure [35].

The surfaces resulting from the above approach were not appropriate for further manipulation via the hierarchy. Because each level in the hierarchical surface is the result of a separate least squares approximation of the data, corresponding parametric locations in each overlay may have widely varying slopes. This plays havoc with the surface when broad-scale changes in shape are made.

In contrast to the top-down approach in [33] and [20], Lyche and Morken [26] work bottom-up by first approximating the surface with a fine mesh of spline patches and then removing knots in those regions where they will not cause the surface to move out of tolerance. This thesis presents a bottom-up approach to surface approximation using hierarchical splines.

1.5 Overview

Chapter 2 covers the mathematical background for B-splines, refinement and mid-point subdivision, hierarchical B-spline, parameterization and multigrid numerical method. Chapter 3 gives in detail the modified chord-length parameterization method used for the test data, and Chapter 4 discusses the construction of the linear equations used for the approximate and their solution using the full multigrid method. Details of how the full multigrid method is used to construct a hierarchical surface is given in Chapter 5 along with one approach to reducing the storage requirement of the approximating surface. Conclusions and future work are presented

in Chapter 6.

Chapter 2

Mathematical Background

2.1 Uniform Bicubic B-Spline Surfaces

The use of parametric surfaces originated in the field of CAGD. B-spline tensor-product surfaces permit the most supple movements for the purposes of animation [5]. Within this family of surfaces, we are interested in the uniform case when knot spacing is constant in both the U and V parametric directions. A detailed derivation of the uniform B-spline basis functions and efficient algorithms for their evaluation, along with the reasons for their extensive use in CAGD were presented in the paper by Barsky [5]. The properties presented in this thesis, such as local control and control of the degree of continuity at the joints between surface patches, are very desirable for fitting a spline surface to discrete data [3]. In this section, we will briefly review the B-spline surface formulation and present the notation used in subsequent chapters. Detailed explanations can be found in [3] and [14].

A B-spline surface is defined by a set of *control vertices.* Figure 2.1 displays a single patch of order 4 defined by 4×4 control vertices. Notice that the surface does not necessarily interpolate the control vertices.

As shown in Figure 2.2, a bicubic B-spline surface is arranged in a piecewise manner, where each piece is a segment of the surface called a *surface patch.* Each surface patch is defined by a subset of the control vertices. The number of control vertices required to define a surface patch depends upon the order of the underlying basis functions. The entire surface $S(u, v)$ is a

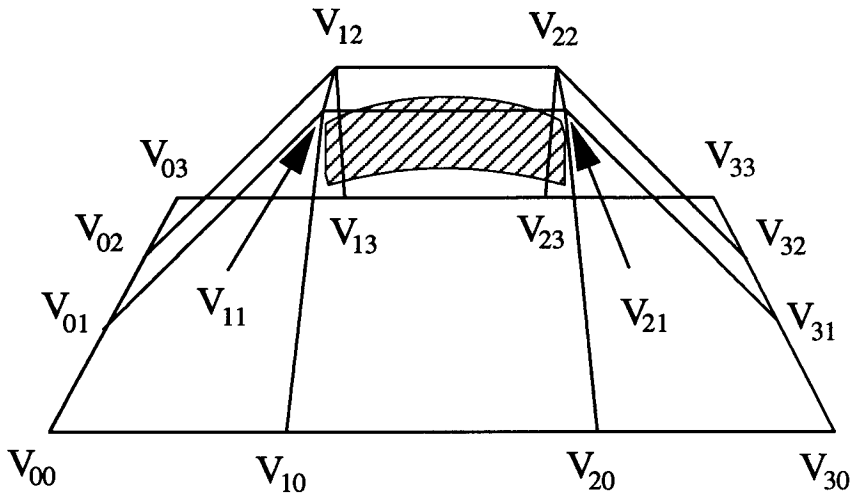


Plate 2.1: A bicubic B-spline surface with a single patch defined by 16 control vertices.

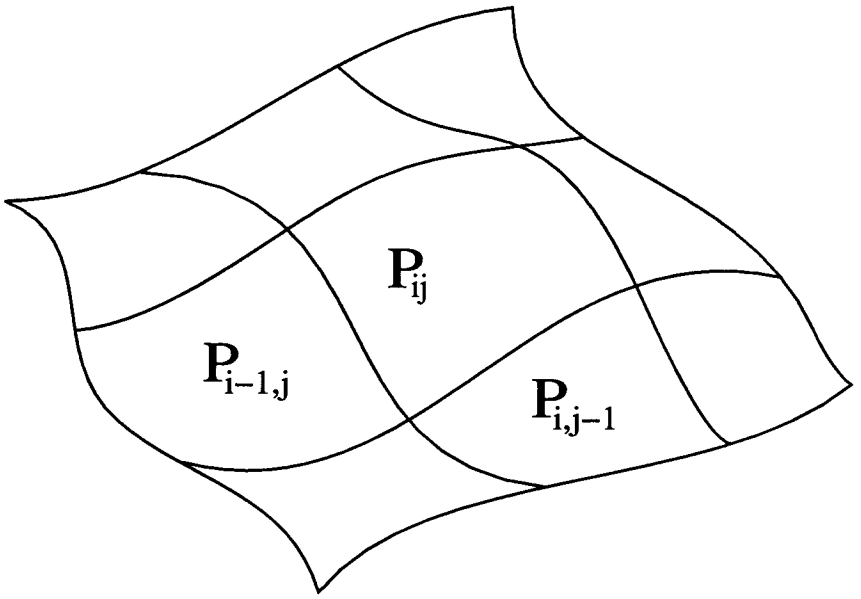


Plate 2.2: A B-spline surface is a mosaic of surface patches.

mosaic of these patches sewn together and has continuity appropriate to the order of the basis functions (i.e., C^{k-1} where k is the degree of the B-spline). Tensor-product surfaces have a rectangular topology and we will label each individual patch as P_{ij}. A bicubic B-spline surface consists of patches that are cubic in both the U and V parametric directions.

The control vertices form a two-dimensional array although each vertex V_{ij} is a vector in three-dimensional space (i.e., $V_{ij} = (x, y, z) \in R^3$).

The mathematical formulation for a patch P_{ij} is defined as a tensor-product in the following way:

$$P_{ij}(u,v) = \sum_{r=-2}^{1} \sum_{s=-2}^{1} V_{i+r,j+s} B_{r,k}(u) B_{s,h}(v) \quad (u,v) \in [(U_{i-1}, V_{j-1}) \times (U_i, V_j)] \tag{2.1}$$

where k and h are the polynomial orders of the basis functions (for cubic B-spline, they are both 4.) and the $\times$ sign means the span in parametric space and will be explained related to Equation 2.3.

The bivariate basis function $B_{i,k}(u_i)B_{j,h}(v_j)$ is the tensor-product of a set of univariate basis functions, where $(U_i, V_j) \in R^2$ is an array of knot positions in parametric space. For uniform spline patches, the knot spacings in both U and V are constant. Each point $P_{ij}(u,v)$ is an average of the 16 control vertices weighted by its underlying basis functions. This is one important property of a B-spline formulation since it means that a control vertex controls the surface shape locally. A single bicubic B-spline surface patch is fully defined by 4 $\times$ 4 control vertices and is unaffected by any other control vertex in the surface. Conversely, a given control vertex exerts influence over only 16 surface patches and has no effect on the remaining patches. B-spline also has other great properties such as variation diminishing, convex hull and etc. [14].

The uniform bicubic B-spline formulation is used throughout this thesis. The basis functions are:

$$[B_{-2}(u)\ \ B_{-1}(u)\ \ B_0(u)\ \ B_1(u)] = \frac{1}{6}[u^3 u^2 u^1 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}. \tag{2.2}$$

A surface $S(u,v)$ is a smooth composition of all patches. It is defined as:

$$S(u,v) = \sum_{i=1}^{m}\sum_{j=1}^{n} B_{i,k}(u_i)B_{j,h}(v_j)V_{i,j} \quad (u,v) \in [(U_0,V_0) \times (U_m,V_n)]. \tag{2.3}$$

Note that the parameter ranges for the U and V directions are the union of each patch's parametric range (Figure 2.3) in Equation 2.1, namely

$$[(U_0,V_0) \times (U_m,V_n)] = [(U_0,V_0) \times (U_1,V_1)] \cup [(U_1,V_1) \times (U_2,V_2)] \cup \ldots [(U_{m-1},V_{n-1}) \times (U_m,V_n)].$$

A one-to-one mapping provides a unique position in parametric space for each surface point $S(u,v)$. However, when we employ only the uniform B-spline formulation, separate parameter ranges for each patch is possible as long as they are consistent. For example, we can have the parameter range $(0,1)$ in both U and V directions for all the patches.

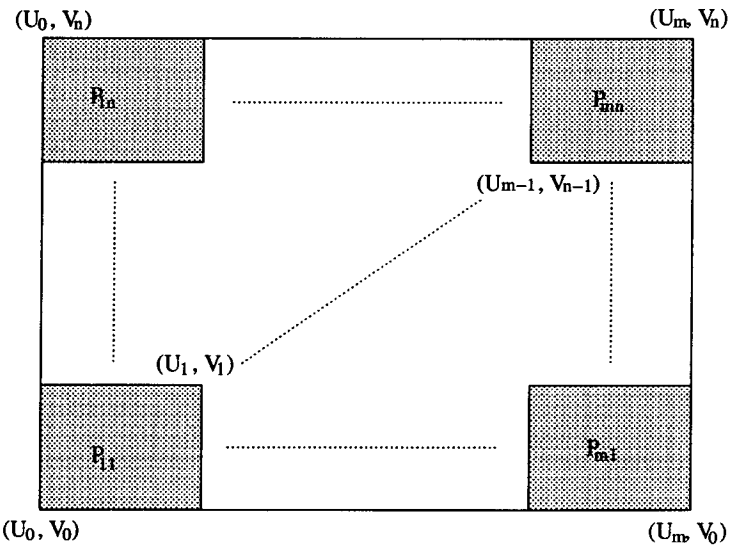


Plate 2.3: The parametric range for a surface is the union of the parametric range for each patch.

2.2 Subdivision Scheme

An important operation that can be performed on spline curve and surfaces is subdivision (also known as refinement) which takes a set of control vertices and basis functions and represents the surface using more control vertices and basis functions. In this section, we will present a brief review of this scheme and discuss a specific but important type of subdivision called midpoint subdivision.

Definition 2.1 *Given a B-spline representation*

$$S(u,v) = \sum_{i=1}^{m}\sum_{j=1}^{n} B_{i,k}(u)B_{j,h}(v)V_{i,j}.$$

The procedure of describing $S(u,v)$ in terms of a larger set of basis functions $N_{i,k}$ and $N_{j,h}$ and a larger net of control points W_{ij} is called **subdivision**, *namely*

$$S(u,v) = \sum_{i=1}^{M}\sum_{j=1}^{N} N_{i,k}(u)N_{j}(v)W_{i,j}.$$

M and N are the number of control vertices in U and V dimension respectively after subdivision.

□

In Figure 2.4, a surface consisting of a single patch is refined into four patches. Note that after subdivision, the patch does not change shape, there are simply more control vertices in the control graph. Subdivision is achieved by inserting new knot positions into the existing knot sequences defining $B_{i,k}(u)$ and $B_{j,h}(v)$. The newly generated basis functions are expressed in terms of *discrete splines* $\alpha_{i,k}(j)$ [9] in the following way:

$$B_{i,k}(u) = \sum_{j=1}^{m+\delta_u} \alpha_{i,k}(j)N_{j,k}(u) \tag{2.4}$$

where δ_u is the number of knots inserted in the U parametric direction. The subdivision in the V direction is similar.

From Equation 2.4, the net of control points W_{ij} can be derived in the same way:

$$W_{i,j} = \sum_{r=1}^{m} \sum_{s=1}^{n} \alpha_{r,k}(i)\alpha_{s,l}(j)V_{r,s}. \tag{2.5}$$

We can express Equation 2.5 in matrix form as:

$$[W] = [\alpha_1][V][\alpha_2]. \tag{2.6}$$

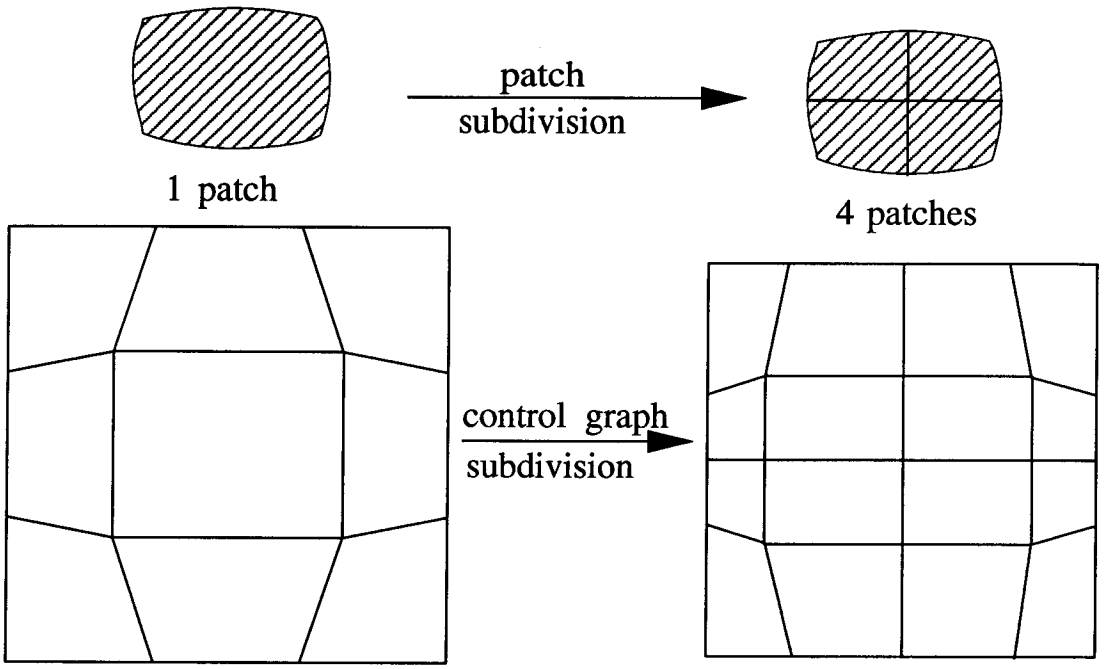


Plate 2.4: Subdivision splits a single patch into four patches.

As stated before, the most straightforward subdivision splits each parametric range in U and V at its midpoint. This converts each patch determined by the control vertices $[V]$ into four patches (Figure 2.4). In parametric space, the region for the original patch is split into four regions along the midlines as shown in Figure 2.5. For uniform cubic B-splines:

$$\begin{aligned} [\alpha_1] &= \begin{cases} [M_1] & \text{for regions I and III} \\ [M_2] & \text{for regions II and IV,} \end{cases} \\ [\alpha_2] &= \begin{cases} [M_1] & \text{for regions I and II} \\ [M_2] & \text{for regions III and IV,} \end{cases} \end{aligned} \tag{2.7}$$

where

$$[M_1] = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \end{pmatrix} \quad [M_2] = \begin{pmatrix} \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}. \tag{2.8}$$

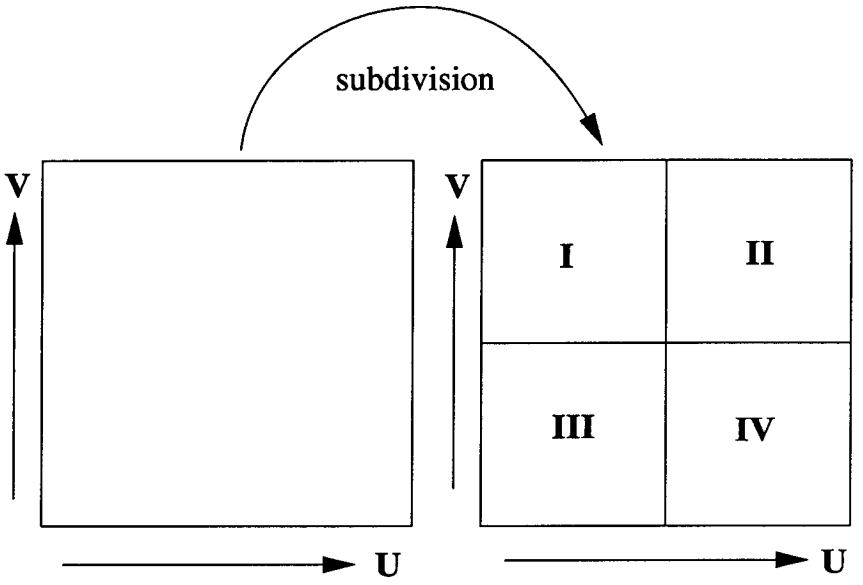


Plate 2.5: Subdivision splits the parametric space at the midpoint.

2.3 Hierarchical B-splines

2.3.1 Practical Consideration of Tensor-Product Surfaces

Subdivision for tensor-product surfaces is non-local because the nature of knot insertion is non-local. For example, if patch P_{ij} in Figure 2.6a is the place where we want to refine, this can only be done by inserting two knots. One sequence of knot positions is along the U parametric direction $\{U_{ik} | 1 \leq k \leq n\}$ and the other sequence of knot positions $\{V_{kj} | 1 \leq k \leq m\}$ is along the V direction. However, because of the way the basis functions are defined and evaluated, subdivision does not simply split patch P_{ij} into four patches. Each patch in row i and column

j are also split resulting in many more patches and control vertices than desired or necessary [1]. In Figure 2.6b, we can see more surface patches are generated than we expected (9 patches versus 3 patches). When modelling large complex surfaces, this property could add hundreds of unnecessary patches.

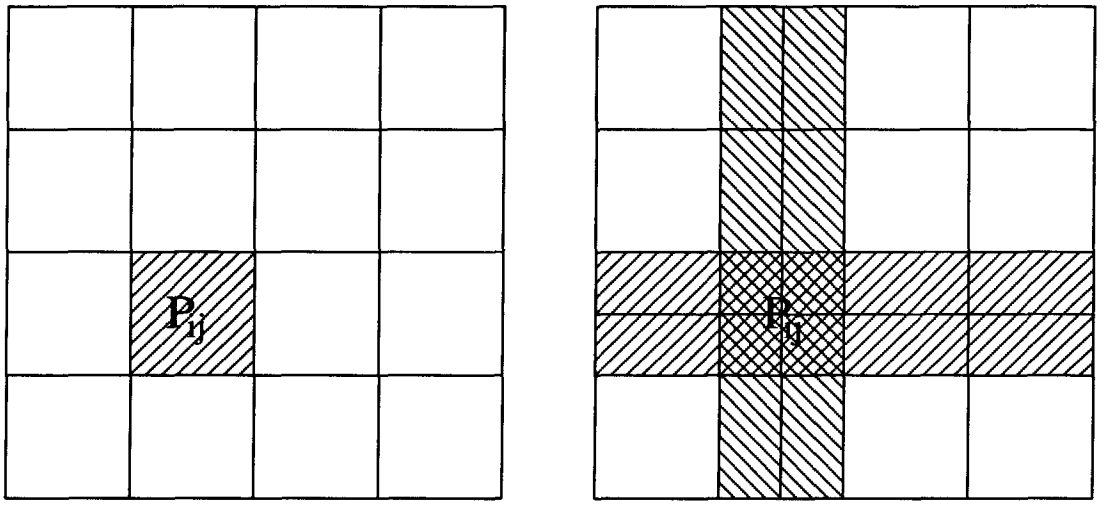


Plate 2.6: Subdivision through non-local knot insertion.

2.3.2 Localized Subdivision–Hierarchical B-splines

The hierarchical B-spline [17] is designed to localize the effect of subdivision through the use of *overlay surfaces*. Overlay surfaces are hierarchically controlled subdivisions. It is important to keep in mind that the W surface (Equation 2.5) is an exact re-representation of its parent V surface. If one of the control vertices of W is moved, then the W surface departs from its V parent, but only in the area influenced by the displaced control vertex. Outside of that region, the two representations are identical. Because of this correspondence, we could just use the V definition as our basic description of the surface, and save only those control vertices of W which

[1] For a bi-cubic B-spline one-patch surface, to split the patch into four, three new knots have to be inserted.

define the position of the displaced surface. The child surface defined by only those retained control vertices of W is called an *overlay*. This portion is regarded as a separate surface to be manipulated. To maintain the continuity between the overlay surface and its parent surface (the V representation), more than one control vertex of W must be kept and these are only those control vertices that do not disturb the continuity of the surface.

When editing takes place at one level of the surface, any overlays resting within the edited area are expected to remain embedded in that area. They will follow editing changes only if they can be dynamically tied to that area. This means that their control vertices must move in accordance with the movement of the section of the surface being edited.

Hierarchical surface representation meets this requirement by introducing the *reference* and *offset* concepts. At level l, each control vertex V_{ij} of any part of the surface is of the form of:

$$V_{ij}^l = R_{ij}^l + O_{ij}^l \tag{2.9}$$

where R_{ij}^l is the reference control vertex and O_{ij}^l is the offset. Usually, R_{ij}^l is derived from the parent surface definition through subdivision, and O_{ij}^l is the change to the original W_{ij}^l's vectors resulting from modification of the overlay surface. Thus the surface representation in Equation 2.3 at level l can be rewritten as:

$$\begin{aligned} S^l(u,v) &= \textstyle\sum_{i=1}^{m_l}\sum_{j=1}^{n_l} B_i^l(u_i)B_j^l(v_j)V_{i,j}^l \\ &= \textstyle\sum_{i=1}^{m_l}\sum_{j=1}^{n_l} B_i^l(u_i)B_j^l(v_j)(R_{i,j}^l + O_{i,j}^l) \\ &= \textstyle\sum_{i=1}^{m_l}\sum_{j=1}^{n_l} B_i^l(u_i)B_j^l(v_j)R_{i,j}^l + \sum_{i=1}^{m_l}\sum_{j=1}^{n_l} B_i^l(u_i)B_j^l(v_j)O_{i,j}^l. \end{aligned} \tag{2.10}$$

Here the superscript l is used to indicate the parameter associated with a certain hierarchy level l. Now we can see that each overlay surface is composed of two components. One is the reference surface:

$$S_R^l(u,v) = \sum_{i=1}^{m_l}\sum_{j=1}^{n_l} B_i^l(u_i)B_j^l(v_j)R_{i,j}^l \tag{2.11}$$

and the other is the offset surface:

$$S_O^l(u,v) = \sum_{i=1}^{m_l}\sum_{j=1}^{n_l} B_i^l(u_i)B_j^l(v_j)O_{i,j}^l \tag{2.12}$$

Editing changes to the V-defined surface will automatically cause revisions in the W-defined surface by directly changing the R's, and through them revisions are made dynamically to the W's. Editing changes to the W-defined surface are recorded as changes to the O's. Thus any changes in coarser levels will change all of the finer level R's and result in a global change in surface shape. Changes to the offsets at the current level affect the current level of refinement and all of the finer levels.

Since the reference information R_{ij}^l at level l is always directly derivable from its parent surface, the crucial information needed to store the upper levels are the non-zero offsets. Repeated substitution of Equation 2.11 into Equation 2.10 results in the recursive formulation of Equation 2.3 with L being the finest level:

$$\begin{array}{rl} S(u,v) & = \sum_{i=1}^{m_0} \sum_{j=1}^{n_0} B_i^0(u_i) B_j^0(v_j) V_{i,j}^0 \\ & + \sum_{i=1}^{m_1} \sum_{j=1}^{n_1} B_i^1(u_i) B_j^1(v_j) O_{i,j}^1 \\ + \ldots & + \sum_{i=1}^{m_L} \sum_{j=1}^{n_L} B_i^L(u_i) B_j^L(v_j) O_{i,j}^L. \end{array} \tag{2.13}$$

Thus, a full representation of a hierarchical surface requires only the control nodes of the root level, the knot spacing at each level and the non-zero vector offset values O_{ij}^l within each level. This offers a considerable amount of storage saving. This is especially suitable for editing a complex shape because the storage for the surface is proportional to the number of edited points.

2.4 Parameterization

Definition 2.2 *Given a data set* $\{D_{ij} = (x_{ij}, y_{ij}, z_{ij}) | D_{ij} \in R^3, (1,1) \leq (i,j) \leq (M,N)\}$ *and a parametric position set* $\{U_{ij} = (u_{ij}, v_{ij}) | U_{ij} \in [(U_{00}, V_{00}) \times (U_{mn}, V_{mn})] \subset R^2\}$. *Find the parametric surface* $S(u,v) \in R^3$ *such that*

$$S(u_{ij}, v_{ij}) = D_{ij} \tag{2.14}$$

where

$$S(u_{ij}, v_{ij}) = \sum_{r=0}^{m} \sum_{s=0}^{n} B_i(u_{ij}) B_j(v_{ij}) V_{rs}.$$

If Equation 2.14 contains all of the $M \times N$ data points, surface S is an interpolation of the data D.

□

In order to generate the equations for the surface approximation problem, each data point D_{ij} must be associated with a corresponding parametric domain point (u_{ij}, v_{ij}). Defining this association for a particular data set and a particular surface is referred to as parameterization. If the association is already known, this is referred to as a *non-parametric problem.* It is called a *parametric problem* if we are attempting to approximate data from a physical object where domain information is unknown and therefore must be estimated.

2.4.1 Parameterization Schemes

Parameterization is a difficult problem, but a number of different algorithms exist which have been proven effective for univariate and bivariate problems [16] [32]. Different parameterizations of the same data set will result in very different surfaces in terms of their shape and continuity.

Since the parameterization scheme used in the U and V direction is independent, let us discuss the different parameterization schemes in the context of curve fitting using a single parameter. Given a set of data points $D_i = (x_i, y_i)$ and a knot sequence $\{t_0, t_1, \ldots, t_n\}$ with increasing order, we fit the data to a curve $C(t) = (x(t), y(t))$. Without loss of generality, we can assume $t_0 = 0$, and $h_i = t_{i+1} - t_i$ where h_i forms the knot spacing vector $H = (h_0, h_1 \ldots, h_{n-1})$.

Uniform Parameterization

The easiest way to determine h_i is to simply set all h_i's to be the same constant value, namely $h_i = Const$, for $i = 0, 1, \ldots, n-1$. This is called *uniform* or *equidistant* parameterization and was proposed by de Boor [12] and Ahlberg [1]. It is sometimes used because the computations are simple since time is not needed to compute each h_i. However, the amount of computational time saved is minor, and curves with loops and cusps can occur even though the defining equations would appear to make that impossible. Examples are cited by de Boor [12] and

Epstein [13]. This method is too simplistic to cope with most practical situations, with one exception given by Foley [15], where the uniform parameterization is superior to chord-length parameterization. The overall poor performance of the uniform parameterization is attributed to the fact that it "ignores" the geometry of the data points. If we interpret the parameter t of the curve as a time variable, as time passes from time t_0 to time t_n, the point $C(t)$ traces out the curve from point $C(t_0)$ to $C(t_n)$. With uniform parameterization, point $C(t)$ spends the same amount of time between any two adjacent data points, irrespective of their relative distances. A good analogy is that of a car driving along the interpolating curve. We have to spend the same amount of time between any two data points. If the distance between two points is large, we must move with a high velocity. If the next two data points are close to each other, we will overshoot since we can not abruptly change our velocity and acceleration (i.e., the first derivative and second derivative respectively in the physical counterparts) [16].

Chord-length Parameterization

Given the physical analogy above, it is perhaps more reasonable to adjust the velocity according to the geometric distribution of the data points. One way of achieving this is to have the knot spacing proportional to the distances between data points as calculated in Euclidean space. The equation is as follows:

$$h_i = \| D_{i+1} - D_i \| \tag{2.15}$$

A parameterization of the form Equation 2.15 is called a *chord length parameterization.* This is a commonly used parameterization scheme and usually produces better results than uniform knot spacing does, although not in all cases. Foley [15] cites an example where chord length parameterization does not give the shape users have in mind and wild oscillations occur because the data is poorly scaled or the direction of data changes abruptly.

Epstein [13] proves that the parameterization based on a vector-norm distance such as chord length will not produce curves with cusps at the data points, thus giving some theoretical advantage to the chord length parameterization over the uniform one.

Centripetal Parameterization

Another similar parameterization based on the distance norm was developed by Lee [24] and is called the *centripetal model.* This parameterization is based on the same paradigm as chord-length parameterization. If we set

$$h_i = \sqrt{\| D_{i+1} - D_i \|}, \tag{2.16}$$

the resulting motion of a point on the curve will minimize the centripetal force acting on it. So for trajectory based designs where we care about machine wear, this is the right candidate.

Affine Invariant Parameterization

The uniform, chord length and centripetal methods are each invariant under rotations, translations and equal scaling in the x and y coordinates (consider data fitting in 2-D). However, the chord length and centripetal methods are not scale invariant. This means that when the x and y coordinates of the data are scaled differently, the resulting fit is not a simple scaled version of the original fit. The geometric properties of scale, rotation and translation invariance are important in character font applications [31]. Affine invariant chord length and angle parameterizations are designed to meet these kinds of practical use [31].

2.4.2 Summary

Parameterization is a difficult problem that is very application dependent. Usually the input data points decide which parameterization method to choose and there is no general method which is suitable for most practical case. (e.g., the case when scale, data point spacing, and angles between adjacent data points all vary largely.) Norm based parameterizations such as chord length and centripetal parameterization have certain important geometric properties which make them superior to the uniform case. Chord length and centripetal parameterization are satisfactory for most geometric applications. The problem with these two schemes is that they are not scale invariant, that is, if the data is scaled differently in the x and y directions, the interpolation result will be different.

Affine invariant parameterization is a useful property. Uniform parameterization has this property but it does not produce good results when the data point spacings are large and variable. Affine invariant chord length and angle parameterizations generally provide pleasing fitting results. However, both will fail if the data points fall in the one dimension-less hyperplane [35].

2.5 Multigrid Numerical Method

2.5.1 Iterative Methods on Linear Algebraic Equations

The surface construction problem discussed here involves a large system of linear equations. Let the matrix form of our system of equations be:

$$Av = d \tag{2.17}$$

where A represents the coefficient matrix, the vector v contains the unknowns and the vector array d contains the known data point triples.

The existing methods can be classified into two types, direct methods and iterative methods. Direct methods, such as Gaussian elimination, provide unique and exact solutions, however they are extremely inefficient with computational costs of $O(N^3)$. When the matrix A is sparse, these elimination methods fill up those entries which are originally zero and a lot of storage space is wasted.

For many practical applications, iterative methods are used to provide faster solutions. The speed of the algorithm depends on the rate of convergence, and since each iteration costs more or less the same amount of computational time, a rapid convergence rate is crucial. Given an initial guess of u^0, at the r^{th} iteration the iterative procedure calculates the new approximation using:

$$u^{r+1} = Cu^r + c \tag{2.18}$$

where C is the iteration matrix and c is a constant vector.

Many iterative methods have to been developed in the past for numerically solving algebraic equations. Among these, Gauss-Seidel and Jacobi relaxation methods are the two types

commonly used. The Jacobi method is very similar to the Gauss-Seidel method. The only difference is that the Jacobi method keeps using the solution from iteration r for all variables while computing new values for iteration $r+1$. Gauss-Seidel makes use of a new value immediately after it is calculated.

Here, we give a brief overview of the formulation and function of the Gauss-Seidel relaxation scheme which is employed throughout this thesis. We denote the matrices representing the diagonal, lower-triangular and upper-triangular parts of A by D, L and U respectively, with L and U having zero elements on the diagonal, thus $A = L + D + U$. For the case of Gauss-Seidel relaxation, where each entry x_i^r in u^r is replaced immediately when x_i^{r+1} is available, the iteration equation is:

$$(D + L)u^{r+1} = d - Uu^r. \tag{2.19}$$

The corresponding matrix C and the vector c can be computed as:

$$C = -(D + L)^{-1}U, \quad c = (D + L)^{-1}b \tag{2.20}$$

where v is the exact solution and u is the approximate solution to equation 2.17. The error vector $e = v - u$, satisfies the residual function:

$$Ae = r \tag{2.21}$$

where $r = d - Au$ and is called the *residual.*

The residual vector contains error components of different frequencies. An efficient way to damp the rapid fluctuations in the residuals is by Gauss-Seidel relaxation scheme. During the first few iterations, Gauss-Seidel has very fast convergence, with residuals rapidly decreasing from one iteration to the next. But once the high frequency components in the error are smoothed out, the convergence rate levels off. This inherent computational sluggishness can be studied from a spatial frequency perspective [37]. A local Fourier analysis [7] shows that the convergence is fast as long as the residuals have high frequencies compared to the scale of the grid spacing. As soon as the residuals are smoothed out, convergence slows down. The overall convergence is asymptotical, and thus slow overall if high accuracy in the solution is required.

One solution to this is through the use of multigrid methods.

2.5.2 Multigrid Philosophy

One way to improve a relaxation scheme, at least in its early stages, is to use a good initial guess. A technique for obtaining an improved initial guess is to perform some preliminary iterations of relaxation on a coarse grid and then use interpolation to get the resulting approximation as an initial guess on a finer grid. A coarse grid is chosen because relaxation on a coarse grid is less expensive than on a fine grid since there are fewer unknowns to be updated. Another reason a coarse grid relaxation is faster is that there is a faster convergence rate on a coarse grid. If we examine the smooth components of the residual on a coarse grid, they appear less smooth than on a grid that is twice as fine as shown in Figure 2.7. This suggests that when relaxation stalls, signalling the predominance of smooth error modes, it is advisable to move to a coarser grid, on which those smooth error modes appear more oscillatory [8]. The relaxation will then be more effective. This is the basic idea of multigrid method.

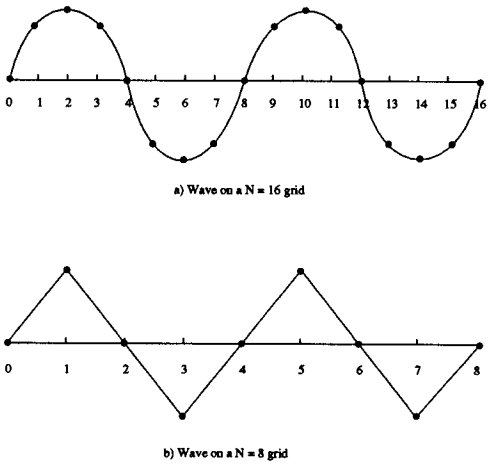


Plate 2.7: A wave on $\Omega^h(N = 16)$ is projected onto $\Omega^{2h}(N = 8)$. On the coarse grid, the wave seems more oscillatory than on the fine grid.

2.5.3 Basic Strategies

Suppose one has a set of grids $\Omega^{h_0}, \Omega^{h_1}, \ldots, \Omega^{h_n}$ over the same domain Ω (h_i is the grid spacing at the ith grid). The grid spacing ratio is defined as $h_{k+1} : h_k$, which is usually chosen as $2 : 1$. The reason for choosing this value as the grid spacing ratio will be explained later. There are two strategies which can be used:

- Nested Iteration.

  This strategy incorporates the idea of using coarse grids to obtain a better initial guess for fine grids.

  1. Relax $Av = d$ on level 0 and interpolate to level 1 providing initial guess.
  2. Relax $Av = d$ on level 1 and interpolate to level 2 providing initial guess.

  $\vdots$

  n. Relax $Av = d$ on level n-1 and interpolate to level n providing initial guess.

  n+1. Relax $Av = d$ on level n to obtain the final solution.

- Coarse Grid Correction.

  This strategy incorporates the idea of using the residual equation to relax on the error.

  1. Relax $Av = d$ on Ω^h to obtain the approximation u^h.
  2. Compute the residual $r = d - Au^h$.
  3. Relax residual equation $Ae = r$ on Ω^{2h} to obtain the approximation to the error e^{2h}.
  4. Correct the approximation obtained on $\Omega^{2h} : u^h \leftarrow u^h + Interpolate(e^{2h})$.

These two strategies will be adopted in our multigrid method discussed below. They cooperate with each other and demonstrate great efficiency. First, we will give the various methods used to compose multigrid method as a whole, which will be called *operators*.

2.5.4 Multigrid Operators and Parameters

Nested iteration requires that the original problem be re-defined on the coarser grid [8]. One problem is that there might still be smooth components in the error when we reach the finest grid due to the process of transferring from a coarser grid to a finer grid. As we will see below, the scheme designed to get rid of this defect is another pathway to multigrid method.

There is a rationale for using the coarse grid correction. However, it leaves a lot of questions unanswered. The relaxation method (Gauss-Seidel) used in the coarse grid correction scheme needs to be determined. Different relaxation methods might greatly affect the algorithm's performance. We have to define the *restriction* operator used to transfer the residuals from Ω^h to Ω^{2h} and the *prolongation* operator which is used to transfer the error estimate back to Ω^h from Ω^{2h}. The restriction operator from a fine grid to a coarse grid is denoted as I_h^{2h} and the prolongation operator from a coarse grid to a fine grid I_{2h}^h. In the coarse grid correction scheme, the number of pre-relaxations, before restricting the residuals onto the next coarser grid, is ν_1, and the number of post-relaxations after prolongating the error solution onto a finer grid is ν_2. Also, the grid spacing ratio ρ should be chosen according to provide multigrid method optimal performance.

In multigrid method, the intralevel processes are usually basic relaxation methods (such as Gauss-Seidel relaxation), the interlevel prolongation processes involve local Lagrange interpolations, and the restriction processes involve local averaging operations. The exact form of these operations are problem-dependent. Fortunately, there are well established standards that have been obtained from numerical experiment. For general analysis of the theory, there are detailed references by Hackbusch [21] and Brandt [7]. Further discussion concerning the choices for the concrete problem will be presented in later chapters.

2.5.5 Multigrid V-cycle

Using the intergrid transfer operators and necessary parameters, the two-grid coarse grid correction scheme can be refined into a multigrid method. The idea of residual correction on a coarse grid is applied recursively and the correction process is repeated on successively coarser

grids until a direct solution of the residual equation at the coarsest level is inexpensive. The coarse grid correction scheme has been incorporated into the multigrid V-cycle to provide fast numerical performance.

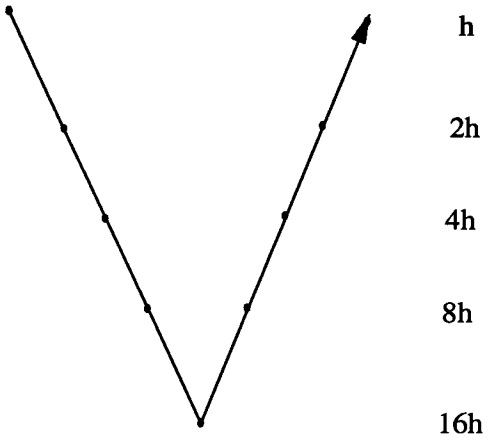


Plate 2.8: The schedule for grid visiting when the multigrid V-cycle is executed. Each level is identified with its spacing. Multigrid V-cycle starts from the finest level and proceeds to the coarsest level, then returns to the finest level.

Figure 2.8 shows the schedule for the visiting order of the grids. Because of the pattern of this diagram, this algorithm is called the *V-cycle*. The V-cycle scheme has a compact recursive definition which is given by:

Algorithm 2.1 *Multigrid V-Cycle Operator MGV*

$$\mathbf{u}^h \leftarrow MGV^h(\mathbf{u}^h, \mathbf{d}^h)$$

1. *Relax* ν_1 *times on* $A^h v^h = d^h$ *with a given initial guess* u^h.

2. *IF* Ω^h = *coarsest grid, THEN go to step 4*
   ELSE $\mathbf{d}^{2h} \leftarrow I_h^{2h}(\mathbf{d}^h - \mathbf{A}^h\mathbf{u}^h)$
   $\mathbf{u}^{2h} \leftarrow 0$
   $\mathbf{u}^{2h} \leftarrow MGV^{2h}(\mathbf{u}^{2h}, \mathbf{d}^{2h})$.

3. *Correct* $\mathbf{u}^h \leftarrow \mathbf{u}^h + I_{2h}^h \mathbf{u}^{2h}$.

4. *REPEAT* $\mathbf{u}^h \leftarrow MR^h(\mathbf{u}^h, \mathbf{d}^h)$ ν_0 *times.*

5. *END.*

The MR operator is a exact solution operator which relaxes the equations ν_0 times and returns the exact solution at the coarsest level.

The V-cycle is just one of a family of multigrid cycling schemes. The entire family is called the *ν-cycle* method and is defined recursively by the following algorithm:

Algorithm 2.2 *Multigrid ν-Cycle*

$$\mathbf{u}^h \leftarrow MG\nu^h(\mathbf{u}^h, \mathbf{d}^h)$$

1. *Relax* ν_1 *times on* $A^h v^h = d^h$ *with a given initial guess* u^h.

2. *IF* Ω^h *= coarsest grid, THEN go to step 4*
   ELSE $\mathbf{d}^{2h} \leftarrow I_h^{2h}(\mathbf{d}^h - \mathbf{A}^h \mathbf{u}^h)$
   $\mathbf{u}^{2h} \leftarrow 0$
   REPEAT $\mathbf{u}^{2h} \leftarrow MG\nu^{2h}(\mathbf{u}^{2h}, \mathbf{d}^{2h})$ ν *times.*

3. *Correct* $\mathbf{u}^h \leftarrow \mathbf{u}^h + I_{2h}^h \mathbf{u}^{2h}$.

4. *REPEAT* $\mathbf{u}^h \leftarrow MR^h(\mathbf{u}^h, \mathbf{d}^h)$ ν_0 *times.*

5. *END.*

In practice, only $\nu = 1$ (which gives a V-cycle) and $\nu = 2$ are used. Figure 2.9 shows the schedule of grid visiting for $\nu = 2$, which is called the *W-cycle scheme.*

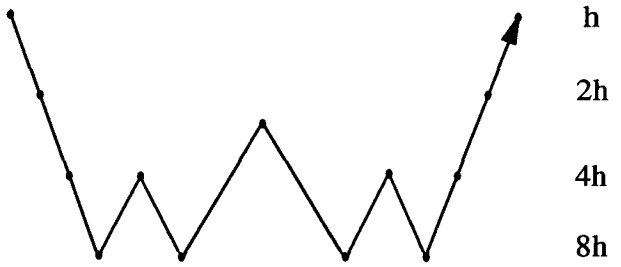


Plate 2.9: The schedule for grid visiting of W-cycle scheme.

2.5.6 Full Multigrid V-cycle

In Section 2.5.3, we proposed two distinct, basic multigrid strategies; nested iteration and coarse grid correction. In the multigrid V-cycle, the coarse grid correction scheme was employed to smooth out the low frequency error components. Nested iteration involves solving a small scale problem on the coarser grid and transferring the solution back to the finer grid to provide the initial approximation. Clearly, another recursive path leads to the coarsest grid.

The algorithm that joins nested iteration with V-cycle is called the *full multigrid V-cycle.* The visiting schedule is shown in Figure 2.10.

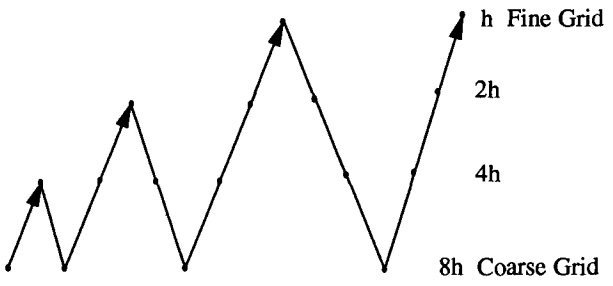


Plate 2.10: Full multigrid grid visiting schedule.

Here we have the compact algorithm:

Algorithm 2.3 *Full Multigrid V-Cycle Operator FMGV*

$$\mathbf{v}^h \leftarrow FMGV^h(\mathbf{u}^h, \mathbf{d}^h)$$

1. *IF* Ω^h = *coarsest grid, THEN go to step 3.*
   ELSE $\mathbf{d}^{2h} \leftarrow I_h^{2h}(\mathbf{d}^h - \mathbf{A}^h\mathbf{u}^h)$
   $\mathbf{u}^{2h} \leftarrow 0$
   $\mathbf{u}^{2h} \leftarrow FMGV^{2h}(\mathbf{u}^{2h}, \mathbf{d}^{2h})$.
2. *Correct* $\mathbf{u}^h \leftarrow \mathbf{u}^h + I_{2h}^h\mathbf{u}^{2h}$.
3. *REPEAT* $\mathbf{u}^h \leftarrow MR^h(\mathbf{u}^h, \mathbf{d}^h)$ ν_0 *times.*
4. *END.*

FMG method start from the coarsest grid and proceed to the finest grid. Each V-cycle inside a full multigrid iteration is preceded by a smaller V-cycle designed to provide the best possible initial guess. As shown in Section 2.5.7, the extra work done in these preliminary V-cycles is not only inexpensive, but it generally pays for itself by providing better initial guess for the finest grid.

Full multigrid method is the complete knot into which the many threads of numerical methods are tied. It is a remarkable synthesis of ideas and techniques which individually have been well known and used for a long time. Taken alone, many of these methods have serious defects (Gauss-Seidel relaxation, coarser grid correction, nested iteration, and etc.). Full multigrid is a technique for integrating them so that they can work together in a way to remove the limitations. The result is a simple but powerful algorithm.

2.5.7 Performance Evaluation

Multigrid is a very efficient method for solving a system of linear equations. Both the storage and computational costs are of optimal order.

Storage Cost

Consider a two-dimensional grid with $M \times N$ (which we will represent by MN for later reference) grid points at the finest level. At each level, two arrays must be stored, the right hand side of Equation 2.17 and the unknowns from the left hand side. So for the storage requirements, the finest grid Ω^h needs $2MN$ units, the next coarser grid Ω^{2h} needs $\frac{1}{4}$ as many, etc. If we accumulate all of the storage units at each level, we have the upper bound on storage which is:

$$S = 2MN(1 + 2^{-2} + 2^{-4} + \cdots + 2^{-2n}) < \frac{2MN}{1 - 2^{-2}} = \frac{8}{3}MN. \tag{2.22}$$

Since we only store the components necessary for carrying out the relaxation, namely the input and output of the equations, and have no intermediate storage, the storage cost is optimal.

Computational Cost

Now let us consider the computational cost. It is convenient to measure these costs in terms of *work units* (WU). A work unit is the cost of performing one relaxation sweep on the finest grid. Equivalently, it is the cost of expressing the fine grid operator. It is customary to neglect the cost of intergrid transfer operations which could only amount to 15-20 percent of the cost of the entire cycle.

First consider a multigrid V-cycle with one pre-relaxation and one post-relaxation, namely $\nu_1 = \nu_2 = 1$. Since the coarsest level has a constant number of relaxations and is only visited once, we can simplify by assigning $\nu_0 = 2$. Adding these costs and again using the geometric series for an upper bound gives:

$$T_{MGV} = 2(1 + 2^{-2} + 2^{-4} + \cdots + 2^{-2n}) < \frac{2}{1 - 2^{-2}}WU = \frac{8}{3}WU. \tag{2.23}$$

For a full multigrid V-cycle, we just sum up the costs for each single V cycle. Starting from the finest grid, the computational cost is bounded by $\frac{8}{3}WU$ as just calculated above. The next coarser grid costs $\frac{1}{4}$ as much as the finer grid. Then the computational cost for full multigrid V-cycle is:

$$T_{FMGV} = (\frac{2}{1 - 2^{-2}})(1 + 2^{-2} + 2^{-4} + \cdots + 2^{-2n}) < \frac{2}{(1 - 2^{-2})^2}WU = \frac{32}{9}WU. \tag{2.24}$$

Since each sweep at the finest level is $O(MN)$, the total computational cost of full multigrid V-cycle is also $O(MN)$ for a single cycle, which is of optimal order.

Convergence Rate

In order to determine the final cost for solving a system of equations, we have to know the convergence rate. The total cost is the cost for each cycle multiplied by the number of cycles required to achieve a particular accuracy. The number of cycles depends totally on the convergence rate for a fixed error tolerance.

Convergence rate analysis is a complex area [21] [7]. Local mode analysis [21] provides a non-rigorous way to predict the convergence rate which conforms well to experimental results. A key observation is that when $\nu_1 = \nu_2 = 1$, there is an error reduction rate of $\frac{1}{4}$ for each sweep. The relationship between ν_1, ν_2 and the error deduction rate r can be expressed as:

$$r = \frac{1}{(\nu_1 + \nu_2)^2}. \tag{2.25}$$

Due to the preliminary cycling through coarser grids, only $O(1)$ V-cycles are needed by the time the algorithm reaches the finest grid. Therefore, the total computational cost of FMG method is still $O(MN)$ [8].

Chapter 3

Parametric Space Mapping

3.1 Parameterization

Tensor-product B-splines (the surface formulation used in this thesis) are rectangular. Figure 3.1 shows such as surface with the boundary edges marked North, South, East and West. A cylindrical mapping on the data of Figure 3.1 is shown in Figure 3.2a. The East and West edges are mapped to the first column of data and the N and S edges are mapped to the last row of data.

As stated previously, we wish to approximate a cylindrically digitized object with a B-spline surface that the boundary of the surface corresponds to the neck of the figure (i.e., the last row of data). This goal can not be met by mapping a cylindrical surface onto cylindrical data as shown in Figure 3.2a, the mapping from the surface shown in Figure 3.2b to the head will meet this requirement.

After the initial mapping, we also wish to more closely approximate those regions which are articulated, such as the mouth and the eyes. These areas contain the most interesting features and are important for modelling and editing. Finally, we would like the parameterization algorithm to be as fast as possible.

This goal is achieved in a three step process. First the data is mapped into an intermediate uniform parametric space ST. Then the mapping function $M(u_i, v_j) = (s_i, t_j)$ is defined to map each parametric pair (u_i, v_j) in UV space into ST space. This mapping function is defined

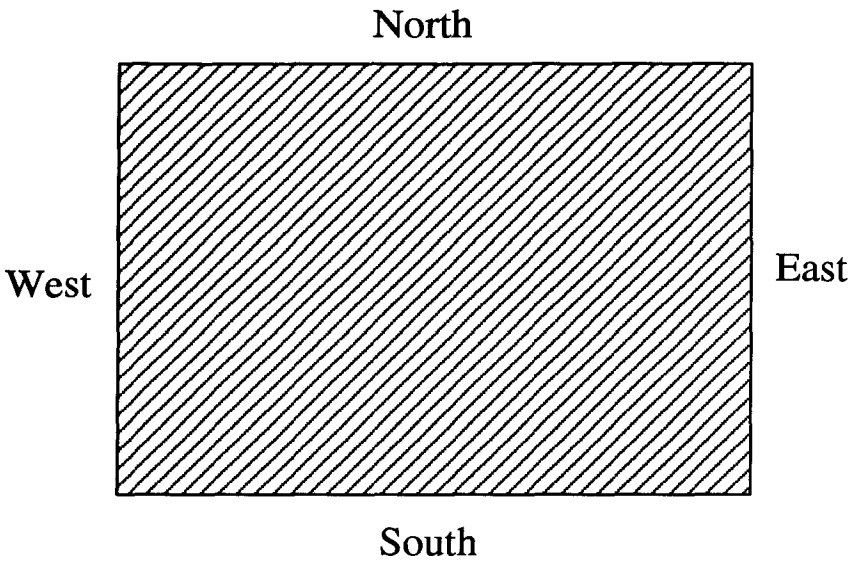


Plate 3.1: Labels for the tensor-product B-spline surface.

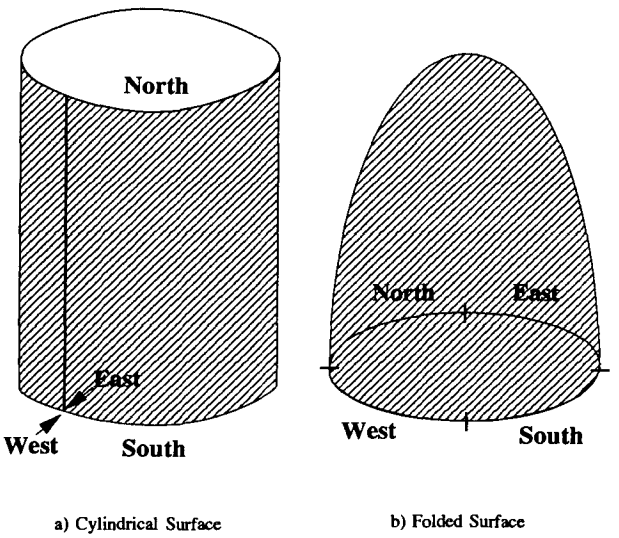


Plate 3.2: Two approaches to approximating the head data: a) wrap a cylindrical surface around the head, b) fold the four edges of a rectangular surface to cover up the head with the opening at the neck.

through Bernstein polynomials having a given knot spacing. Initially, a single patch is created, then the patch is refined and control vertices are displaced according to the chord-length measure. At last, the UV space is uniformly sampled and mapped back to data space through function $M(u_i, v_j)$ in order to form a set of gridded data points D_{ij} in Equation reffittingdef.

3.2 Initial Parameterization

Imagine that we put a square rubber sheet over Victor Hugo's bust and let the edge of the sheet match the opening at the neck. The first row of the data will be mapped to the center of the sheet with the rest of the rows spreading out to the edge (Figure 3.3).

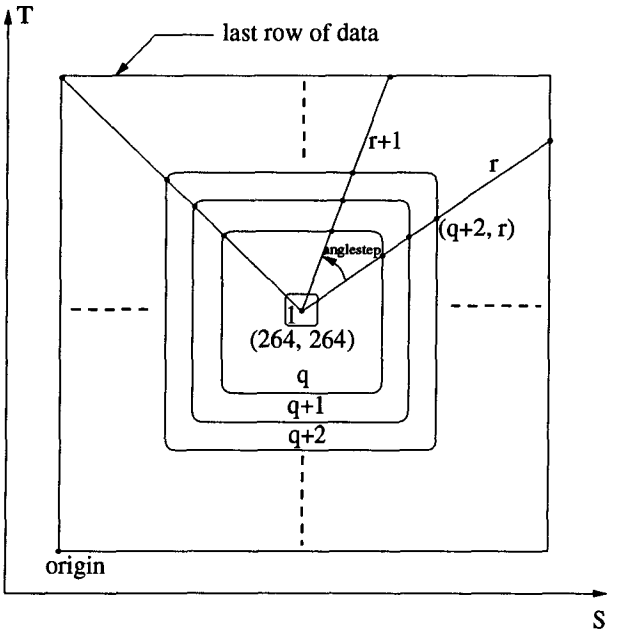


Plate 3.3: The initial mapping from data space to parametric space. Each row of data is mapped onto a square in parametric space starting from the center to the edge.

A parametric space ST (same as the UV space used before) is introduced here for initial parameterization of the data. The ST space sits under the UV space which is used for the final data parameterization. The ST space acts as the rubber sheet here. The range of the region is

from 0 to $2M$ where M is the number of rows of data. In the case of Victor Hugo, there are 264 rows. We assign the lower left corner of S to be the origin, so the center of S is at $(264, 264)$.

If we view the mapping by column, each column of data forms a ray emitted from the center of S and ended at the point where it intersects with the boundary of S (Figure 3.3). Each ray spans a constant of angle *anglestep* which is determined by the number of columns N. First, we calculate the length of the ray segment. The M data points sitting on the ray segment divide it into M sub-segments with equal *spacing*. Here is the algorithm for calculating the initial parametric value pair (s, t) in parametric space $ST(s, t)$ for each data point $D_{ij} \in R^2$:

Algorithm 3.1 Procedure *InitialMap*

```
anglestep = 360.0 / N;

for column = 1 to N do
    angle = column * anglestep;
    for row = 1 to M do
        if(angle <= 45.0 || angle > 315.0)
            s = 264.0 + row;
            t = 264.0 + row * tan(angle);
        if(angle > 45.0 && angle <=135.0)
            s = 264.0 + row / tan(angle);
            t = 264.0 + row;
        if(angle > 135.0 && angle <=  225.0)
            s = 264.0 - row;
            t = 264.0 - row * tan(angle);
        if(angle > 225.0 && angle <= 315.0)
            s = 264.0 - row / tan(angle);
            t = 264.0 - row;
```

```
        end.
    end.
```

The (s,t) pair is not the final parametric value used to form the fitting equations. The ST space is an intermediate space where the mapping deformation occurs.

3.3 Deforming Parametric Space

3.3.1 Deformation Function

The deformation takes place in UV space which sits on top of the ST space introduced in Section 3.2. Note that we currently have three different spaces to manipulate, the data space which belongs to R^3, the intermediate space ST and parametric space UV.

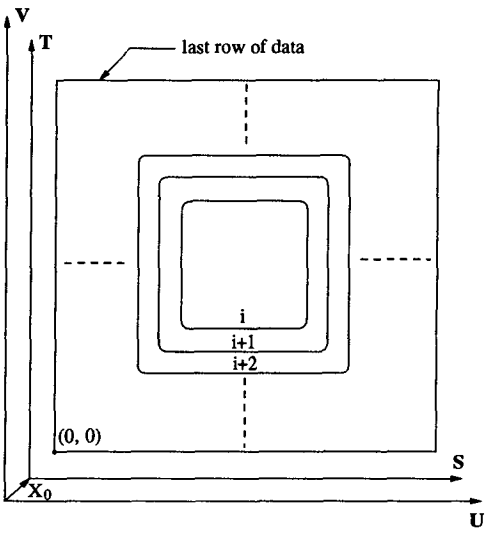


Plate 3.4: Two coordinate frames imposed on the parametric space.

The deformation is defined in terms of a tensor-product bivariate Bernstein polynomial. We impose a grid of *control points* P_{ij} on the square region in ST space (a control point is defined to be the point lying on the control grid and should be distinguished from control vertex which

is a spline control vertex). They form $m+1$ lines in the S direction, and $n+1$ lines in the T direction (Figure 3.5). Their locations are defined to be:

$$P_{ij} = X_0 + \frac{i \times 528}{m}\vec{\mathbf{S}} + \frac{j \times 528}{n}\vec{\mathbf{T}}. \tag{3.1}$$

where 528 is the length and width of the parametric region.

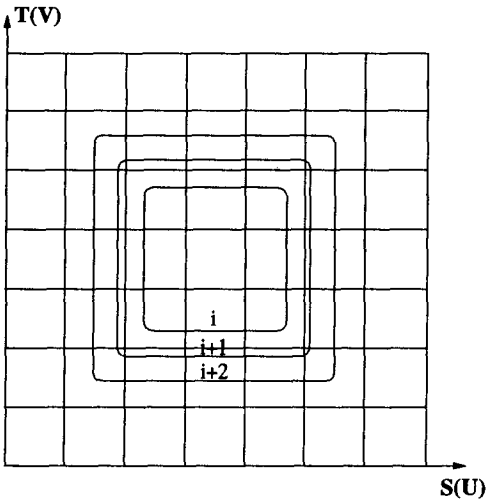


Plate 3.5: A control grid imposed on the parametric space with grid size of $m = n = 7$. Each control point is undeformed and has the same coordinates in UV space as in ST space.

When first defined, each (u, v) of P_{ij} pair is equal to its (s, t) pair. Actually this is also true for any data point in R^2. The deformation is achieved by moving the control points from their undisplaced grid positions. The deformed position X' in ST of an arbitrary point $X = (u, v)$ in UV space is found by first computing its (s, t) coordinate when there is no deformation. Under this condition our UV coordinate system is identical to our intermediate ST coordinate system, $(s, t) = (u, v)$. Then the vector valued bivariate Bernstein polynomial is evaluated:

$$X' = \sum_{i=0}^{m} \binom{m}{i} (528 - s)^{m-i} s^i \sum_{j=0}^{n} \binom{n}{j} (528 - t)^{m-i} t^j P_{ij} \tag{3.2}$$

where P_{ij} is a vector containing the Cartesian coordinates of the control point in ST space.

3.3.2 Refinement and Local Control

The control points here act as the coefficients of a Bernstein polynomial. Since the Bernstein polynomial is the basis function for Bézier curves and surfaces, there are meaningful relationships between the deformation and the placement of the control points. The parametric region in ST space can be viewed as a Bézier surface patch defined by these control points. The order of the basis function is defined by m and n. Whenever we move a control point, we are reshaping the Bézier surface by moving a control vertex.

A Bézier surface has the property of interpolating the boundary points. This is important because we can constrain the boundary of our control grid to the boundary of our parametric region. Consequently, the movement of interior control points or boundary control points will only effect the parametric mapping inside the parametric region. Otherwise, the boundary of our parametric region will not be stable and would not provide a constant size parametric space for mapping the data. For a B-spline formulation, it is known that the surface does not interpolate the boundary control vertices. Compared with the B-spline basis functions, the Bézier surface's Bernstein basis functions save the computational cost of calculating the boundary of a surface patch and constrain the surface points by interpolating the boundary control vertices automatically.

Bilinear interpolation is adequate for our particular application which means both m and n Equation 3.2 are chosen as 1. In order to form the chord length parameterization, the control grid is refined and the control points are displaced. The following section provides the criteria for grid refinement and control point displacement.

3.3.3 Deformation

Discrete Gradient

To construct a chord-length parameterization, the first step is to calculate how much variation there is in the digitized data.

Definition 3.1 *The discrete gradient with respect to i in parametric direction g_i at data point D_{ij} is defined as*

$$g_i = D_{ij} - D_{i-1,j} \quad 2 \leq i \leq M; 1 \leq j \leq N \tag{3.3}$$

In the same way, g_j is defined as

$$g_j = D_{ij} - D_{i,j-1} \quad 1 \leq i \leq M, 2 \leq j \leq N \tag{3.4}$$

We define the sum of $g_i + g_j$ as the discrete gradient *at a data point D_{ij}.* □

Without loss of generality, let us consider the i index direction. Recall that the magnitude of point D_{ij} is the radial distance from the central axis to the surface. Given the distance z between two adjacent sample points, then the distance between two adjacent data points $D_{i-1,j}$ and D_{ij} is $d = \sqrt{z^2 + g_i^2}$[1]. Because of the fine sampling grid from the digitizer, the magnitude of z is constant and is usually very small compared to the magnitude of the non-zero g_i's. So we can approximate the above formula as $d = g_i$.

This implies that if we direct more parametric lines to those high gradient areas, the data points that are farther apart will also be farther apart in parametric space. This is the general idea of chord length parameterization. If the refinement of the control grid over parametric space is fine enough, more control points will be generated and displaced. Thus we can achieve the chord length parameterization within a certain tolerance.

Refinement and Free Control Points

The deformation is built starting from a single bilinear patch defined by four control points $P^1_{00}, P^1_{10}, P^1_{01}$ and P^1_{11} making up the control quadrilateral. These four points are not displaced since they bound the region being approximated. Thus all points P^1_{ij} have the same (s,t) pairs in ST space as the corresponding (u,v) pairs in UV space.

Each refinement locally subdivides the 1-level control quadrilateral and generates four smaller control quadrilaterals residing at level 2. At level 1, the criterion to decide whether

[1] Since the sample points are really fine, we neglect that the surface is curved and take it as flat.

or not to refine a control quadrilateral is the discrete gradient sum inside it. First, the overall discrete gradient sum SG for all of the data points which cover the whole parametric area E are calculated. Second, the tolerance threshold T for each control quadrilateral with area A is set as

$$T = SG\frac{A}{E}. \tag{3.5}$$

Whenever the gradient sum inside a control quadrilateral exceeds its tolerance, it is refined; otherwise, it is only refined if the user interactively refines it by selecting the region and then clicking on the refine button.

The refinement is performed in UV space. The mid-point subdivision technique is adopted for simplicity. As shown in Figure 3.6 in UV space, the original control quadrilateral is evenly split into four equal sub-control quadrilaterals.

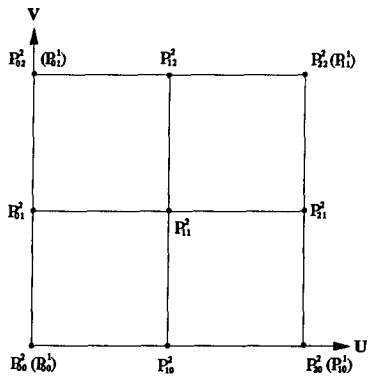


Plate 3.6: The control grid in UV space is uniformly refined from a 2×2 grid into a 3×3 grid. The control points inside brackets are the indices of the 4 control points as they are indexed at level 1.

So we have the default coordinates for newly generated control points as

$$
\begin{aligned}
\begin{pmatrix} u \\ v \end{pmatrix}_{P^2_{00}} &= \begin{pmatrix} u \\ v \end{pmatrix}_{P^1_{00}}, \\
\begin{pmatrix} u \\ v \end{pmatrix}_{P^2_{01}} &= \tfrac{1}{2}\left(\begin{pmatrix} u \\ v \end{pmatrix}_{P^1_{00}} + \begin{pmatrix} u \\ v \end{pmatrix}_{P^1_{01}}\right), \\
\begin{pmatrix} u \\ v \end{pmatrix}_{P^2_{02}} &= \begin{pmatrix} u \\ v \end{pmatrix}_{P^1_{01}}, \\
\begin{pmatrix} u \\ v \end{pmatrix}_{P^2_{10}} &= \tfrac{1}{2}\left(\begin{pmatrix} u \\ v \end{pmatrix}_{P^1_{00}} + \begin{pmatrix} u \\ v \end{pmatrix}_{P^1_{10}}\right), \\
\begin{pmatrix} u \\ v \end{pmatrix}_{P^2_{11}} &= \tfrac{1}{4}\left(\begin{pmatrix} u \\ v \end{pmatrix}_{P^1_{00}} + \begin{pmatrix} u \\ v \end{pmatrix}_{P^1_{01}} + \begin{pmatrix} u \\ v \end{pmatrix}_{P^1_{10}} + \begin{pmatrix} u \\ v \end{pmatrix}_{P^1_{11}}\right), \\
\begin{pmatrix} u \\ v \end{pmatrix}_{P^2_{12}} &= \tfrac{1}{2}\left(\begin{pmatrix} u \\ v \end{pmatrix}_{P^1_{01}} + \begin{pmatrix} u \\ v \end{pmatrix}_{P^1_{11}}\right), \\
\begin{pmatrix} u \\ v \end{pmatrix}_{P^2_{20}} &= \begin{pmatrix} u \\ v \end{pmatrix}_{P^1_{10}}, \\
\begin{pmatrix} u \\ v \end{pmatrix}_{P^2_{21}} &= \tfrac{1}{2}\left(\begin{pmatrix} u \\ v \end{pmatrix}_{P^1_{10}} + \begin{pmatrix} u \\ v \end{pmatrix}_{P^1_{11}}\right), \\
\begin{pmatrix} u \\ v \end{pmatrix}_{P^2_{22}} &= \begin{pmatrix} u \\ v \end{pmatrix}_{P^1_{11}}
\end{aligned}
$$

The (s,t) pairs for these newly generated control points can be derived from Equation 3.2 by substituting the (u,v) coordinates calculated above.

As shown in Figure 3.6, after refinement there are 9 control points altogether at the second level. The four corner P^2_{ij}'s stay at the same positions as the four P^1_{ij}'s and are not be repositioned any more. This leaves 5 new control points that can be moved to deform the parametric space. We will refer these control points as *free control points.*

Free Control Point Displacement

We will now discuss how to deform the parametric space by moving the free control point. As shown in Figure 3.6, we have five control points which can be displaced, namely P^2_{10}, P^2_{01}, P^2_{11}, P^2_{21} and P^2_{12}.

To displace a free control point, for example P_{10}^2, the first thing is to calculate the discrete gradient sum along the chord $P_{00}^1 P_{10}^1$. This is done by sampling along the chord $P_{00}^1 P_{10}^1$ and obtaining the discrete gradient value at each point and then summing them up. The sampling rate is constant. This means that the number of points sampled (M_S) is proportional to the length of the chord $P_{00}^1 P_{10}^1$ in ST space. Each sampled point (s,t) is mapped back into discrete data space. This can be assured to be a one-to-one mapping by "inversely" applying Algorithm 3.1 since this algorithm is one-to-one. We sum up the discrete gradient values at each sampled data point S_i by:

$$S = \sum_{i}^{M_S} g_{S_i} \tag{3.6}$$

to obtain the sum S.

As shown in Figure 3.7, each sampled point S_i will fall into a quadrilateral formed by four data points $D_{ij}, D_{i+1,j}, D_{i+1,j+1}, D_{i,j+1}$. There are three cases for this point to be located in the quadrilateral:

- At a data point (a corner).
- On an edge.
- Inside the quadrilateral.

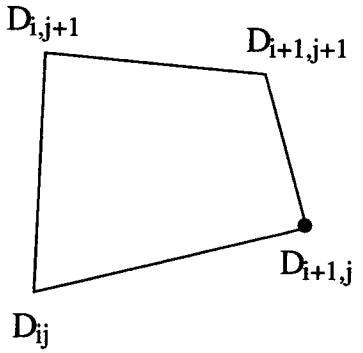
In the first case, we take the discrete gradient at the data point. For the other two cases, we assume that the data space is continuous and use bilinear interpolation to calculate the discrete gradient at the sampled position. For example, we get $D_{i+\delta i,j+\delta j}$ when we map S_i to data space. Consider the second case and suppose $\delta j = 0$. Then the gradient will be:

$$(1-\delta i)g_{ij} + g_{i+1,j}\delta i \quad 0 \leq \delta i < 1. \tag{3.7}$$

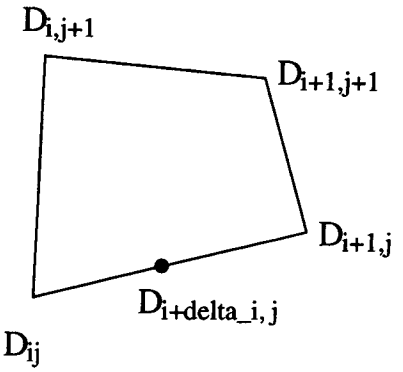
In the third case, when neither δi nor δj is zero, the gradient value should be calculated as:

$$(1-\delta i)(1-\delta j)g_{ij} + (1-\delta i)\delta j g_{i,j+1} + \delta i(1-\delta j)g_{i+1,j} + \delta i \delta j g_{i+1,j+1}. \tag{3.8}$$

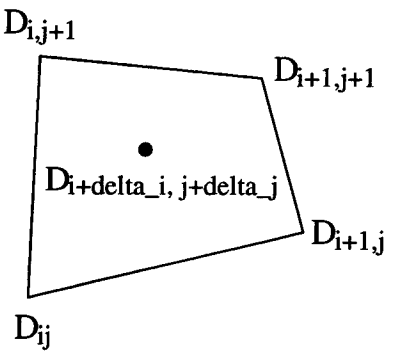
After the gradient sum is found along a chord, the gradient midpoint O at which the discrete gradient $S_O = \frac{1}{2}S$ is found. Then control point P_{10}^2 is moved to the position O



a) Case 1: The sample point is one of the corner points.



b) Case 2: The sample point is on an edge of the data quadrilateral.



c) Case 3: The sample point is in the data quadrilateral.

Plate 3.7: The three cases of the sample point falling into a data quadrilateral.

to balance the discrete gradient sum on either side of P_{10}^2. Note the this displacement of P_{10}^2 happens in ST space. Figure 3.8 shows that after P_{10}^2 is moved toward P_{20}^2, the sub-quadrilateral $P_{10}^2 P_{20}^2 P_{21}^2 P_{11}^2$ will cover less discrete data points in data space than the adjacent sub-quadrilateral $P_{00}^2 P_{10}^2 P_{11}^2 P_{01}^2$. In UV space, these two quadrilaterals cover the same amount of parametric space. This means we have more parametric space to cover the high gradient areas in data space.

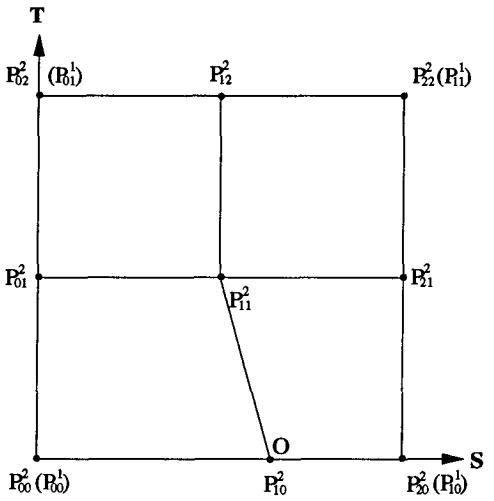


Plate 3.8: A control point is displaced from its original position.

The other three *on-edge* control points $P_{01}^2, P_{12}^2, P_{21}^2$ are displaced in the same way to balance the gradient sum along that edge on which it sits.

Control point P_{11}^2 is displaced based on the same idea of discrete gradient sum balance. An iterative process is designed to compute the approximate point position $B = (s, t)$ such that the discrete gradient sums inside each of the four sub-quadrilaterals surrounding B are very close to each other. The area discrete grid sum is calculated by expanding Equation 3.6 from one dimension to two dimensions:

$$S_{area} = \sum_{i=1}^{M_S} \sum_{j=1}^{N_S} g_{S_{ij}} \tag{3.9}$$

where M_S and N_S are the number of sample points in the U and V parametric directions respectively, $g_{S_{ij}}$ is the discrete gradient value at sample point S_{ij} and S_{area} is the total discrete gradient sum of the area.

If we refine the grid of control points enough times, a near chord-length parameterization is obtained for those areas with a fine control grid. Color plate 2 shows the control grid after five levels of refinement.

3.4 Sampling Data Points

The FMG method requires gridded data, but because of the odd configuration of the spline surface, the image of the gridded data in the parametric space is anything but gridded.

To obtain gridded data points for the right hand side of Equation 2.14, the parametric domain of the spline (UV space) is sampled on a grid and mapped back onto data space through the intermediate ST space.

There are X and Y sampling points in the U and V directions respectively (Figure 3.9). In practice the values of X and Y depend on the density of the original data. The step size was chosen as 257 in both U and V to provide a sufficiently dense sampling to catch most of he details of the surface and to provide the ratio of 2 : 1 for full multigrid methods. Thus we have $X = Y = 257$ and $ustep = vstep = \frac{528}{264}$.

We now have the parameter set $\{(u_{ij}, v_{ij}) = (i \times ustep, j \times vstep) | 0 \leq i \leq X, 0 \leq j \leq Y\}$ to find the 3-D data position associated with each (u_{ij}, v_{ij}) pair.

Each (u_{ij}, v_{ij}) is mapped onto the ST space to obtain the corresponding (s_{ij}, t_{ij}) pair using Equation 3.2. This is a one-to-one mapping. In the same way that we calculated the discrete gradient, the (s_{ij}, t_{ij}) pair is mapped into data space and the real pair $(i+\delta i, j+\delta j)$ is returned. Here i and j are the indices for an existing data point D_{ij}. Bilinear interpolation (Equation 3.7 and Equation 3.8) was used to calculate the exact value at $D_{i+\delta i, j+\delta j} = Z_{qr}$ from the four existing data points $D_{ij}, D_{i,j+1}, D_{i+1,j}$ and $D_{i+1,j+1}$. This 3-D coordinate triple (x_{ij}, y_{ij}, z_{ij}) associated with parameter pair (u_{ij}, v_{ij}) was used to generate the system of linear equations in

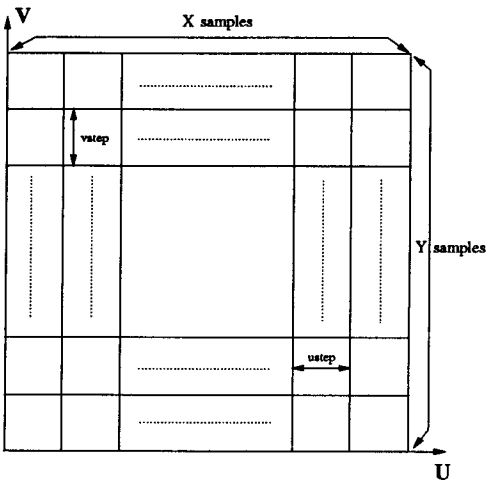


Plate 3.9: The UV space is evenly sampled for produce the parametric pair set Ψ.

Equation 2.14.

With respect to the original data set, the parameterization is near chord-length. The new data set also approximates the original surface with approximately the same number of points. Feeding these data points into the FMG solver, we can solve the equations for the control vertices which define the parametric B-spline surface. This will be discussed in the following chapter.

3.5 Summary

A parametric space deformation technique based on image warping was introduced. The technique deforms the isoparametric lines to follow rough regions in the raw data. The deformation was controlled by control points that form a control grid in parametric space that can be refined and displaced. The gradient distribution criterion used to move a control point generates a near chord-length parameterization. The parametric space after deformation was sampled evenly and then mapped onto the data space. Bilinear interpolation is used to obtain the exact position of the data point associated with parameter pair.

Chapter 4

Multi-resolution Surface Fitting Using Multigrid

4.1 Deriving Equations and Constraints

The remaining problem is to solve the set of linear equations to obtain a surface that interpolates the gridded data points $\{Z_{ij} \in R^3 | 0 \leq i \leq X, 0 \leq j \leq Y\}$ The control vertices V are the unknowns and the data points Z are knowns. For each data point Z_{ij}, the interpolation condition can be expressed as an equation in terms of the unknown control vertices. The number of equations should be equal to the number of data points.

4.1.1 Data Point Distribution

The data points to be interpolated must now be distributed among the patches to make a correspondence from a data point to a surface point. As shown in Figure 4.1, if we sample at the four corners of each patch, $X \times Y$ patches provide enough freedom for interpolating the samplings. For bicubic B-spline surface, $X \times Y$ patches are defined by $(X+3) \times (Y+3)$ control vertices. Therefore, additional $2X+2Y$ equations have to be derived based on boundary conditions to determine the unique solution of Equation 2.14.

The additional equations will be covered in Section 4.1.3.

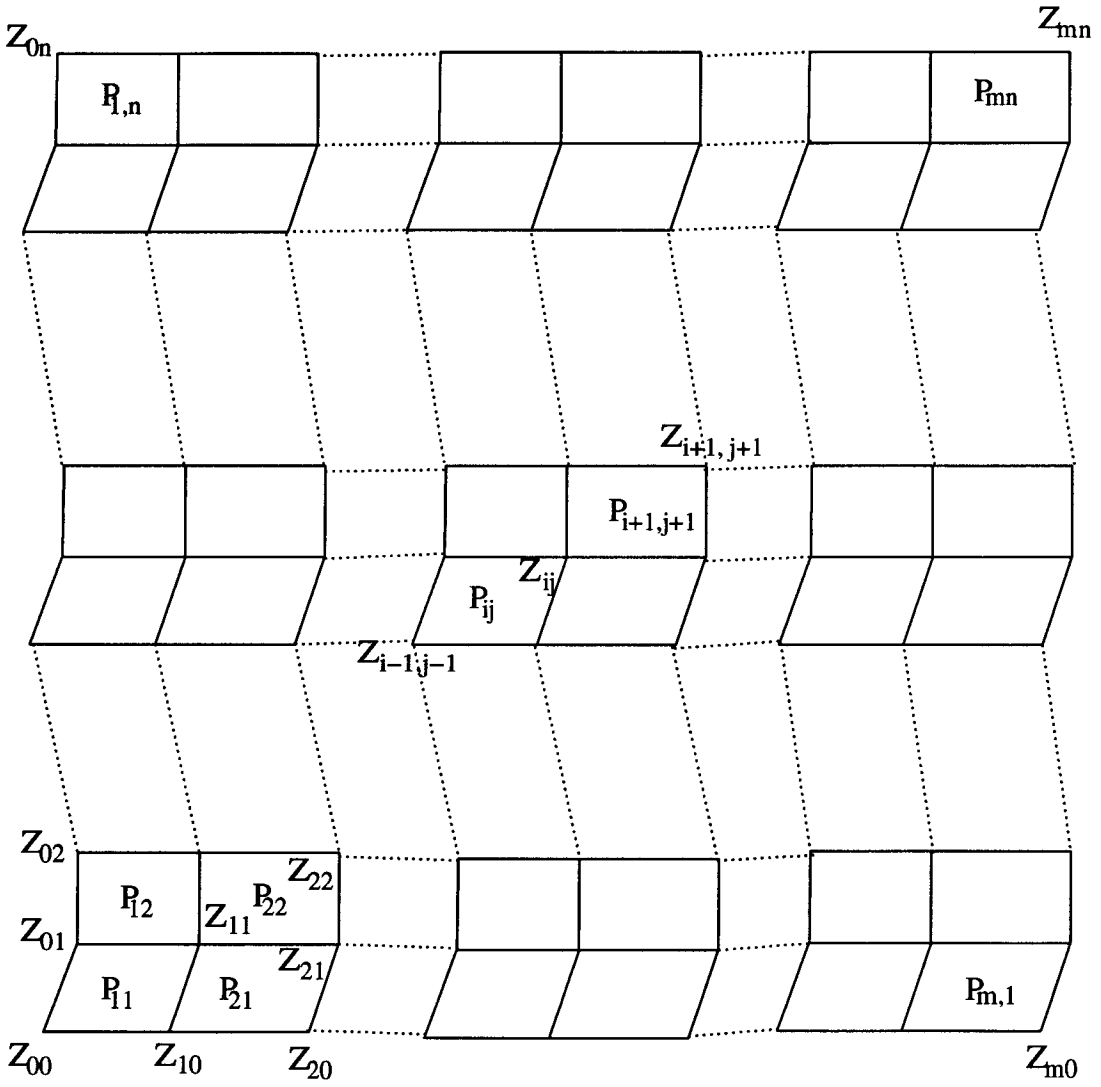
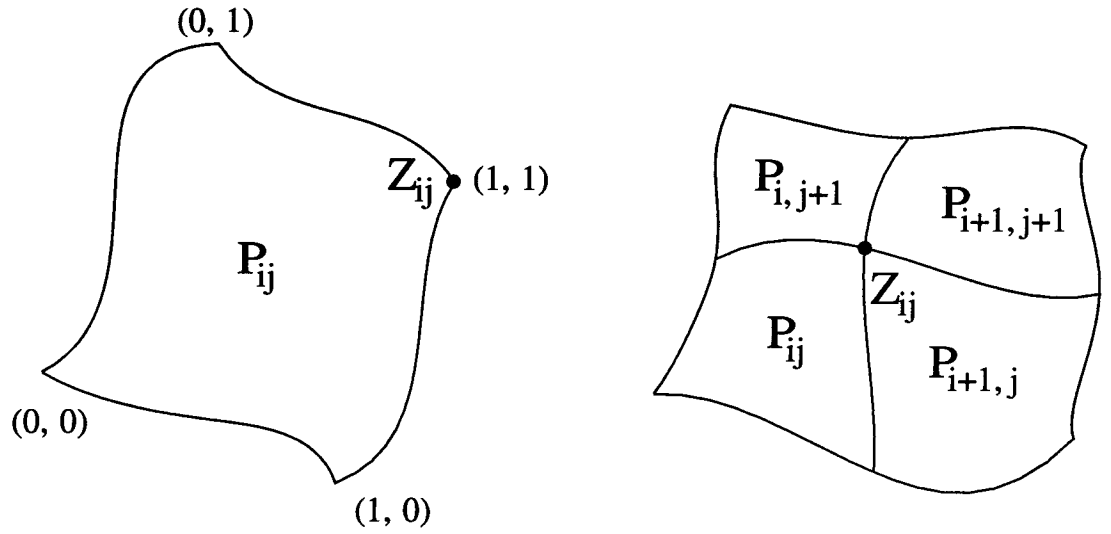


Plate 4.1: The $(X + 1) \times (Y + 1)$ data points are mapped onto $X \times Y$ patches. Every patch corner corresponds to a data point.

4.1.2 Deriving the Positional Interpolation Equations

For each data point, the corresponding parametric location in the patch is $(1, 1)$ (Figure 4.2).



a) A single patch.

b) A data point is distributed among patches.

Plate 4.2: A data point Z_{ij} is mapped onto the upper right corner of a single patch. At the same time, it is the corner point of three adjacent patches.

Substitute $u = 1$ into Equation 2.2, we have:

$$\left[\begin{array}{cccc} B_{-2}(1) & B_{-1}(1) & B_0(1) & B_1(1) \end{array}\right] = \left[\begin{array}{cccc} 0 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}\right]$$

Use this precalculated basis function value in Equation 2.1, we have:

$$\begin{aligned} P_{ij}(1,1) = \; & \tfrac{1}{36}(V_{i-1,j-1} + 4V_{i-1,j} + V_{i-1,j+1} + 4V_{i,j-1} + 16V_{i,j} \\ & +4V_{i,j+1} + V_{i+1,j-1} + 4V_{i+1,j} + V_{i+1,j+1}) \\ & i = 1, \ldots, m-1 \;\; and \;\; j = 1, \ldots, n-1. \end{aligned} \tag{4.1}$$

According to the mapping from data points to surface points, $P_{ij}(1,1) = Z_{ij}$. We can express Equation 4.1 in matrix form as:

$$Av = d \tag{4.2}$$

where v is the vector containing the control vertices from V_{11} to $V_{X-1,Y-1}$ and d is the vector containing the data points transformed by certain control vertices (certain unknown control vertices are moved to right hand side to make the matrix A banded). The corresponding matrix A is

$$\begin{pmatrix}
16 & 4 & & & & & 4 & 1 & & & \\
4 & 16 & 4 & & & & 1 & 4 & & & \\
 & \ddots & \ddots & \ddots & & & & \ddots & \ddots & \ddots & \\
 & & 4 & 16 & 4 & & & & 1 & 4 & \\
 & & & 4 & 16 & & & & & & \\
4 & 1 & & & & 16 & 4 & & & & \\
1 & 4 & 1 & & & 4 & 16 & 4 & & & \\
 & \ddots & \ddots & \ddots & & & \ddots & \ddots & \ddots & & \\
 & & 1 & 4 & 1 & & & 4 & 16 & 4 & \\
 & & & 1 & 4 & & & & 4 & 16 & \\
 & & & & & \ddots & & & & & \ddots
\end{pmatrix}$$

Note this matrix is symmetric, positive definite and thus it is well-conditioned. It is also diagonally strong which makes it possible for relaxation scheme to converge. The matrix is sparse which is the key condition for multigrid solver to perform well.

The above equation is a set of $(X-1) \times (Y-1)$ equations, one for each data point. These equations are expressed in $(X+1) \times (Y+1)$ unknown control vertices.

4.1.3 Deriving the Boundary Conditions

In order to completely determine the unknown control vertices, an additional $2X+2Y$ equations are required. Each of these equations corresponds to an extra condition imposed on each of the $2X+2Y$ boundary data points. The boundary data points are:

$$\begin{aligned}
&Z_{0,j} \quad j = 0, \ldots, n-1; \\
&Z_{i,n} \quad i = 0, \ldots, m-1; \\
&Z_{m,j} \quad j = 1, \ldots, n; \\
&Z_{i,0} \quad i = 1, \ldots, m.
\end{aligned}$$

The remaining degrees of freedom are determined by specifying the tangents and curvature at the edge of the surface which interpolates the boundary data points. At each boundary data point, the curvature is set to be zero and the surface tangent to the control graph.

The following equations are derived for each corner data point:

$$V_{00} = Z_{00} \quad V_{0n} = Z_{0Y} \quad V_{X0} = Z_{m0} \quad V_{XY} = Z_{XY} \tag{4.3}$$

Other boundary data points fall into the following equations:

$$\begin{array}{ll} Z_{0j} = \frac{1}{6}(4V_{0j} + V_{0,j-1} + V_{0,j+1}) & for 1 \leq j \leq Y-1 \\ Z_{i0} = \frac{1}{6}(4V_{i0} + V_{i-1,0} + V_{i+1,0}) & for 1 \leq i \leq X-1 \\ Z_{iY} = \frac{1}{6}(4V_{iY} + V_{i-1,Y} + V_{i+1,Y}) & for 1 \leq i \leq X-1 \\ Z_{Yj} = \frac{1}{6}(4V_{Xj} + V_{X,j-1} + V_{X,j+1}) & for 1 \leq j \leq Y-1 \end{array} \tag{4.4}$$

The corresponding matrix for these equations is:

$$\begin{pmatrix} 4 & 1 & & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & 4 & 1 & \\ & & & & 1 & 4 & \end{pmatrix}.$$

Up until now, for the 265×265 data points we derived by sampling the parametric space, we had 265×265 linear equations, one for each data point. These 265×265 linear equations involve 265×265 unknown B-spline control vertices which define the final approximating surface. In the following section, we will discuss the solution of these equations.

4.2 Solving For Control Vertices

4.2.1 Full Multigrid Method

The system of linear equations defining the approximation is solved using the full multigrid method. The benefits are two-fold:

- The excellent numerical behaviour of FMG provides a fast and stable way of solving these equations.
- FMG method provides a multi-resolution surface which can be used to directly create the levels in a hierarchical spline surface.

The FMG method was described in Section 2.5.7 showing that it is an excellent candidate for a well conditioned sparse matrix on a uniform grid.

4.2.2 Full Multigrid Operators

In the following several sections, we will discuss the relaxation schemes, coarsest level solver, intergrid operators and the interpolation operator used in our implementation.

Relaxation Operator

From the matrix corresponding to Equation 4.1, we know that it is a nine point formula [1] on the uniform data point grid. The general practice has been to use the *Red-Black Gauss-Seidel* relaxation scheme for the five point formula. This leads us to choose the *four colour Gauss-Seidel* relaxation scheme for the nine point formula because one quarter of the residuals calculated after post-relaxation are zero [21].

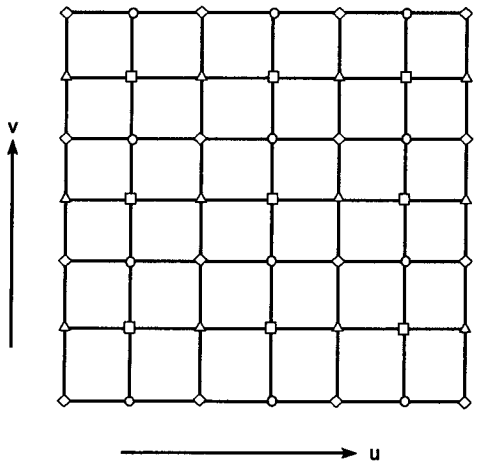


Plate 4.3: A illustration of the four-colour relaxation scheme on a uniform grid. First the ◇ points are swept, then the ○ points, △ points and finally □ points.

The four colour relaxation scheme is similar to the Red-Black Gauss-Seidel relaxation, shown in Figure 4.3. We divide the whole set of data points on the uniform grid into four sub-sets and relax them in turn. The four sub-sets of data points are represented with ◇, ○, □ and △

[1] If we take the control vertices as unknowns, we call an equation composing a matrix a nine point formula if each linear equation contains nine unknowns, which are adjacent to each other on a grid.

respectively. First all the $\diamond$ points (which contain every other original data point on the grid in both the u and v parametric directions.) are relaxed (Gauss-Seidel relaxation) one by one. Then the $\bigcirc$ points, $\square$ points and $\triangle$ points are relaxed in the same way in turn.

From Figure 4.3, we can see each $\diamond$ point being relaxed is only calculated from non-$\diamond$ data points. Thus before all of the $\diamond$ points are totally relaxed, they do not refer to each other. If one has a multi-processor machine, it is possible to parallelize the relaxation process which greatly speeds up the whole multigrid cycle since most of the computation cost is spent on relaxation.

Another relaxation scheme implemented is *line Gauss-Seidel relaxation* based on the following analogy. In parametric space, these gridded control vertices in both u and v direction have very clear geometric meaning. In the u direction, each "line" of control vertices defines a B-spline curve residing inside the surface which is a blending of curves along u isoparametric direction with the curves along the v isoparametric direction. The surface is defined by the whole set of control vertices which can also be considered to be a blending. The surface is called a tensor product of those single curves. The line Gauss-Seidel relaxation method relaxes the u direction unknowns and then the v direction unknowns "line" by "line". Because the unknowns are coupled closely in both u and v directions, this method is very effective in reducing the residuals.

Coarsest Level Operator

The coarsest level operator MR (Algorithm 2.1) is Gauss-Seidel relaxation. The explicit equation for relaxation is:

$$v_i^{j+1} = \frac{1}{l_{ii}} \left(d_i - \sum_{k<i} l_{ik} v_i^{j+1} - \sum_{k>i} l_{ik} v_i^j \right). \tag{4.5}$$

Experiments indicate that to reduce the residual down to be 0.001% of the exact solution, only 11 iterations are required at the coarsest level. So the convergence rate is very fast.

Prolongation Operator

The most common prolongation method is interpolation. A common prolongation operator is *bilinear* interpolation. (Others may be found in [21]) The operator is expressed as:

$$\left]\begin{array}{ccc} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{array}\right[$$

which is called the *stencil* form of the operator. Stencils representing maps of Ω^{2h} into Ω^h use reverse square brackets. A stencil defined on the fine grid is centered at a point present in coarse grid. Each entry indicates the contribution of the coarse grid point to the three (or nine) corresponding fine grid points. Specifically the central entry is the coefficient of $v^h_{2i,2j}$ in the formula for v^h_{ij}.

Restriction Operator

There are two types of restriction operators that are commonly used, injection and full weighting. (In 2-D, injection is defined by $v^H_{ij} = v^h_{2i,2j}$.) In 2-D the *full-weighting* operator. In 2-D, it is expressed as

$$\begin{array}{l} v^H_{ij} = \frac{1}{16}(v^h_{2i+1,2j+1} + v^h_{2i+1,2j-1} + v^h_{2i-1,2j+1} + v^h_{2i-1,2j-1} + \\ 2(v^h_{2i,2j+1} + v^h_{2i,2j-1} + v^h_{2i+1,2j} + v^h_{2i-1,2j+1}) + 4v^h_{2i,2j}) \end{array} \tag{4.6}$$

The *stencil* forms of these operators are as follows:

$$\left]\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{array}\right[$$

and

$$\left]\begin{array}{ccc} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{array}\right[$$

respectively.

Thus, for injection, the value at the coarse grid points is just the value at the corresponding fine grid point. Full-weighting uses a weighted average of the values at the corresponding fine

grid point and its nearest neighbours. In our implementation, the full-weighting operator was chosen.

The stencils above are for an interior point which has all non-empty neighbouring points. At a boundary point which is not a corner point, there are only 5 neighbours instead of 8, so the stencil in this case is:

$$\begin{bmatrix} 0 & \frac{1}{6} & \frac{1}{12} \\ 0 & \frac{1}{3} & \frac{1}{6} \\ 0 & \frac{1}{6} & \frac{1}{12} \end{bmatrix}.$$

For a corner point, there are only three neighbours, thus the stencil is:

$$\begin{bmatrix} 0 & \frac{2}{9} & \frac{1}{9} \\ 0 & \frac{4}{9} & \frac{2}{9} \\ 0 & 0 & 0 \end{bmatrix}.$$

Interpolation Operator

The *interpolation* operator takes a solution at the coarser level and interpolates it to a finer level between adjacent V-cycles. As suggested by Brandt [7], bicubic interpolation is a better candidate for interpolation operator than bilinear interpolation. Because it introduces less high frequency errors in the initial guess at a finer level than bilinear interpolation does. Thus, a better initial guess can be obtained at a finer level from the interpolated result.

For an interior grid point (i, j) with grid spacing h where i is odd and j is even, the points $(i+1, j)$ and $(i+3, j)$ are coarser grid points. The interpolated value for (i, j) is:

$$V^l(i,j) = -\frac{1}{16}V^{l-1}(i-3,j) + \frac{9}{16}V^{l-1}(i-1,j) + \frac{9}{16}V^{l-1}(i+1,j) - \frac{1}{16}V^{l-1}(i+3,j) \quad (4.7)$$

We can repeat the interpolation process with respect to the v direction at (i, j) with j odd. Applying this interpolation to a vector that is 1 at some interior point and vanishes elsewhere, we obtain the matrix [21]:

$$\begin{pmatrix} \frac{1}{16^2} & 0 & -\frac{9}{16^2} & -\frac{1}{16} & -\frac{9}{16^2} & 0 & \frac{1}{16^2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{9}{16^2} & 0 & \frac{81}{16^2} & \frac{9}{16^2} & \frac{81}{16^2} & 0 & -\frac{9}{16^2} \\ -\frac{1}{16} & 0 & \frac{9}{16} & 1 & \frac{9}{16} & 0 & -\frac{1}{16} \\ -\frac{9}{16^2} & 0 & \frac{81}{16^2} & \frac{9}{16^2} & \frac{81}{16^2} & 0 & -\frac{9}{16^2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{16^2} & 0 & -\frac{9}{16^2} & -\frac{1}{16} & -\frac{9}{16^2} & 0 & \frac{1}{16^2} \end{pmatrix}.$$

For a boundary grid point, Equation 4.7 is not be applied since one or two terms are missing. It is more convenient to use the one-sided quadratic interpolation in the interior domain which yields

$$V^l(ih, jh) = -\frac{1}{8}V^{l-1}(ih - 3h, jh) + \frac{3}{4}V^{l-1}(ih - h, jh) + \frac{3}{8}V^{l-1}(ih + h, jh) \qquad (4.8)$$

which leads to matrix:

$$\begin{pmatrix} 0 & 0 & -\frac{3}{64} & -\frac{1}{8} & -\frac{3}{32} & 0 & \frac{1}{64} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{9}{32} & \frac{3}{4} & \frac{9}{16} & 0 & -\frac{3}{32} \\ 0 & 0 & \frac{3}{8} & 1 & \frac{3}{4} & 0 & -\frac{1}{8} \\ 0 & 0 & \frac{9}{64} & \frac{3}{8} & \frac{9}{32} & 0 & -\frac{3}{64} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Suppose we have $X \times Y$ solutions after we complete the V-cycle for the coarser level. The interpolation operator will generate $(2X-1) \times (2Y-1)$ "control vertices" as an initial guess for upper level from the $X \times Y$ solutions. Using interpolation, the geometric shape of the surface defined by the $X \times Y$ control vertices at the coarser level is different than the surface defined by the $(2X-1) \times (2Y-1)$ control vertices. This suggests that a more suitable alternative for interpolation is B-spline surface subdivision. (see Chapter 2) Midpoint subdivision will generates $(2X-1) \times (2Y-1)$ control vertices from $X \times Y$ ones. The central $(2X-3) \times (2Y-3)$ control vertices define a surface which consists of as four times many patches as the surface defined by the $X \times Y$ control vertices. Moreover, both surfaces have exactly the same shape. Since we expect that different levels of the surface are converging to the final approximation shape at the finest level, we prefer using the exact shape generated at coarser level as the initial guess at finer level to using the shape interpolated to finer level by bicubic interpolation (the latter one could add high frequency components). The user will not experience unusual changes in shape which can be introduced by the bicubic interpolation.

Now we give a general analysis of the computation cost of both bicubic interpolation and mid-point subdivision. For each new control vertex created by bicubic interpolation, nine neighbouring points are involved. For the subdivision scheme, an even indexed control vertex is calculated from two neighbouring data points and an odd indexed control vertex is calculated

from three neighbouring points. Based on the computation cost of each generated control vertex, subdivision is less expensive. However, bicubic interpolation makes use of all the existing coarser level points and transfers them directly to finer level in constant time without further calculation. Thus bilinear interpolation only needs to compute $\frac{3}{4}$ of the total control points for the finer level, decreasing computation cost. The disadvantages and advantages of this methodology even out in practice. For example, when implementing with a 265×265 grid, the interpolation process costs almost the same amount of time for both methods.

4.3 Performance Evaluation

For the Victor Hugo data set, the FMG method was used to solve a system of 257×257 linear equations using a nine level grid scheme scheme. The first level has 2×2 data points. At the finest level, all of the sampled data points Z_{ij} are interpolated by a surface consisting of 256×256 patches.

The final result must be a multi-resolution surface definition, thus the approximation result at each level must be saved before it is interpolated to finer level. This means that for each level, we need not only a matrix $[V_{ij}]$ to store the unknowns, but also a backup matrix $[VBAK_{ij}]$ to store the fit result after each smaller v-cycle. This backup process is done each time that the fitting at a certain level is done. This means of a cost of $\frac{16}{3}(257 \times 257)$ storage units. Still, it is $O(N)$ cost given the number of data points to be interpolated.

Running on a Silicon Graphics Crimson, solutions to all levels of equations are found in approximately 30 seconds. According to Hackbusch [21], if we set $\nu_1 = \nu_2 = 2$ and $\gamma = 2$ (i.e. use W-cycle instead of V-cycle), the residual r of each iteration is reduced by a factor of $(\nu_1 + \nu_2)^2 = 16$. As shown in the following table, this criteria is met. Table 4.1 also shows the time cost of each iteration at every level. Here time t is in hundredths of seconds, and L and I indicate the level index (the smaller, the coarser) and iteration number respectively. The residual tolerance is set to be 0.0001, which is very small compared to the general magnitude of a control vertex which was 200.

L	I	r	t
1	11	0.0000013258	15
2	1	0.4906180920	22
2	2	0.0276220148	23
2	3	0.0012658379	23
2	4	0.0000636812	23
2	5	0.0000032423	23
3	1	0.5924855008	98
3	2	0.0340934001	97
3	3	0.0018144016	99
3	4	0.0001093170	97
3	5	0.0000067578	97
4	1	0.6936782689	415
4	2	0.0331101110	417
4	3	0.0018249213	416
4	4	0.0001103976	416
4	5	0.0000068922	416
5	1	0.5606160961	1705
5	2	0.0383129169	1687
5	3	0.0019832604	1704
5	4	0.0001116628	1695
5	5	0.0000069643	1704

Table 4.1: The computation cost at each level in FMG.

Since FMG pursues a direct solution to the linear equations instead of an approximate solution like least squares method, noise components in the original data set do not effect the efficiency. However, the algorithm does not have the ability to remove or reduce the effect of noise on the approximation result.

As for the different relaxation and interpolation techniques used inside the FMG cycle, a brief analysis will be provided for comparison. At the finest level, the time used for line Gauss-Seidel is almost the same as for four-colour Gauss-Seidel relaxation methods, which is about 2 seconds for each sweep. Then line Gauss-Seidel method does not smooth the error components as fast as the four-colour Gauss-Seidel. When we used line Gauss-Seidel, more sweeps were required to meet the same residual tolerance. The reason for this phenomena requires more investigation. As expected, the subdivision interpolation method performs better than the bicubic interpolation method because the former one produces a smaller residual at the finer level.

Chapter 5

Hierarchy Building

The FMG approach produces approximation to the data having multiple resolutions. The next step is to generate hierarchical B-spline from the results.

Lyche and Morken [26] work bottom up by first approximating the surface with a fine mesh of spline patches then removing knots in those regions where knot removal will not cause the surface to move out of tolerance.

In the following sections, we are going to present a bottom-up approach for generating the hierarchical representation of the surface that we obtained from the FMG solver.

5.1 Inverse Subdivision

5.1.1 Hierarchical Surface File Format

The hierarchical B-spline surface is generated by generating a file defining a hierarchical B-spline surface from the multi-resolution representation. Recall from Chapter 2 that a hierarchical B-spline surface is very economic in terms of space, we only need to store those non-zero offset vectors and the first level control vertices as reference vectors. The following script file is a typical hierarchical B-spline surface file generated by the *Dragon* [1] [17].

```
Hierarchical B-Spline Save file: Version 9
```

[1] *Dragon* is a Hierarchical B-spline surface editor developed by Dr. David Forsey at University of Waterloo.

```
Surface: 'Plane 0
Root Frame: 0 'Link0
Number of Frames: 1
Number of Offsets: 35
Topology: U0 V0
Symmetry: 1.500000 V
Euler Angle Orientation: 0.000000 0.000000 0.000000
Position: 0.000000 0.000000 0.000000
Offset Data
Level 0 - spacing 1 4x4
U0 0:F  0:-60 0 -60
1:F  0:-20 0 -60
2:F  0:20 0 -60
3:F  0:60 0 -60
U1 0:F  0:-60 0 -20
1:F  0:-20 -25.5 -20
2:F  0:20 -25.5 -20
3:F  0:60 0 -20
U2 0:F  0:-60 0 20
1:F  0:-20 -25.5 20
2:F  0:20 -25.5 20
3:F  0:60 0 20
U3 0:F  0:-60 0 60
1:F  0:-20 0 60
2:F  0:20 0 60
3:F  0:60 0 60
Level 4 - spacing 0.0625
U24 20:T  :29.0296 23.0948 47.4123
```

```
28:T  :29.0296 -23.0948 47.4123
```

This file contains the surface definition as well as a definition for the coordinate frames for an articulated figure. The header file contains the statistical data about the surface such as the number of frames and number of non-zero offsets.

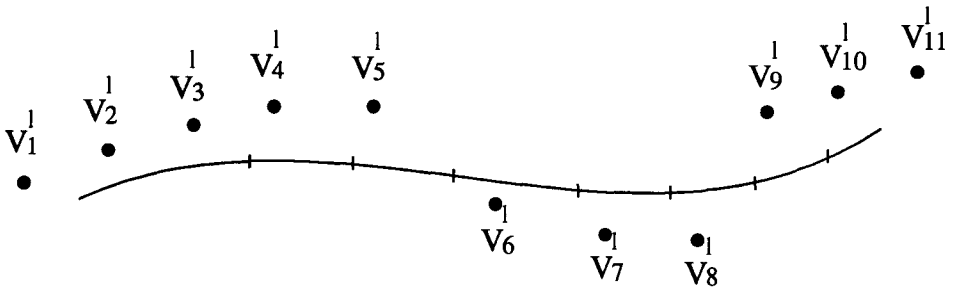
5.1.2 Inverse Subdivision Scheme

This section describes an approach that greatly reduces the number of offsets, but does not produce a hierarchical B-spline surface that is animatable.

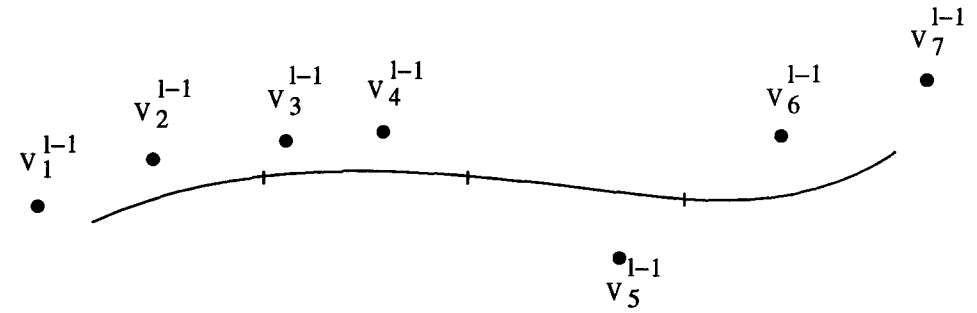
This scheme is designed to minimize the number of non-zero offsets. We begin from the finest level l in FMG. At this level, we have the entire definition of the surface and the finest surface mesh. The next lower level $l-1$ is generated in such a way that if we refine level $l-1$ and generate the finest level by offset reference, we have s minimum number [2] of non-zero offsets. This idea can be described using a spline curve. Suppose we have a cubic B-spline curve defined by 11 control vertices (Figure 5.1a) containing 8 curve segments, and we let this curve be at level l in the hierarchy. Imagine a spline curve obtained by subdividing a 4-segment B-spline curve (Figure 5.1b) with the same order, and adjusting the control vertices by offset referencing.

This 4-segment curve is at level $l-1$. Calculating the 7 control vertices defining the 4-

[2] This minimization is not proven in theory, it is a minimum number easily obtained.



a) A curve defined by 11 control vertices and having 8 segments.



b) A curve obtained by inversely subdividing the curve in a).

Plate 5.1: The inverse subdivision process generate a lower level curve from a upper level curve in a way that reduces the number of non-zero offsets while the hierarchy is built using subdivision.

segment curve is an overdetermined problem. The equations look like this:

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} v_1^{l-1} \\ v_2^{l-1} \\ v_3^{l-1} \\ v_4^{l-1} \\ v_5^{l-1} \\ v_6^{l-1} \\ v_7^{l-1} \end{pmatrix} = \begin{pmatrix} v_1^l \\ v_2^l \\ v_3^l \\ v_4^l \\ v_5^l \\ v_6^l \\ v_7^l \\ v_8^l \\ v_9^l \\ v_{10}^l \\ v_{11}^l \end{pmatrix}. \tag{5.1}$$

If a unique solution is desired, we can reduce the matrix to the following 7×7 matrix:

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} v_1^{l-1} \\ v_2^{l-1} \\ v_3^{l-1} \\ v_4^{l-1} \\ v_5^{l-1} \\ v_6^{l-1} \\ v_7^{l-1} \end{pmatrix} = \begin{pmatrix} v_1^l \\ v_2^l \\ v_4^l \\ v_6^l \\ v_8^l \\ v_{10}^l \\ v_{11}^l \end{pmatrix}. \tag{5.2}$$

This matrix is well-conditioned and thus we know that the solution is unique. The resulting seven control vertices define a curve at level l with the property that when we refine this curve, seven out of the resulting eleven control vertices are identical to the corresponding ones at level l, namely $v_1^l, v_2^l, v_4^l, v_6^l, v_8^l, v_{10}^l, v_{11}^l$. We term this technique *inverse subdivision.* So if we use this idea to build the hierarchy in our bottom-up approach and generate each lower level using inverse subdivision, more than half of the offsets at each level will be guaranteed to be zero. This is a big bonus towards an economical storage cost for our final hierarchical surface.

The bottom-up hierarchy building process is presented in the following algorithm:

Algorithm 5.1 *Inverse Subdivision*

```
for level = highestLevel to lowestLevel+1 do
    InverseSubdivide(level);
```

```
end.

SaveBaseLevel(lowestLevel);

for level = lowestLevel to finestLevel-1 do
     SubdivideLevel(level);
     foreach CV at level+1
       if(Offset(CV) != 0)
          SaveNonzeroOffset(Offset(CV));
     end.
end.
```

5.1.3 Remarks: Solution to Triangular Linear Equation

A simple symbolic manipulation proves that the inverse of the subdivision matrix is filled everywhere and all the entries change when the dimension changes.

Exploiting the banded structure of the matrix, we can reduce it to a unit upper triangular form having an upper triangle consisting of all zeros excepts for the superdiagonal. This can be done by elementary row operations. That is, the system is transformed to:

$$\begin{pmatrix} 1 & \beta_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \beta_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \beta_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \beta_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \beta_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & \beta_6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1^{l-1} \\ v_2^{l-1} \\ v_3^{l-1} \\ v_4^{l-1} \\ v_5^{l-1} \\ v_6^{l-1} \\ v_7^{l-1} \end{pmatrix} = \begin{pmatrix} v'_1 \\ v'_2 \\ v'_4 \\ v'_6 \\ v'_8 \\ v'_{10} \\ v'_{11} \end{pmatrix}. \tag{5.3}$$

where β_i is defined by the recurrence relation:

$$\begin{aligned} \beta_1 &= 1 \\ \beta_i &= \tfrac{1}{4}(6 - \beta_{i-1}) \quad i = 2, \ldots, 7. \end{aligned} \tag{5.4}$$

The v_i's can be calculated as follows:

$$\begin{aligned} v_1' &= 2v_1^{l-1} \\ v_i' &= \frac{4v_i - v_{i-1}'}{6-\beta_{i-1}} \quad i = 2, \ldots, 7. \end{aligned} \tag{5.5}$$

Now that the transformation of the system of equations is complete, the equations can be solved by back substitution from the bottom up by performing the following computations:

$$\begin{aligned} v_7^{l-1} &= v_7' \\ v_i^{l-1} &= v_i' - \beta_i * v_{i+1} \quad i = 6, \ldots, 1. \end{aligned} \tag{5.6}$$

5.1.4 Problems with Inverse Subdivision

Because of the nature of inverse subdivision, the offsets generated by Algorithm 5.1 are not well behaved. The magnitude of offsets become large and the base level control vertices do not approximate the original surface at all. It turns out that with each inverse subdivision, the lower level we get, the "wilder" the surface shape becomes. At the base level, the control vertices go off everywhere and the surface is not a reasonable approximation of the finer levels.

This instability motivates us to find another way to build a hierarchical surface which behaves well at each level.

5.2 Use of MG Data

Remember that the multi-resolution data is available from the FMG solver. These multi-level control vertices define a reasonable shape at each resolution level. Instead of just making use of the finest level result to build the hierarchy, we take advantage of the availability of all levels of data to build the overlays.

We begin from the lowest level, a flat single-patch surface (Plate A.4). The 16 control vertices $VBAK_{ij}^0$ (Algorithm 2.3) , which are solutions for the coarsest level in FMG, define the base level of our hierarchical B-spline having a parametric spacing of one. Then the base level is subdivided and level 1 control vertices $VBAK_{ij}^1$ are generated. Since we use uniform knot spacing, the parametric spacing for subsequent levels is halved recursively. The FMG data

V_{ij}^1 at level 1 with compatible dimension is used as the final surface definition at level 1. Thus offsets at level l are calculated by

$$O_{ij}^l = VBAK_{ij}^l - V_{ij}^l \tag{5.7}$$

where we use V_{ij}^l to indicate the control vertices generated by refining $VBAK_{ij}^{l-1}$ at level $l-1$. The algorithm is given below:

Algorithm 5.2 Hierarchy Building

```
AssignBaseLevel(0);

for level = 0 to finestLevel do
    SubdivideLevel(level);
    for i = 0 to m
        for j = 0 to n
            offset = VBAK[level+1][i][j] - V[level][i][j];
            if (offset !=0)
               SaveNonZeroOffset(offset);
        end.
    end.
end.
```

where m and n are used to indicate the dimensions of control vertices in the u and v directions respectively at each level. The values vary with the level index number.

Because each level is a filtered version of the next higher resolution level, the vertices are well behaved. Algorithm 5.2 is not intended to reduce the non-zero offsets, as Algorithm 5.1 was. Adjacent levels in FMG are just interpolations of points at different resolutions. Nothing in the formulation determines the magnitude of the offset. It turns out that almost all of the offsets generated with algorithm 5.2 are non-zero. If we store this as a hierarchical B-spline

surface, the cost is approximately the same as the cost of storing all of the levels of the FMG solution. The following section shows how to reduce this cost.

5.3 Zeroing Non-zero Offsets

The FMG approach works well because the solution at one level is a good initial guess for the solution at the next finer level. When converted into a hierarchical B-spline surface, most of the offsets are non-zero but their magnitude is small (Figure 5.2). The challenge is to reduce the amount of information required without significantly altering the shape of the surface.

The first approach is to zero any offsets whose magnitude is less than a given tolerance beginning at the coarsest resolution. Since this operation changes the reference control vertices R_{ij} of the next level, the offsets at that level are recalculated to restore the approximation at that level.

```
100:A  :      0.0946581752      0.0206951908     -0.7312056357
101:A  :      0.1082406374      0.0363397667     -0.1349227312
102:A  :     -0.0510964493     -0.0067193945      0.2410037345
103:A  :     -0.4924907012     -0.0094809132      1.4855188643
104:A  :     -0.0812187087      0.0016742268      0.1157502732
105:A  :      0.2453602217      0.0028018600     -0.5366043224
106:A  :      1.0591058826     -0.0002960616     -2.5841572814
107:A  :      0.2264022796     -0.0016349198     -0.5210445247
108:A  :     -0.1049558748     -0.0005351651      0.3200479296
109:A  :     -1.0981894014      0.0036999245      2.1388922244
110:A  :      0.2164414669      0.0024895610     -0.4276355013
```

Plate 5.2: A portion of hierarchical B-pline data file without zeroing non-zero offsets.

Here is the compact algorithm for zeroing out the offsets within tolerance:

Algorithm 5.3 `Offset Zeroing`

```
AssignBaseLevel(0);
```

```
for level = 0 to finestLevel do
    SubdivideLevel(level);
    for i = 0 to m
        for j = 0 to n
            offset = VBAK[level+1][i][j] - V[level][i][j];
            if (abs(offset) > tolerance * level)
               SaveNonZeroOffset(offset);
            else
               VBAK[level+1][i][j] = V[level][i][j];
        end.
    end.
end.
```

Figure 5.3 shows the relationship between the number of non-zero offsets remaining after the zeroing process and the tolerance used. If a relatively large tolerance is used, a large number of zero offsets are created, but the resulting surface is a much poorer approximation of the data.

With the same offset zeroing technique, the residual per data point (the total residual caused by offset zeroing at the finest level divided by the number of data points interpolated.) is plotted in Figure 5.4. And the maximum residual is plotted in Figure 5.5.

5.4 Smoothing Algorithm

Plate A.9 shows the surface with a tolerance of 0.5 out of 200. Because offset are blindly truncated, the difference between offsets which are just above and below the tolerance is exaggerated (Figure 5.6).

Rather than truncating certain offsets, the magnitude of all the offsets are reduced by the same amount the surface recedes back to its reference shape. A reasonable amount is that of the offset tolerance, however it can also be specified explicitly by user to meet concrete applications

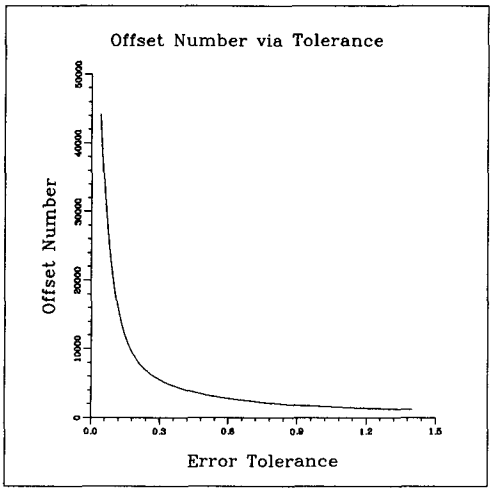


Plate 5.3: Offset number generated in hierarchical surface with given tolerance value. Raw data ranges from -100 to +100.

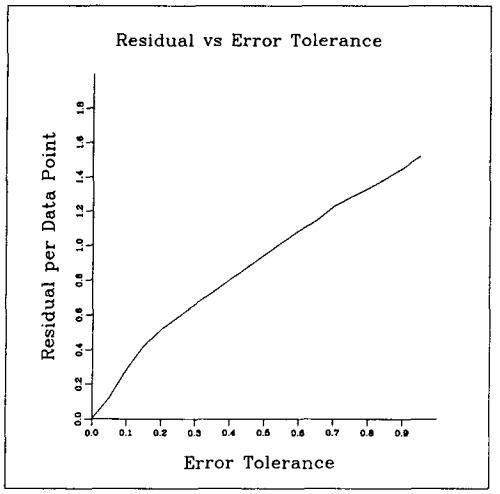


Plate 5.4: Residual per data point caused by offset zeroing with given tolerance value. Raw data ranges from -100 to +100.

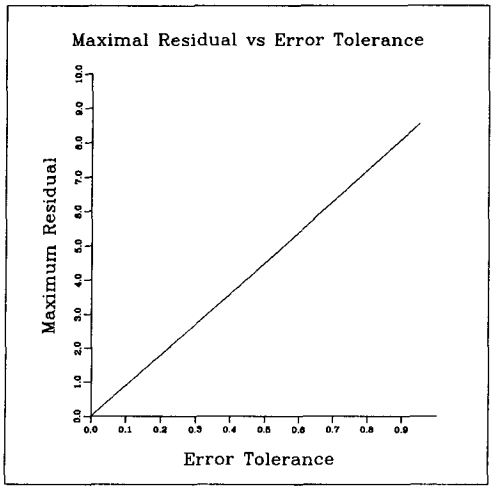


Plate 5.5: Maximum Residual caused by offset zeroing with given tolerance value. Raw data ranges from -100 to +100.

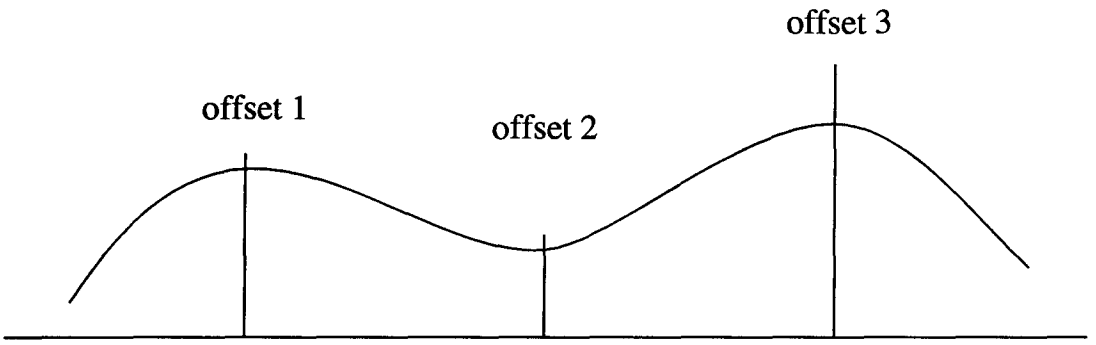
(Figure 5.7).

Here is the complete algorithm incorporating smoothing techniques with the offset zeroing algorithm.

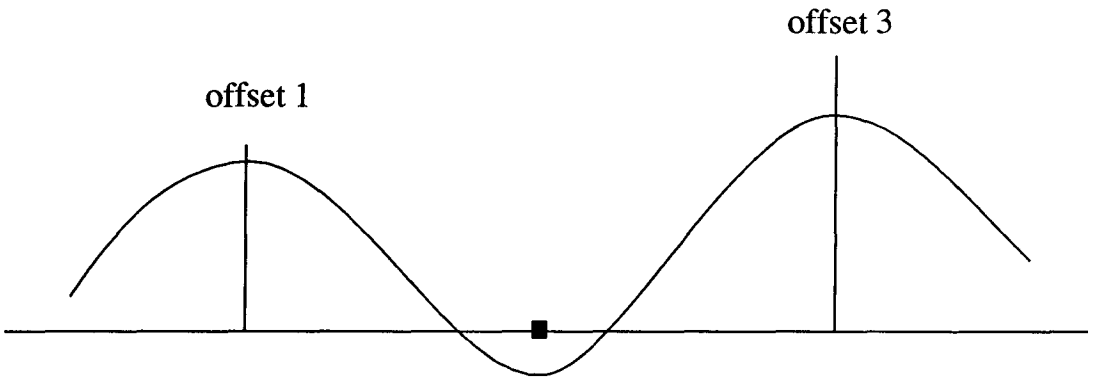
Algorithm 5.4 *Offset Zeroing with Smoothing Function*

```
AssignBaseLevel(0);

for level = 0 to finestLevel do
    SubdivideLevel(level);
    for i = 0 to m
        for j = 0 to n
            offset = VBAK[level+1][i][j] - V[level][i][j];
            if (abs(offset) > tolerance * level){
               offset -= offsetTolerance;
               SaveNonZeroOffset(offset);
```

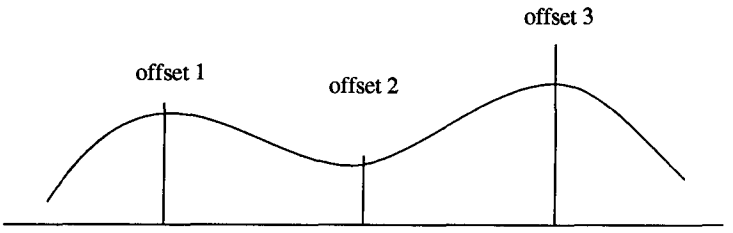


a) Before zeroing offset 2.

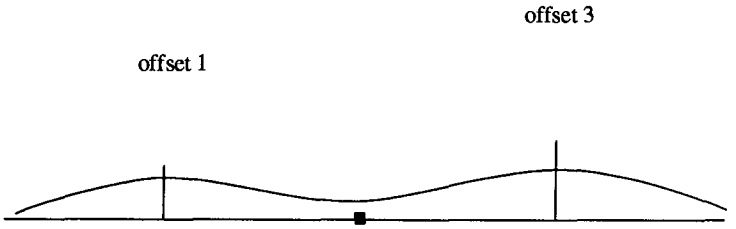


b) After zeroing offset 2.

Plate 5.6: Offset zeroing side effect. a) Before the offset 2 is zeroed. b) The zeroing of offset 2 results in a bump.



a) Before zeroing offset 2.



b) After zeroing offset 2 with smoothing function.

Plate 5.7: Offset zeroing with smoothing function. a) Before the offset 2 is zeroed. b) After offset 2 is zeroed with smoothing function.

```
                }
            else
                VBAK[level+1][i][j] = V[level][i][j];
        end.
    end.
end.
```

5.5 Manipulation of Resulting Surface

By interactive adjustment of the tolerance and smoothing factor, a compact hierarchical representation of the original surface is produced. The resulting surface can be used a input to the Dragon editor for further manipulation and animation. Within the editor, the absolute offsets generated by the approximation code are converted into the tangent offsets that allow fine surface details to be retained as broad scale manipulation are made [17]. Plates A.20 – A.23 show some of the modified shapes.

5.6 Implementation and Results

This method was tested on the data taken from a bust of Victor Hugo (courtesy F. Schmidt) with 264 rows of 361 data points (95304 points. See Figure 1.1 and Plate A.1) with normal magnitude around 200. The value of each data point originally represented the radius from a central axis. As an approximation this information was converted to points in R^3.

The scanned data was sampled in parametric space with a 257×257 grid (66049 data points) and approximated using the FMG method described above. A 9-level hierarchical surface interpolating all the sampled points contains 90494 offsets. With a tolerance of 0.1, this number is reduced to 23270 offsets, about 25% of the sample data set. Smoothing would further reduce storage, but the amount of reduction is highly dependent upon how much smoothing is required, which differs from situation to situation.

Another example is a data set of a digitized face. With 66049 data points, we approximate it with 256×256 patches at the finest level. With a tolerance of 0.05, the offset number in its hierarchical representation is 19235 (Plate A.14).

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, the digitized Victor Hugo's bust has been used as a testbed of the surface approximation techniques designed to solve the problem stated above. For this specific case, the data parameterization challenge is met by a parametric space deformation scheme. This scheme achieves a near chord-length parameterization of the raw data points.

For a detailed representation of the original digitized object, a large number of linear equations have to be solved and a fast algorithm used to solve these equations is desired. Full multigrid methods (FMG) was chosen to carry out this task because of its optimal performance. The FMG solver offers us interactive control of the fitting process and a multi-resolution surface.

To reduce the storage cost (90K points) and to produce a representation suitable for animation, a hierarchical B-spline surface was used to represent the fitted surface. The surface hierarchy was built using two different techniques. The inverse subdivision offered a good compression of the surface storage in terms of non-zero offset number, but the resulting surface was not suitable for multi-resolution editing. The direct use of multi-resolution data from the FMG fitter produced a well-behaved shape using slightly more storage. To further compact the resulting surface, offset zeroing techniques were employed to remove those non-zero offsets within a certain length, and smoothing was used where offset-zeroing produces unsuitable bumps.

The approximation to Victor Hugo's bust was controlled through the selection of the offset

zeroing threshold and smoothing factor. The hierarchical B-spline approximation is suitable for manipulation with the Dragon editor.

Complicated shapes like a human face can be fitted to its very details at a reasonable cost. This is a direct way to achieve realistic models without consuming much interactive manpower. The results are encouraging in that suitable results are possible at a reasonable cost.

6.2 Future Work

Like most of the research topics, this thesis generated more problems than it actually solved.

The initial parameterization scheme results in some sharp turning angles between adjacent data points in parametric space. Subsequently, these angles resulted in notches in the surface under chord-length parameterization. Better initial parameterization is required and a parametric space deforming scheme such as the one presented by Waters and Terzopoulos [39] in improving the final surface fit. The chosen parameterization scheme, although adequate for this particular application, is not general enough. Other methods, such as used in [10], will be investigated with the goal of using scanned data as a mold that can be applied to an existing surface to give it shape.

Future work will also look into better restriction methods for FMG and at the possibility of optimizing the number of zero-offsets in the surface definition.

Bibliography

[1] Ahlberg, J., Nilson, E. and Walsh, J., *The Theory of Splines and Their Applications*, Academic Press, New York, 1967.

[2] Barsky, B. "End Conditions and Boundary Conditions for Uniform B-spline Curve and Surface Representations", *Computers in Industry*, 3(1&2):17-29, 1982.

[3] Bartels, R., Beatty, J. and Barsky, B. *An Introduction to Splines for Use in Computer Graphics and Geometric Modelling*, Morgan Kaufmann Publishers Inc., Palo Alto, California, 1987.

[4] Barsky, B. and Greenberg, D. "Determining a Set of B-spline Control Vertices to Generate an Interpolating Surface", *Computer Graphics and Image Processing*, V.14:203-209, 1979.

[5] Barsky, B., *A study of the Parametric Uniform B-spline Curve and Surface Representation*, Univ. of California at Berkeley Technical Report UCB-CSD-83-118, May 1983.

[6] Birkhoff, C. and MacLane, S., *A Survey of Modern Algebra*, The Macmillan Company, New York, 1965.

[7] Brandt, A., "Multi-level Adaptive Solutions to Boundary Value Problems", *Mathematics of Computation*, V.31(138):333-390, 1977.

[8] Briggs, W. *A Multigrid Tutorial*, SIAM, 1987.

[9] Cohen, E. and Lyche, T., and Reidenfeld, R., "Discrete B-splines and Subdivision Techniques in Computer Aided Geometric Design and Computer Graphics", *Computer Graphics and Image Processing*, V.14(2):87-111.

[10] Cohen, E., O'Dell, C., "A Data Dependent Parameterization for Spline Approximation", *Mathematical Methods in Computer Aided Geometric Design*:155-166, edited by T. Lyche and L. Schumaker, Academic Press, Boston, 1989.

[11] Coons, S., "Surface Patches and B-spline Approximation", *Computer Aided Geometric Design*:1-16, edited by Barnhill, R. and Riesenfeld, R., Academic Press, New York, 1974.

[12] de Boor, C., *A Practical Guide to Splines*, Spring-Verlag, New York, 1978.

[13] Epstein, M., "On the Influence of Parameterization in Parametric Interpolation", *SIAM Journal of Numerical Analysis*, 13(2):261-268, 1976.

[14] Farin, G., *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press Inc., 1988.

[15] Foley, T., "Interpolation with Interval and Point Tension Controls Using Cubic Weighted ν-Splines", *ACM Transactions on Mathematical Software*, V.13(1):68-96, 1987.

[16] Foley, T. and Nielson G., "Knot Selection for Parametric Spline Interpolation", *Mathematical Methods in Computer Aided Geometric Design*:261-271, edited by T. Lyche and L. Schumaker, Academic Press, Boston, 1989.

[17] Forsey, D. *Motion Control and Surface Modelling of Articulated Figures in Computer Animation*, Univ. of Waterloo Technical Report CS-90-28, (Ph.D. Thesis) 1990.

[18] Forsey, D. and Bartels, R. "Tensor Products and Hierarchical Fitting", *Proceedings IEEE Curves and Surfaces in Computer Vision and Graphics II*, Boston, Nov. 1991.

[19] Forsey, D. and Bartels, R. "Surface Fitting with Hierarchical Splines" to appear in Transactions on Graphics.

[20] Forsey, D., and Bartels, R. H., "Hierarchical B-spline Refinement", *Proceedings of SIGGRAPH '88*:205-212, Atlanta, Georgia, 1988.

[21] Hackbusch, W., *Multigrid Methods and Applications*, Springer-Verlag, 1985.

[22] Hoschek, J., "Intrinsic Parameterization for Approximation", *Computer Aided Geometric Design*, V.5:27-31, 1988.

[23] Kass, M., Witkin, A. and Terzopolous, D., "Snakes: Active Contour Models", *International Journal of Computer Vision*, V.4:321-331, 1988.

[24] Lee, E., "On Choosing Nodes in Parametric curve Interpolation", *SIAM Conference on Applied Geometry*, Albany, New York, 1985.

[25] Lin, C., Chang, P. and Luh, J., "Formulation and Optimization of Cubic Polynomial Joint Trajectories for Mechanical Manipulation", *Proceedings of 21st IEEE Conference on Decision and Control*, V.1:330-335, Orlando, Florida, December, 1982.

[26] Lyche, T. and Morken, V., "Knot removal for parametric B-spline curves and surfaces", *Computer Aided Geometric Design*, V.4(3):217-230, 1987.

[27] Marin, S., "An Approach to Data Parameterization in Parametric Cubic Spline Interpolation Problems", *Journal of Approximation Theory*, V.41:64-86, 1984.

[28] McCormick, S., "Multilevel Adaptive Methods for Partial Differential Equations", SIAM, 1989.

[29] Mullineux, G., "Approximating Shapes Using Parameterized Curves", *IMA Journal of Applied Mathematics*, V.29(2):203-220, 1982.

[30] Nahas, M., Huitric, H. and Saintourens, M., "Animation of a B-Spline Figure", *The Visual Computer*, V.3(4), 1987.

[31] Nielson, G., "Coordinate Free Scattered Data Interpolation", *Topics in Multivariate Approximation*:175-184, edited by C. Chui, l. Schumaker and F. Utreras, Academic Press, New York, 1987.

[32] Nielson, G. and Foley, T., "A Survey of Applications of an Affine Invariant Norm", *Mathematical Methods in Computer Aided Geometric Design*:445-467, edited by T. Lyche and L. Schumaker, Academic Press, Boston, 1989.

[33] Schmitt, F., Barsky, B. and Du, W. "An Adaptive Subdivision Method for Surface-Fitting from Sampled Data", *Proceedings of SIGGRAPH '86*:179-188, Dallas, Texas, 1986.

[34] Schmitt, F., Maitre, H., Clainchard, A. and Lopez-Krahm, J., "Acquisition and Representation of Real Object Surface Data", *SPIE Proceedings Vol. 602 of Biostereometrics '85 Conference*, Cannes, France, 2-6 December, 1985.

[35] Screckovic, M., *Adaptive Hierarchical Fitting of Curves and Surfaces*, Master's Thesis, University of Waterloo, 1991.

[36] Sederberg, T. and Parry, S. "Free-Form Deformation of Solid Geometric Models", *Proceedings of SIGGRAPH '86*:151-160, Dallas, Texas, 1986.

[37] Terzopoulos, D., "Image Analysis Using Multigrid Relaxation Methods", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, V.8(2):129-139, 1986.

[38] Terzopoulos, D.,"Multi-level Surface Reconstruction", *Computer Vision, Graphics and Image Processing*, V.24:52-96, 1983.

[39] Waters, K. and Terzopoulos, D., "Modelling and Animating Faces Using Scanned Data", *The Journal of Visualization and Computer Animation*, V2:123-128, 1991.

[40] Welch, W. and Witkin, A. "Variational Surface Modelling", *Proceedings of SIGGRAPH '92*:157-166, Chicago, Ill, 1992.

[41] Wolberg, G., "Digital Image Warping", IEEE Computer Society Press, 1990.

Appendix A

Plates

This appendix includes the pictures generated by the implementation as results.

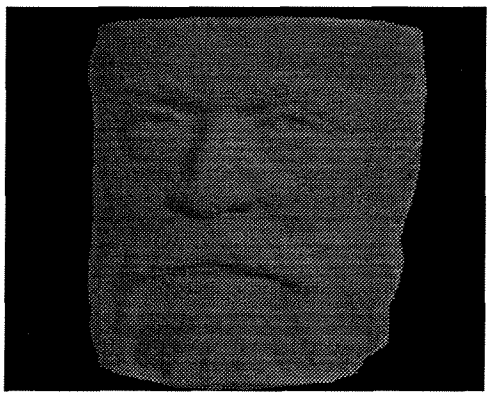


Plate A.1: Victor Hugo raw data.

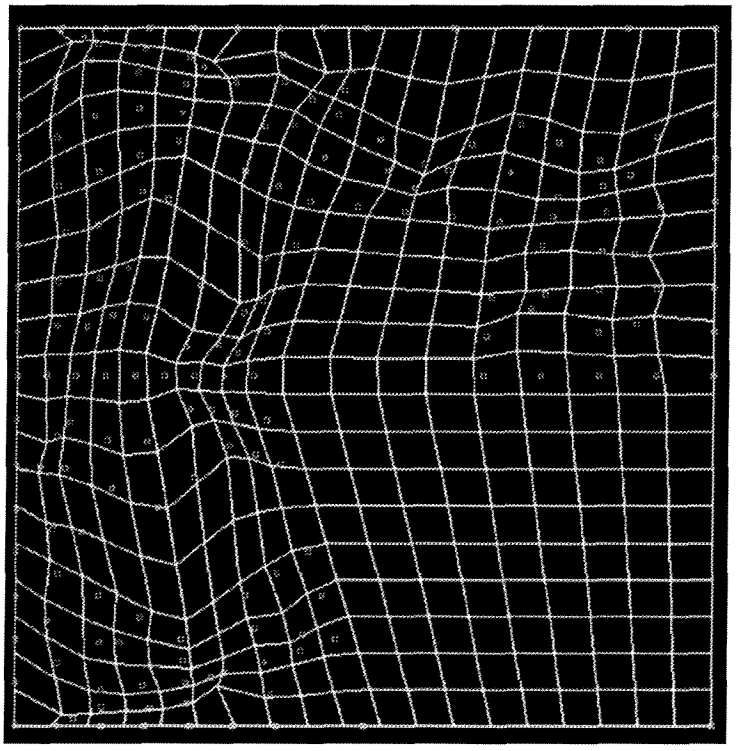


Plate A.2: Deformed UV isoparametric lines in ST space with 5 level refinement.

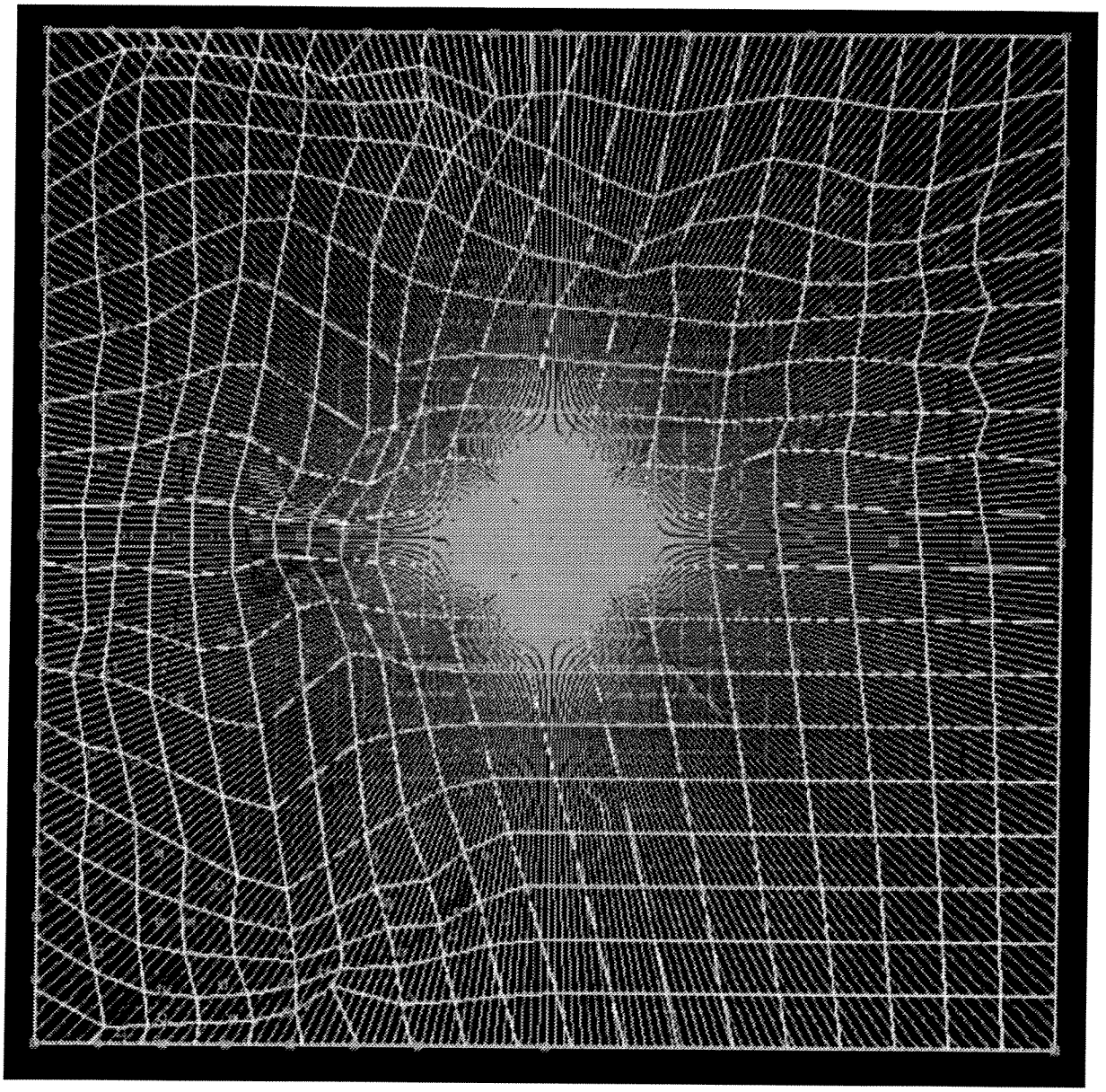


Plate A.3: Deformed UV isoparametric lines in ST space with 5 level refinement. Data points are plotted over the space.

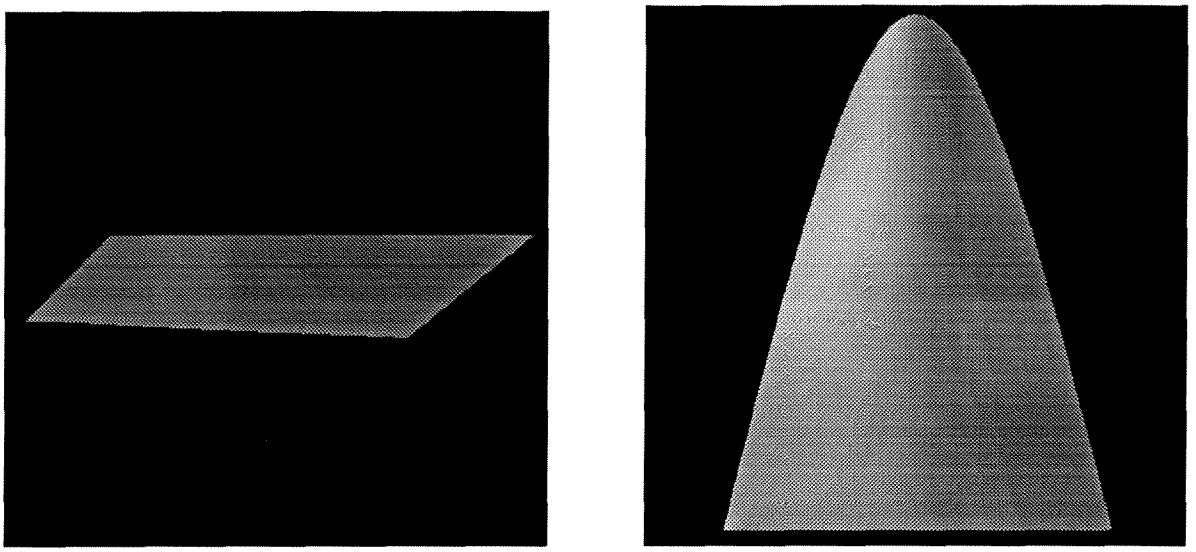


Plate A.4: FMG fit at level 1 and level 2.

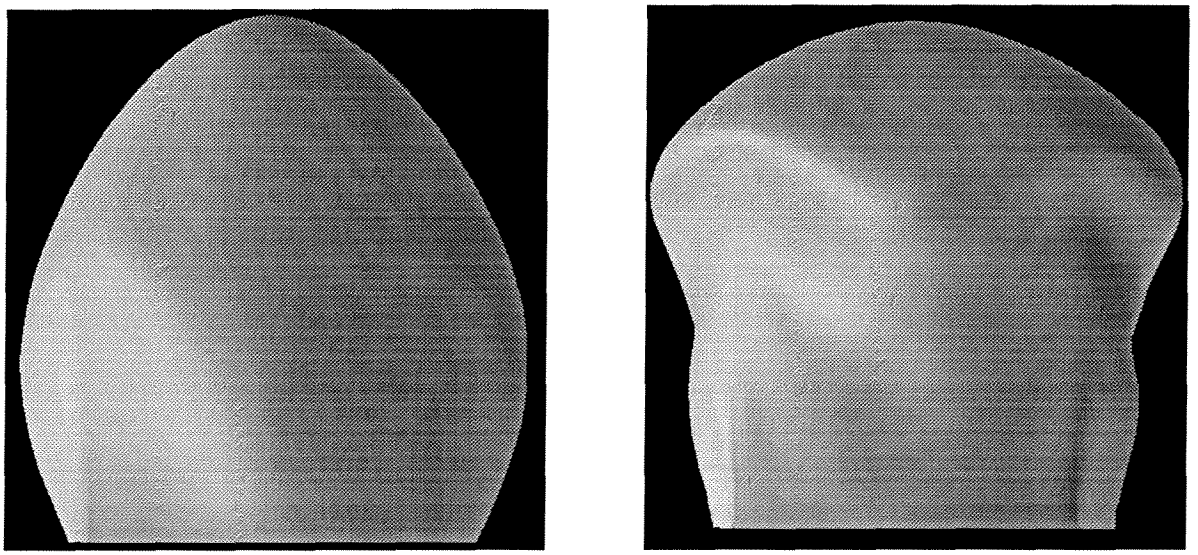


Plate A.5: FMG fit at level 3 and level 4.

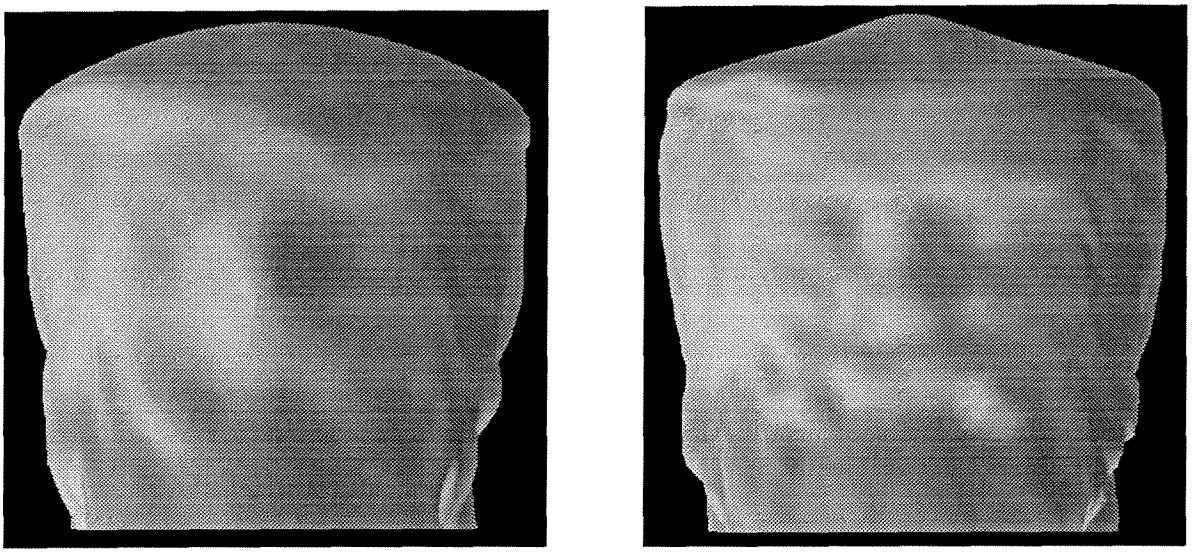


Plate A.6: FMG fit at level 5 and level 6.

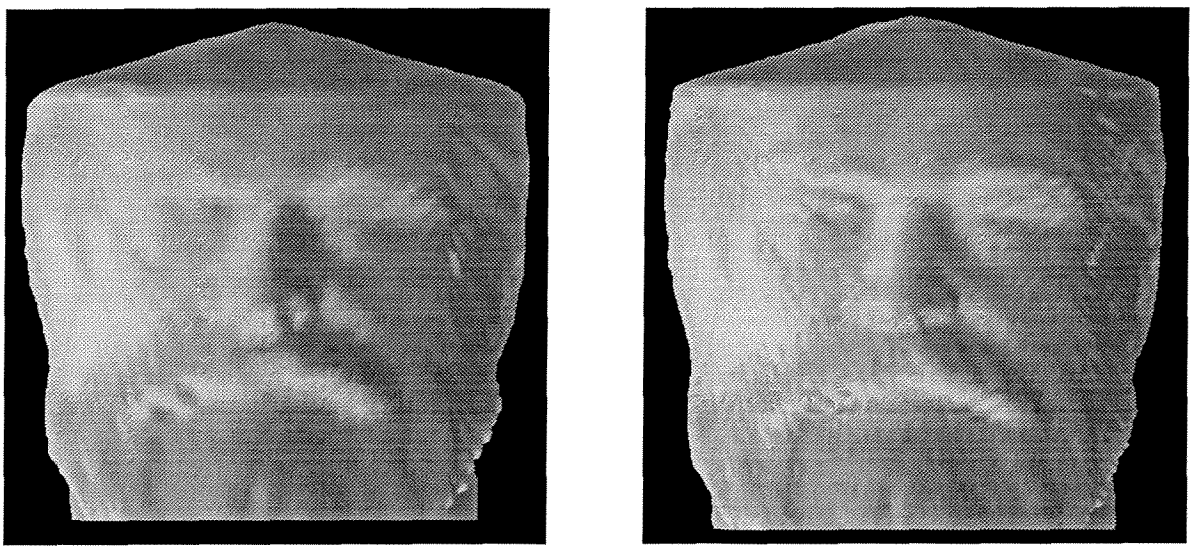


Plate A.7: FMG fit at level 7 and level 8.

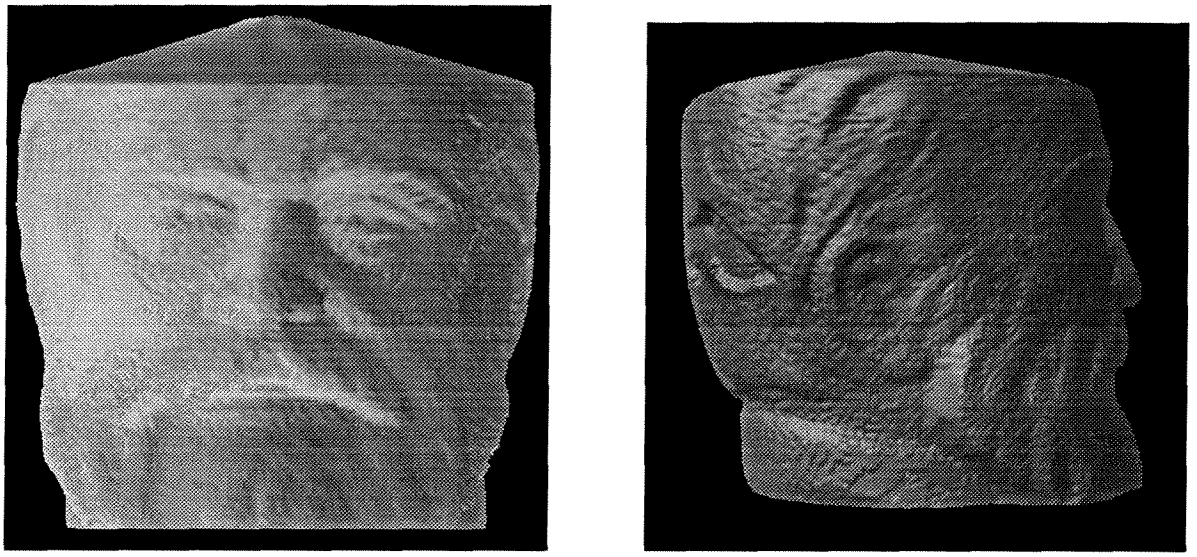


Plate A.8: Left: FMG fit at level 9. Right: Hierarchical B-spline surface without zeroing offsets. Offset number is 90949.

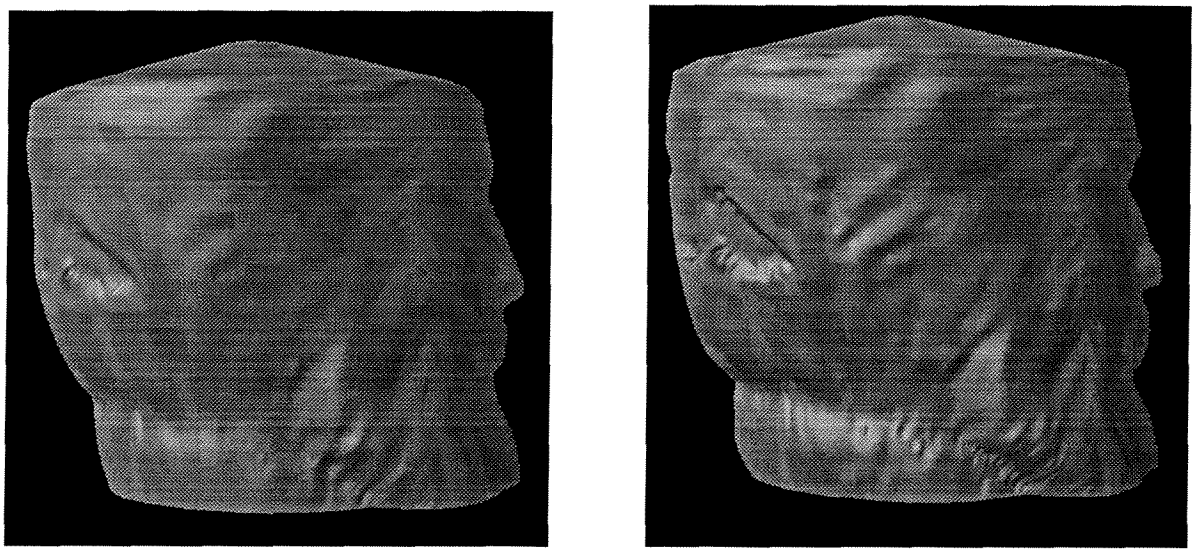


Plate A.9: Left: Hierarchical B-spline surface with offset zeroing threshold 0.5 and smoothing algorithm. Offset Number is 3913. Right: Hierarchical B-spline surface with offset zeroing threshold 0.5 without smoothing algorithm.

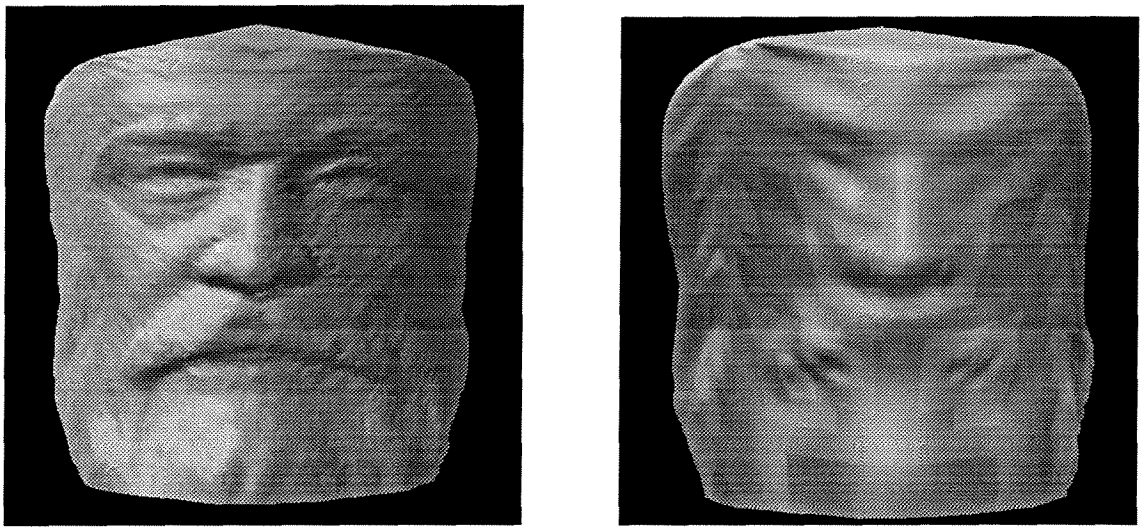


Plate A.10: Left: Hierarchical B-spline surface with offset zeroing threshold 0.1 and smoothing algorithm. Offset Number is 23270. Right: Hierarchical B-spline surface with offset zeroing threshold 1.0 and smoothing algorithm. Offset Number is 1918.

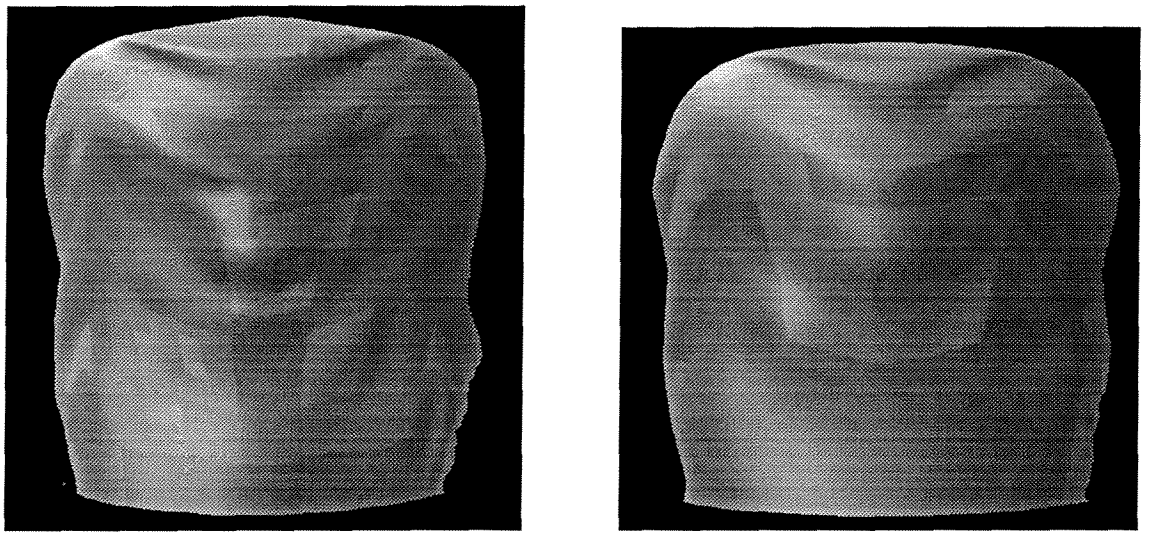


Plate A.11: Left: Hierarchical B-spline surface with offset zeroing threshold 2.0 and smoothing algorithm. Offset Number is 930. Right: Hierarchical B-spline surface with offset zeroing threshold 5.0 and smoothing algorithm. Offset Number is 318.

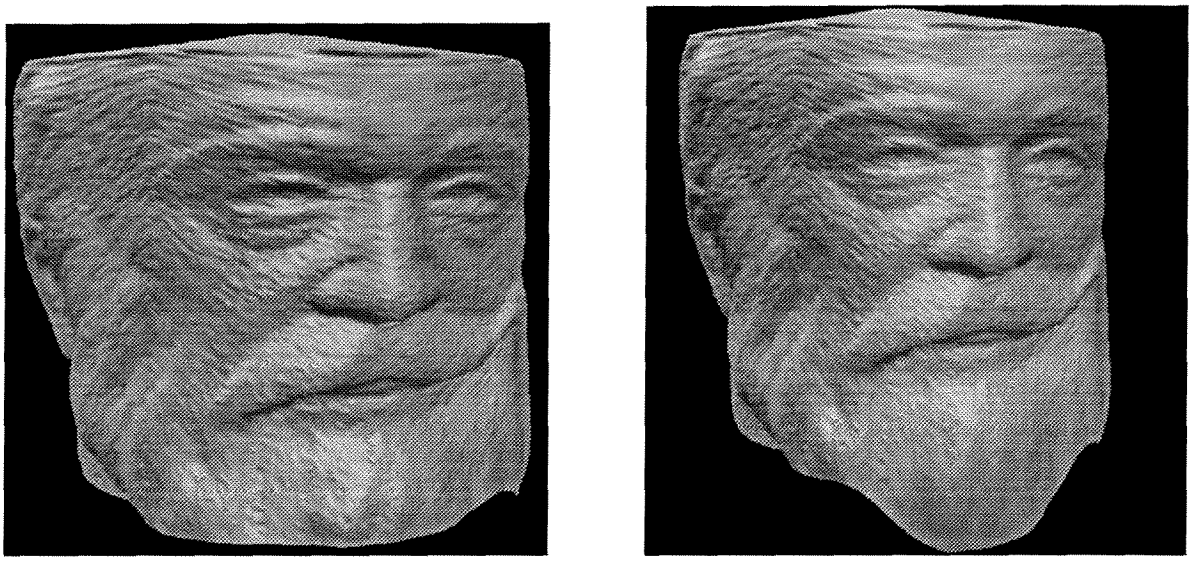


Plate A.12: Left: Surface manipulation – Smile 1. Right: Surface manipulation – Smile 2.

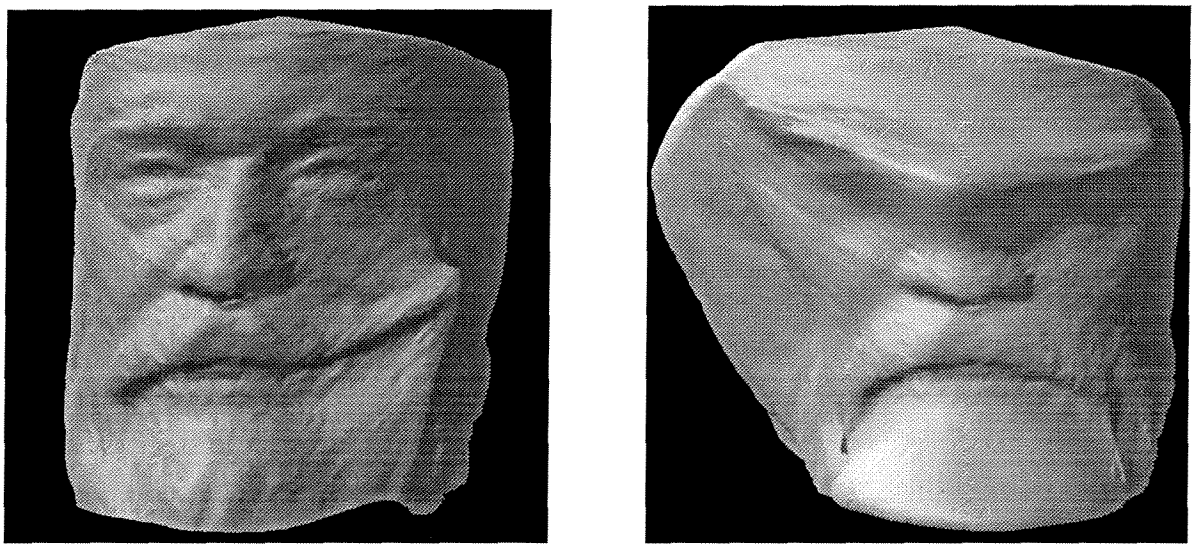


Plate A.13: Left: Surface manipulation – Smile 3. Right: Surface manipulation – Neanderthal man.

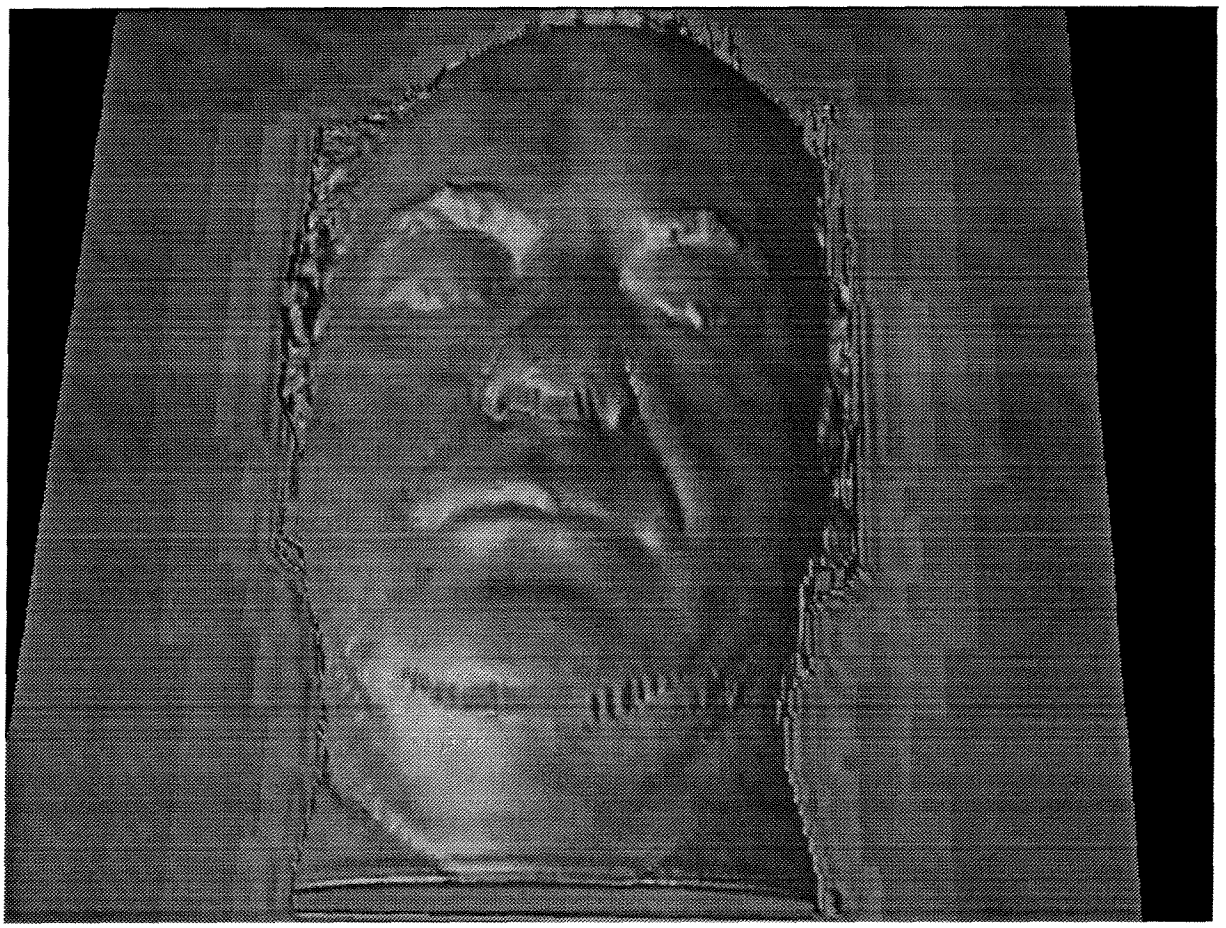


Plate A.14: Another fit to a digitized face.