**Skinning Engineering Models with Non-Uniform, Hierarchical B-Spline Surfaces**

by

David H. Coe

Thesis Submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

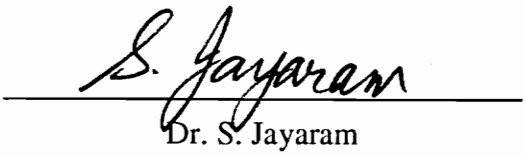
Master of Science

in

Mechanical Engineering

APPROVED:

Dr. A. Myklebust, Chairman



Dr. S. Jayaram

Dr. J. R. Mahan

July 16, 1993

Blacksburg, Virginia

**Skinning Engineering Models with Non-Uniform, Hierarchical B-Spline Surfaces**

by

David H. Coe

Dr. A. Myklebust, Chairman

Mechanical Engineering

(ABSTRACT)

This thesis presents the algorithms and methods to represent the skin of an engineering model with non-uniform, hierarchical B-spline surfaces. Non-uniform, hierarchical B-splines offer the mechanical designer many advantages: parametrically defined components may be added to a surface while maintaining $C^2$ surface continuity; detailed features may be added to a surface without globally affecting the B-spline control net; and an appropriate geometric basis for finite element meshing and analysis in the conceptual design phase can be established. These algorithms are applied to ASCYNT, a conceptual aircraft design code, to verify and validate the algorithms. A single-surface definition of an aircraft skin, appropriate for computational fluid dynamic and radar and infrared cross-section analysis, is designed using non-uniform hierarchical B-spline surfaces.

# Acknowledgments

I would like to thank my advisor, Dr. A. Myklebust, for his generous support and advice during this research. I would also like to express my appreciation to Dr. Jayaram and Dr. Mahan for serving on my committee. Special thanks to all of those in the Virginia Tech CADLAB.

Thanks to Dr. David Forsey, Dr. Wenhsyong Lin, and Dr. Greg Moore. Your help and support are greatly appreciated.

Finally, thanks mom and dad.

# Table of Contents

# List of Illustrations

# 1.0 Introduction

The mechanical design process proceeds through three distinct phases : conceptual design, preliminary design, and final design. Paralleling these design phases is the geometric model of the design. The conceptual design phase is characterized by the need to evaluate many system configurations which would meet the customer's performance specifications. These trade-off studies are ideally accompanied by parametrically defined components which give first-order approximations of the size, power, and weight of the system being designed. With parametrically defined components, the initial size, power, and weight budgets are optimized to arrive at a system configuration for preliminary design considerations. The preliminary design phase expands the knowledge about the design with extensive analysis and experimental data. The finite element model is used extensively in the preliminary design phase to validate the design and correlate with experimental data. With a final validated design, the final, detailed design phase begins. During this detail design phase the first prototype is built from engineering drawings and schematics. The geometric model is typically three-dimensional with two-dimensional engineering drawings.

The geometric model - the *object* of the design process - is central to each design phase, yet differs qualitatively in each phase. The qualities of each geometric model admit certain analytical tasks and preclude others. The parametrically defined component of the conceptual design phase is inadequate for the finite element analysis of the preliminary design phase. The finite element model only approximates the surface of a given component and cannot be used in detailed engineering drawings of the final design phase where accurate surfaces and curves are required. Detailed drawings and the rigid point, line, and surface definitions of the final design phase are too rigid for conceptual design. The geometric model proceeds through a metamorphosis where the geometric model serves a particular analytic need of that design phase. The geometric form is at first fluid and dynamic, but becomes more rigid and detailed and each successive design phase. The research in this thesis addresses the need to develop a engineering surface model which can be utilized in the conceptual and preliminary design phases. The specific surface which is developed in this thesis is a non-uniform hierarchical B-spline surface.

## 1.1 *Research Objectives*

This thesis develops the essential elements of non-uniform, hierarchical, B-spline surfaces and applies this surface design technique to conceptual aircraft design. The need for a engineering surface model which can be used in both the conceptual and preliminary design phases lead to the development of a non-uniform, hierarchical, B-spline surface. The engineering surface should have the following properties:

- The surface should be $C^2$ continuous.

- The surface design method should accommodate predefined components.

- The surface design method should be appropriate for the engineering domain.

The major objective of this research is to develop a $C^2$ continuous surface definition of an aircraft skin from parametrically defined components. These algorithms should be compatible with ACSYNT's current B-spline data structures and design methodology. ACSYNT is a conceptual aircraft design code developed jointly by NASA Ames Research Center and the VPI CAD Laboratory.

## 1.2 *Thesis Organization*

This thesis reviews the essential elements for developing non-uniform, hierarchical B-spline surfaces and applying this surface design method to conceptual aircraft design. The literature survey is presented in Chapter Two. Chapter Three contains the definition of B-spline curves and their properties. The B-spline curves are expanded to tensor product B-spline surfaces in Chapter Four. The following chapter presents the research and development of non-uniform, hierarchical B-spline surfaces. Chapter Six reviews the application of these algorithms to ACSYNT. The results and conclusions are presented in Chapter Seven.

# 2.0 Literature Survey

Free-form surface design techniques range from data point interpolation to differential-equation-based surfaces. Boehm [Boehm84], Barnhill [Barn90a],[Barn90b], and Piegl [Pieg89] present excellent surveys of computer aided geometric design of curves and surfaces. Some basic techniques of free-form surface design are reviewed below.

## 2.1 *Surface Interpolation*

The field of point interpolation with tensor product B-spline surfaces is well developed. De Boor [deBo72] presents the classic techniques for interpolating points. The interpolation approach has been expanded and modified by many researchers. Cheng and Barsky [Chen91] combine interpolation with approximation techniques so that some points are interpolated while others are approximated. Kochanek and Bartles [Koch84] developed interpolating splines with local tension, continuity, and bias control. Hohenmyer and Bartels [Hohm91] describe the process of "skinning" to construct an interpolating surface. These techniques do not provide methods for "local" refinement;

the addition of a new interpolating point will generate a new row and column of control points.

## 2.2 *Surface Lofting*

Lofting a surface is the surface design method where cross-sections are defined and then interpolated by a surface. Ball [Ball 74] describes the lofting tile used by the British Aircraft Corporation in the 1970's. A procedural lofting process is developed by Filip [Fili89] which introduces a unique parametric scheme. Unfortunately the procedural lofting method is not readily understood (or adapted) by other CAD systems and can provide only $C^1$ surface continuity. Tiller [Tiller83] summarizes the basic skinning (lofting) technique for rational B-spline surfaces. Woodward [Wood88] provides another example of cross-sectional design of B-spline surfaces. Kankainen[Kank91] employs skeletal lines to generate the lofted surface of a fighter aircraft fuselage. The lofting techniques are well developed, but cannot easily provide a single surface definition for orthogonally lofted components.

## 2.3 *Hierarchical Surfaces*

Free form surface design with hierarchical B-spline surfaces was introduced by Forsey [Fors88]. This method is limited for engineering design by the use of uniform B-splines. The surface design method did introduce a hierarchical data structure to "localize" the B-spline refinement operations; this surface design method has expanded in this thesis to address the engineering design domain by using non-uniform B-splines.

## 2.4 *Conceptual Aircraft Surface Modeling*

The Virginia Tech CAD Laboratory has actively pursued surface modeling techniques for the conceptual design of aircraft. Wampler [Wamp88a], [Wamp88b] developed the core CAD framework for automated design of conceptual aircraft. Wong [Wong90], [Wong91] developed robust surface intersection techniques. Jones [Jones91] and Gloudemans [Glou89],[Glou90] addressed filleting and surface intersection between components. Fleming [Flem91],[Flem92a], [Flem92b] developed geometric modeling tools for B-spline surface creation. Jayaram [Jaya91],[Jaya92b] incorporated new methods to determine component parameters from B-spline surface definitions. Together these works form the body of ACSYNT's geometric modeling capability.

# 3.0 B-Spline Curves

## 3.1 *Background*

B-spline curves were developed in response to the need for precise, mathematically defined curves in the aircraft and automobile industries [Sarr84]. Ferguson [Ferg64] introduced splines at Boeing in the 1960s to replace the conic lofting methods from the early 1940s. Prior to the conic lofting method, curves were determined by studying the deformations of wooden or metal strips caused by pegs and weights on scaled aircraft fuselages. The thin strip would assume a shape that minimized the strain energy. The curve generated was called a spline curve.

The need to precisely represent complex curves in the aircraft and automobile industry continues. The focus of this research is the development of a single, second derivative continuous ($C^2$) surface for an aircraft skin with a non-uniform, hierarchical, tensor product B-spline surface. A review of B-spline basis functions, curves and properties is presented to provide the mathematical basis for this research. Hierarchical B-spline curves are developed to introduce the essential elements needed for hierarchical surfaces.

## 3.2 *B-Spline Basis Functions*

A cubic B-spline curve, **Q**(u), may be defined as the sum of weighted basis functions. Equation (3.1) gives the mathematical definition of these functions. The basis functions, $B_i$, are functions of the parameter $u$ and are weighted by control vertices $\mathbf{V}_i$. It is these basis functions which gives rise to the name Basis-spline curve. Several good references are available concerning the mathematical framework of B-splines [deBo78], [Bart87], [Fari90], [Faux79], [Foley90], [Mort85], [Yama88] and [Zeid91]. This thesis follows Bartels' presentation in developing the cubic $C^2$ basis functions.

$$\mathbf{Q}(u) = \sum_{i}^{i+3} \mathbf{V}_i B_i(u) = \sum_{i}^{i+3} ( x_i B_i(u), y_i B_i(u), z_i B_i(u) ) \qquad (3.1)$$

## 3.3 *Cubic Spline Basis Derivation*

The cubic basis function is composed of four continuous polynomial segments over the parameter range $[u_i, u_{i+4}]$, (see figure 1). If the intervals have the same length, a uniform cubic B-spline basis function is generated. If the derivation is performed with unequal intervals, the basis functions for non-uniform B-splines are produced. Since a $C^2$ basis function is desired, the four segments must have positional, first-derivative, and second-derivative continuity at each end of their parameter intervals; the endpoint parameter values are called knots. The collection of knots is called a knot sequence. For cubic B-spline basis functions, $B_i(u)$ is zero for $[u \leq u_i]$ and $[u \geq u_{i+4}]$.

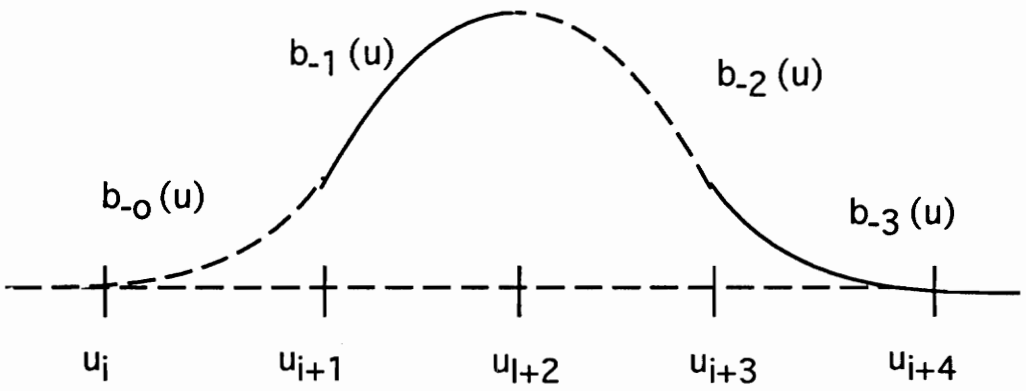



**Figure 1. A Single B-spline Basis Function**

The positional, first-derivative and second-derivative requirements imply:

| | | |
|---|---|---|
| $b_{-0}(0) = 0$ | $b_{-0}^{(1)}(0) = 0$ | $b_{-0}^{(2)}(0) = 0$ |
| $b_{-0}(1) = b_{-1}(0)$ | $b_{-0}^{(1)}(1) = b_{-1}^{(1)}(0)$ | $b_{-0}^{(2)}(1) = b_{-1}^{(2)}(0)$ |
| $b_{-0}(1) = b_{-2}(0)$ | $b_{-0}^{(1)}(1) = b_{-2}^{(1)}(0)$ | $b_{-0}^{(2)}(1) = b_{-2}^{(2)}(0)$ |
| $b_{-0}(1) = b_{-3}(0)$ | $b_{-0}^{(1)}(1) = b_{-3}^{(1)}(0)$ | $b_{-0}^{(2)}(1) = b_{-3}^{(2)}(0)$ |
| $b_{-0}(1) = 0$ | $b_{-0}^{(1)}(1) = 0$ | $b_{-0}^{(2)}(1) = 0$ |

The normalizing condition, equation (3.2), provides sufficient constraints to solve for the coefficients for each cubic polynomial segment. To simplify the following derivation, each interval is parameterized from 0 to 1. The derivation that follows specifies the general cubic equation , its first and second derivatives, and applies end conditions to solve for the coefficients of each basis segment.

(3.2) $b_{-0}(u) + b_{-1}(u) + b_{-2}(u) + b_{-3}(u) = 1$

The first basis segment is defined by:

(3.3) $b_{-0}(u) = a_0 + b_0u + c_0u^2 + d_0u^3$ positional

(3.4) $b_{-0}^{1} = b_0 + 2c_0u + 3d_0u^2$ first derivative

(3.5) $b_{-0}^{2} = 2c_0 + 6d_0u$ second derivative

The end conditions of the first basis segment are:

| Left-end conditions of 1st basis segment | Right-end conditions of 1st basis segment |
|---|---|
| $b_{-0}(0) = 0 \Rightarrow a_0 = 0$ | $b_{-0}(1) = d_0$ |
| $b^1_{-0}(0) = 0 \Rightarrow b_0 = 0$ | $b^1_{-0}(1) = 3d_0$ |
| $b^2_{-0}(0) = 0 \Rightarrow c_0 = 0$ | $b^2_{-1}(1) = 6d_0$ |

The second basis segment is defined by:

(3.6) $b_{-1}(u) = a_1 + b_1u + c_1u^2 + d_1u^3$ positional

(3.7) $b^1_{-1}(u) = b_1 + 2c_1u + 3d_1u^2$ first derivative

(3.8) $b^2_{-1}(u) = 2c_1 + 6d_1u$ second derivative

The end conditions of the second basis segment are:

| Left-end conditions of 2nd basis segment | Right-end conditions of 2nd basis segment |
|---|---|
| $b_{-1}(0) = a_1 = d_0$ | $b_{-1}(1) = d_0 + 3d_0 + 3d_0 + d_1 = 7d_0 + d_1$ |
| $b^1_{-1}(0) = b_1 = 3d_0$ | $b^1_{-1}(1) = 3d_0 + 6d_0 + 3d_1 = 9d_0 + 3d_1$ |
| $b^2_{-1}(0) = 2c_1 = 6d_0$ | $b^2_{-1}(1) = 6d_0 + 6d_1$ |

The third basis segment is defined by:

(3.9) $b_{-2}(u) = a_2 + b_2u + c_2u^2 + d_2u^3$ positional

(3.10) $b^{1}_{-2}(u) = b_2 + 2c_2u + 3d_2u^2$ first derivative

(3.11) $b^{2}_{-2}(u) = 2c_2 + 6d_2u$ second derivative

The end conditions of the third basis segment are:

| Left-end conditions of 3rd basis segment | Right-end conditions of 3rd basis segment |
|---|---|
| $b_{-2}(0) = a_2 = 7d_0 + d_1$ | $b_{-2}(1) = 19d_0 + 7d_0 + d_2$ |
| $b^{1}_{-2}(0) = b_2 = 9d_0 + 3d_1$ | $b^{1}_{-2}(1) = 9d_0 + 6d_0 + 3d_1 + 6d_1 + 3d_2$ |
| | $= 15d_0 + 9d_1 + 3d_2$ |
| $b^{2}_{-2}(0) = 2c_2 = 6d_0 + 6d_1$ | $b^{2}_{-2}(1) = 6d_0 + 6d_1 + 6d_2$ |

The fourth basis segment is defined by:

(3.12) $b_{-3}(u) = a_3 + b_3u + c_3u^2 + d_3u^3$ positional

(3.13) $b^{1}_{-3}(u) = b_3 + 2c_3u + 3d_3u^2$ first derivative

(3.14) $b^{2}_{-3}(u) = 2c_3 + 6d_3u$ second derivative

The end conditions of the fourth basis segment are:

| Left-end conditions of $4^{th}$ basis segment | Right-end conditions of $4^{th}$ basis segment |
|---|---|
| $b_{-3}(0) = a_3 = 19d_0 + 7d_1 + d_2$ | $b_{-3}(1) = 0 = a_3 + b_3 + c_3 + d_3$ |
| $b^1_{-3}(0) = b_3 = 15d_0 + 9d_1 + 3d_2$ | $b^1_{-3}(1) = 0 = b_3 + 2c_3 + 3d_3$ |
| $b^2_{-3}(0) = 2c_3 = 6d_0 + 6d_1 + 3d_2$ | $b^2_{-3}(1) = 0 = 2c_3 + 6d_3$ |

Repeating the normalizing constraint (equation 3.2), expanding each basis segment, and collecting terms of $u$ implies the following four equations:

(3.2) $b_{-0}(u) + b_{-1}(u) + b_{-2}(u) + b_{-3}(u) = 1$ normalizing constraint

(3.15) $a_0 + a_1 + a_2 + a_3 = 1$ $u^0$ terms

(3.16) $b_0 + b_1 + b_2 + b_3 = 0$ $u^1$ terms

(3.17) $c_0 + c_1 + c_2 + c_3 = 0$ $u^2$ terms

(3.18) $d_0 + d_1 + d_2 + d_3 = 0$ $u^3$ terms

Using the above 16 equations (3.3 -3.18), the 16 unknown polynomial coefficients of the four cubic basis segments can be determined. The resulting basis segments are:

(3.19) $b_{-0}(u) = \frac{1}{6}u^3$

(3.20) $b_{-1}(u) = \frac{1}{6}(1 + 3u + 3u^2 - 3u^3)$

(3.21) $b_{-2}(u) = \frac{1}{6}(4 - 6u^2 + 3u^3)$

(3.22) $b_{-3}(u) = \frac{1}{6}(1 - 3u + 3u^2 - u^3)$

## 3.4 *B-spline Curves*

Similar to the cubic, $C^2$ basis function $B_i(u)$, each cubic B-spline curve segment is composed of four weighted basis functions. The basis functions are weighted by control points, $\mathbf{V_i}$. The $i^{th}$ curve segment the B-spline curve is defined by :

$$\mathbf{Q_i}(u) = \sum_{i-3}^{i} \mathbf{V_i} B_i(u)$$

$$\mathbf{Q_i}(u) = \mathbf{V_{i-3}} B_{i-3}(u) + \mathbf{V_{i-2}} B_{i-2}(u) + \mathbf{V_{i-1}} B_{i-1}(u) + \mathbf{V_i} B_i(u)$$

Since the basis functions are non zero only over one interval, $[u_i, u_{i+1}]$, the basis functions may be replaced with the appropriate basis function segments.

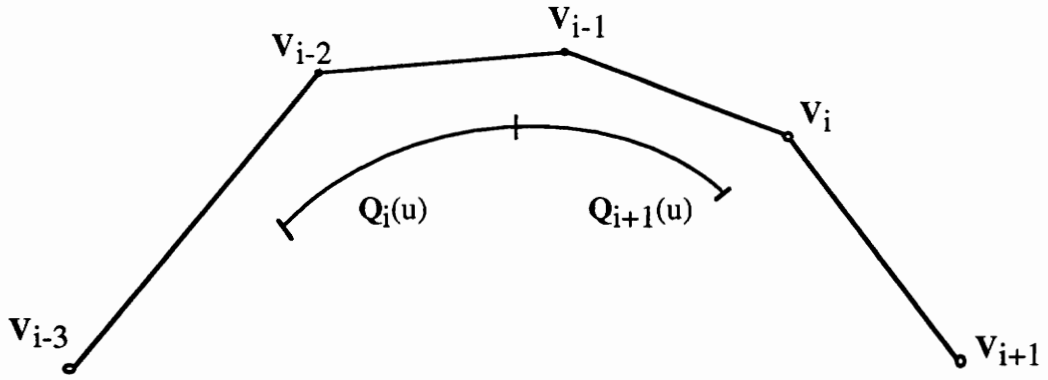
$$\mathbf{Q_i}(u) = \mathbf{V_{i-3}} b_{-3}(u) + \mathbf{V_{i-2}} b_{-2}(u) + \mathbf{V_{i-1}} b_{-1}(u) + \mathbf{V_i} b_{-0}(u)$$

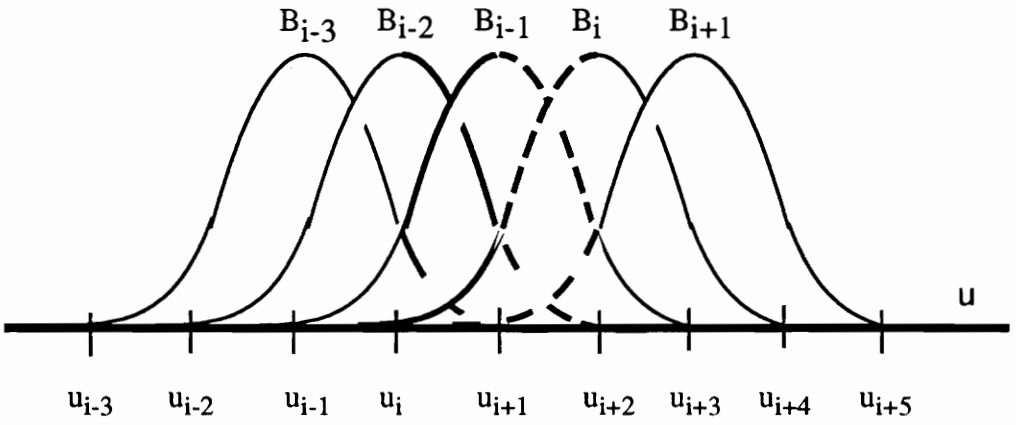
Thus, to define *m* cubic curve segment requires *m* + 3 control points, *m* + 3 basis functions, and *m* + 7 knots. Figure 2 on page 15 depicts a two-segment cubic B-spline curve. Calculating the coefficients for each basis function would be tedious. Fortunately a recurrence relation exists for evaluating basis functions. The Mansfield, de Boor, Cox recursion for basis functions is :

$$B_{i,k}(u) = \frac{u - u_i}{u_{i+k-1} - u_i} B_{i,k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} B_{i+1,k-1}(u)$$

$$B_{i,1}(u) = 1 \text{ if } u_i \le u \le u_{i+1}$$

$$B_{i,1}(u) = 0 \text{ otherwise}$$




$$Q_i(u) = \sum_{i-3}^{i} V_i B_i(u) \qquad Q_{i+1}(u) = \sum_{i-2}^{i+1} V_i B_i(u)$$

$$u = [u_i, u_{i+1}] \qquad u = [u_{i+1}, u_{i+2}]$$

**Figure 2. Multiple Segment Cubic B-spline Curve**

## 3.5 *B-Spline Properties*

B-spline curves possess the following properties:

Local Control:

The movement of one control vertex only affects segments of the curve that use that scaled basis. Moving a single control vertex in a cubic B-spline curve affects four curve segments. This gives the designer local control of the curve. Figure 3 illustrates the movement of one control point of a cubic B-spline curve.

Convex Hull:

The B-spline curve segments lie within the convex hull of the control vertices which define that segment. For cubic B-splines the curve segment lies within the convex hull of the four control vertices that scale the basis functions for that curve segment (figure 4 ).

Affine Invariance:

The B-spline curve may be translated, rotated, or scaled without affecting its shape. This property results from the normalization ($\sum b_r(u) = 1$) of the basis segments. With this property the control net can be rotated, translated, or scaled and the curve computed rather than rotating, translating, or scaling each point on the curve.

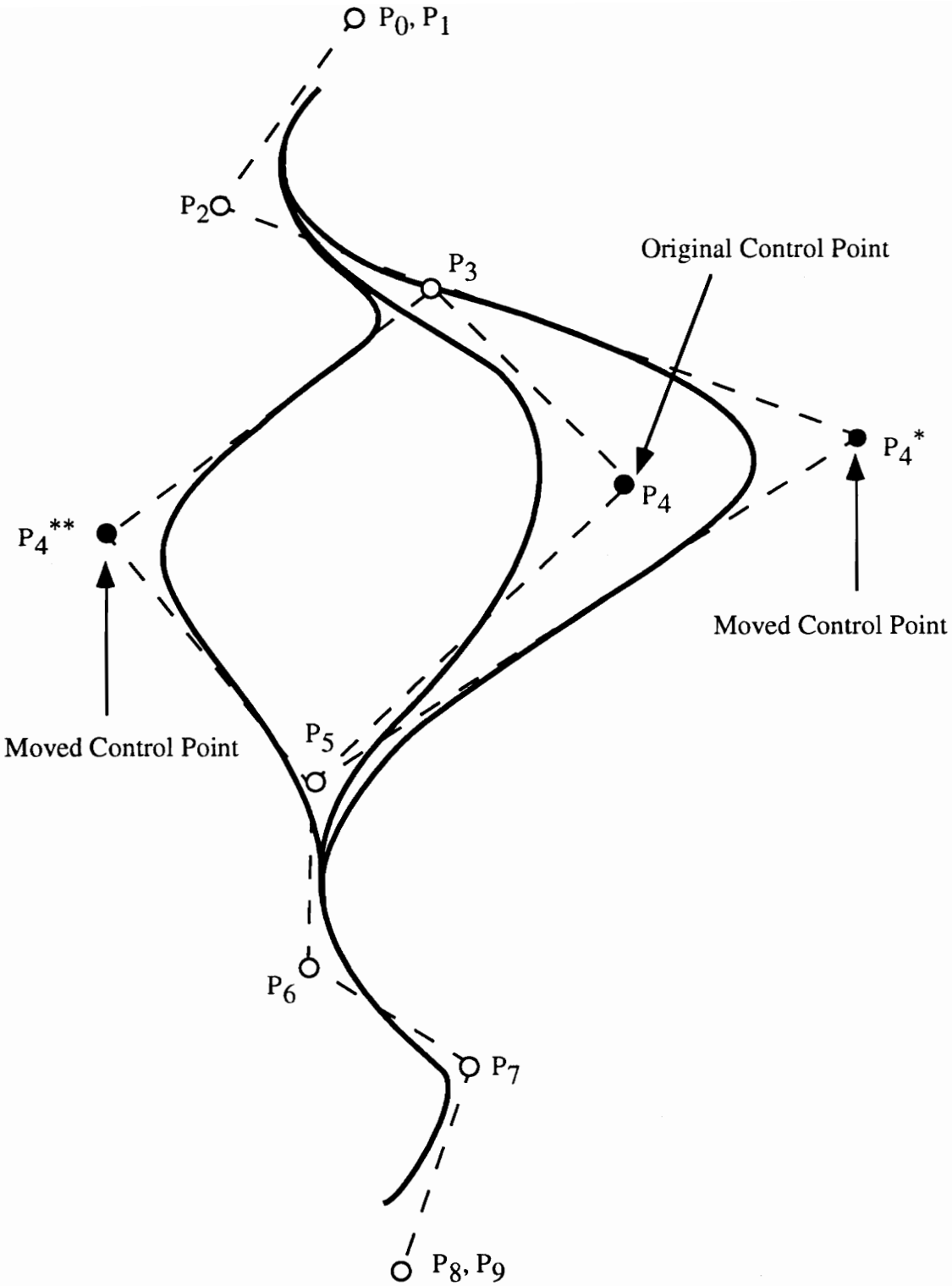



**Figure 3. Moving One Control Point [Tjun93]**

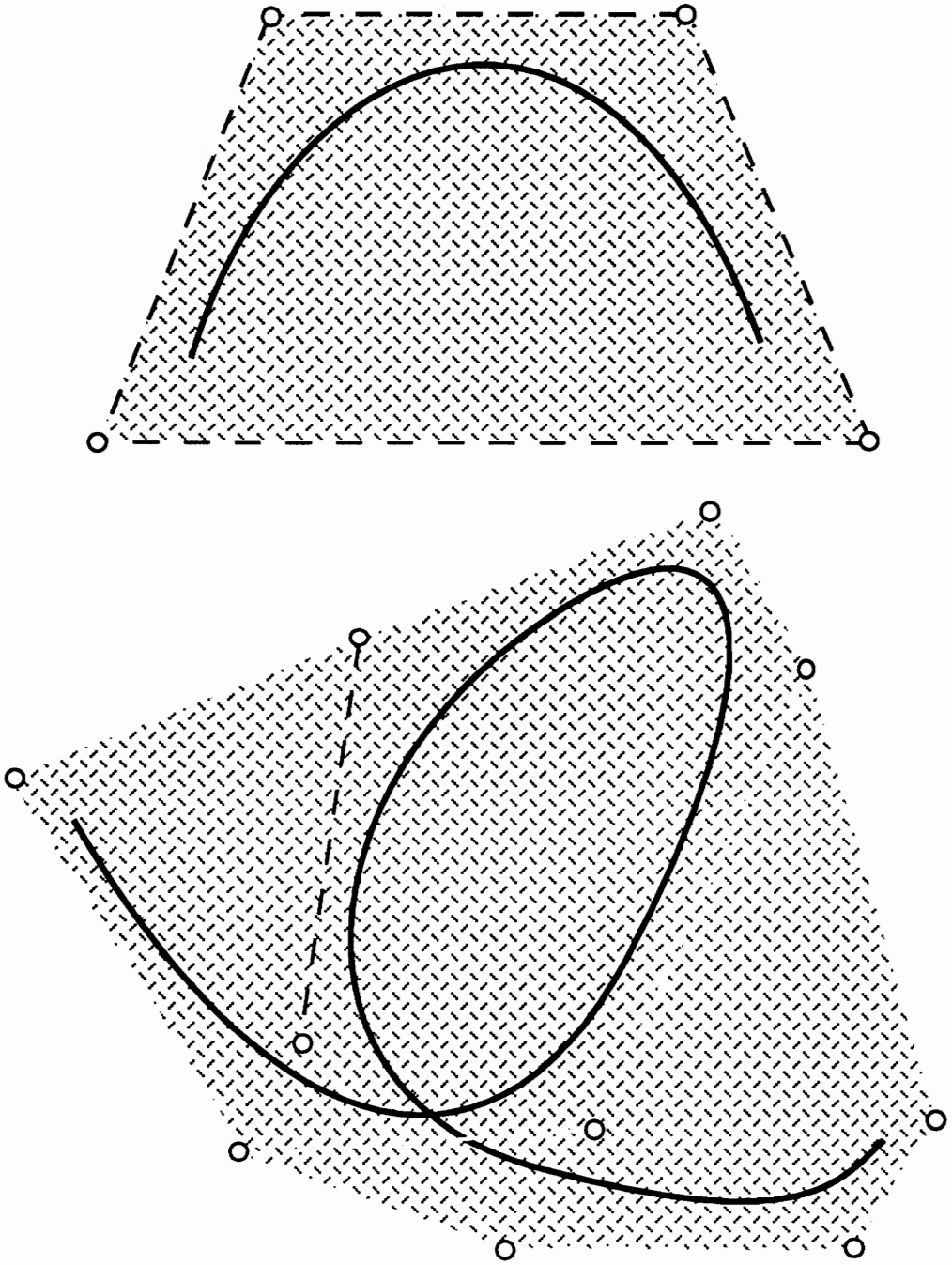


**Figure 4. Convex Hull of a B-Spline Curve Segment [Tjun93]**

Affine Invariance of the Parameter:

The sequence of parameter interval endpoints is called the knot sequence of the B-spline curve. The $u$ -parameter intervals can be scaled without affecting the B-spline curve. This implies that a knot sequence of 0,1,2,3,4,5 is equivalent to 0,0.5,1,1.5,2,2.5.

B-Spline Refinement:

The B-spline curves can be refined with more control vertices by inserting knots into the underlying parameter space. With each new knot a new basis is created and must be weighted by a new control vertex. Formally, the equation is

$$\mathbf{Q}(u) = \sum_{i=0}^{n} \mathbf{V}_i B_{i,k}(u) = \sum_{j=0}^{m+n} \mathbf{W}(N_{j,k})(u).$$

The old vertices are the $\mathbf{V}_i$ 's and the new vertices are $\mathbf{W}_j$ 's. The old basis functions are $B_{i,k}(u)$ and the new basis functions are $N_{j,k}(u)$. A recurrence relation exists for refining B-spline curves. The new vertices can be determined directly from the old vertices using

$$\mathbf{W}_j = \sum_{i=0}^{m} a_{i,k}(j)\mathbf{V}_i,$$

$$a_{i,r}(j) = \frac{u_{j+r-1} - u_i}{u_{i+r-1} - u_i}\, a_{i,r-1}(j) + \frac{u_{i+r} - u_{j+r-1}}{u_{i+r} - u_{i+1}}\, a_{i+1,r-1}(j),$$

$r = 2, 3, 4...$ $k^{th}$ order spline,

$a_{i,1}(j) = 1$ if $u_i \leq u_j \leq u_{i+1}$,

$a_{i,1}(j) = 0$ otherwise.

Figure 5 shows a geometric interpretation of inserting a knot to refine the B-spline curve and to generate new control vertices. The ratios (bolded line to total line length) are mapped from the $u$ -parameter space to the Cartesian space.

## 3.6 *Hierarchical B-spline Curves*

Although B-spline curves possess local control, moving a control point can affect a portion of a curve which the designer does not want to change. This problem can be addressed by refining the B-spline curve near the control point. The refinement procedure produces smaller B-spline curve segments. Moving a control point after this procedure will affect a smaller portion of the curve. Hierarchical B-spline curves provide a method to design curves through refinement and localized control point movement. Creating a hierarchical B-spline curve is a four-step process: overlay creation, curve refinement, control point movement, and hierarchical B-spline creation. These steps are described in the following section.

An overlay curve is, at first, a simple copy of the original B-spline curve. This duplicate curve is refined to "free" control points. A control point is said to be free if the three control points on either side are stationary to maintain the curve's $C^2$ continuity. If the curve designer needs four free control points, then ten knots are inserted into the knot sequence. Three control points, which are not movable, are needed on either side of the free control points to maintain $C^2$ continuity with the original curve, figure 6. The free control points are moved to create the new curve, figure 7. At this point the new B-spline curve and the old B-spline curve are compared. If a portion of the new B-spline curve is the same as the original curve, then those portions are discarded. The B-spline curve is

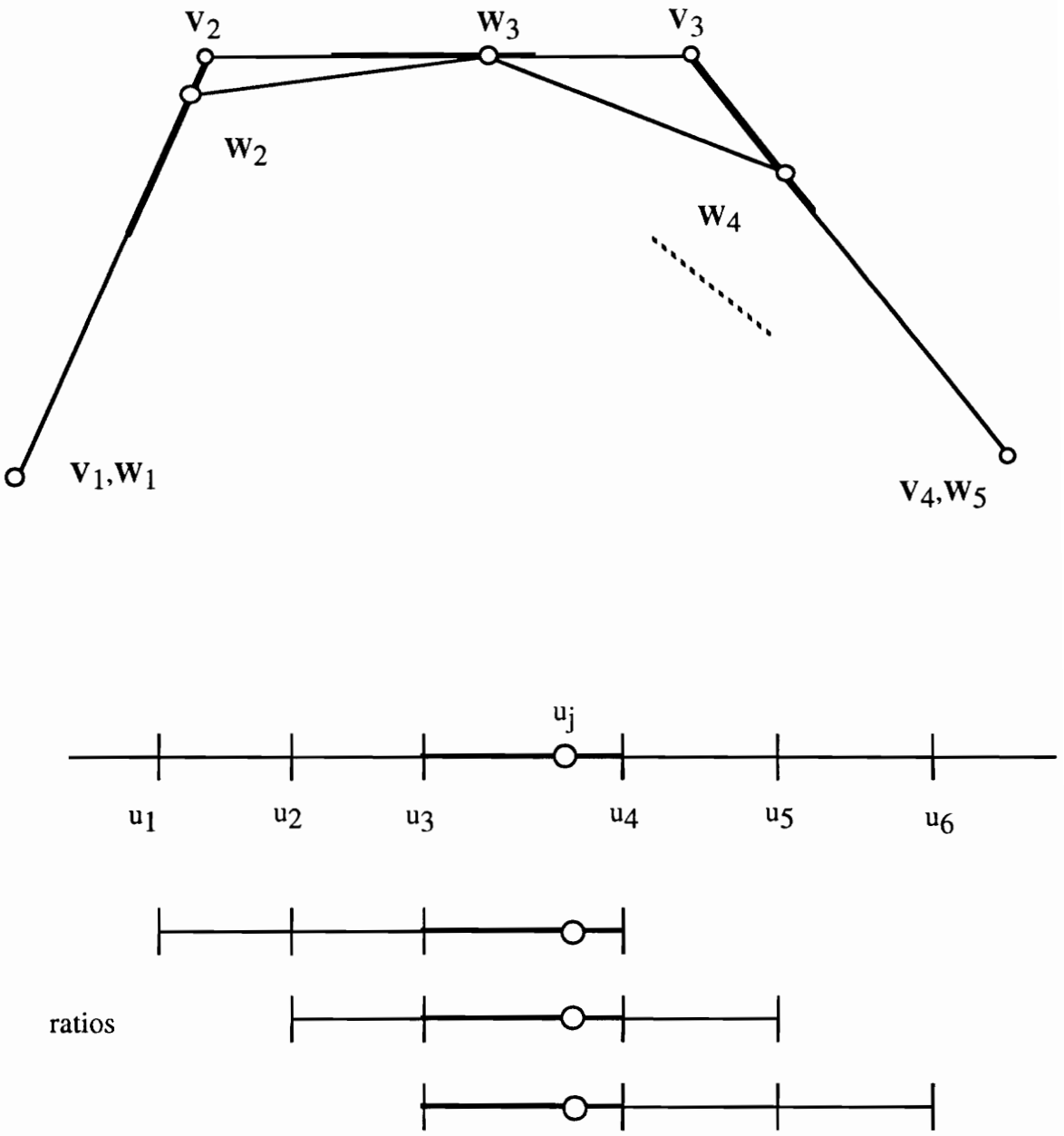
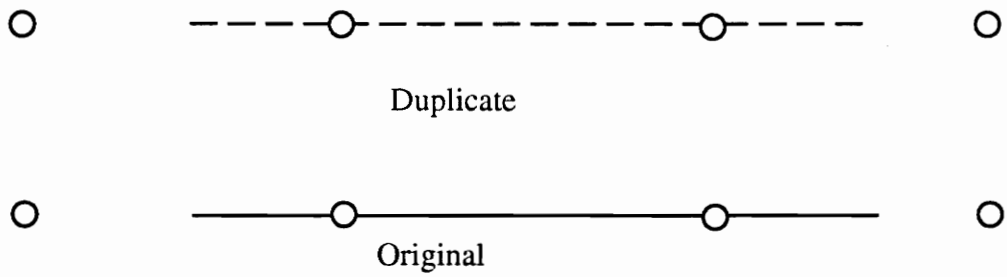



**Figure 5. B-spline Curve Refinement - A geometric interpretation**

Step 1. Overlay Creation




Step 2. Refinement

- Free Control Points
- Non-movable Control Points

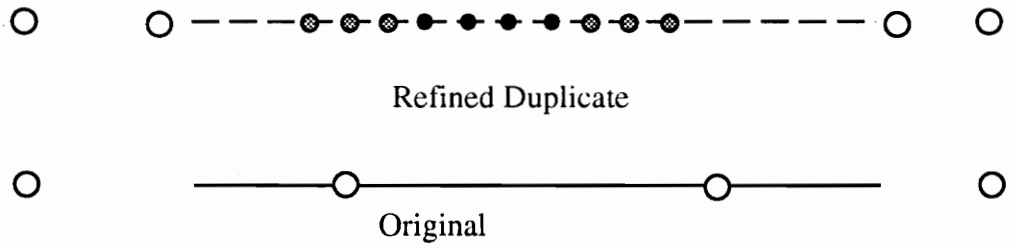



**Figure 6. Free-form Hierarchical Curve Creation - Steps 1 and 2**

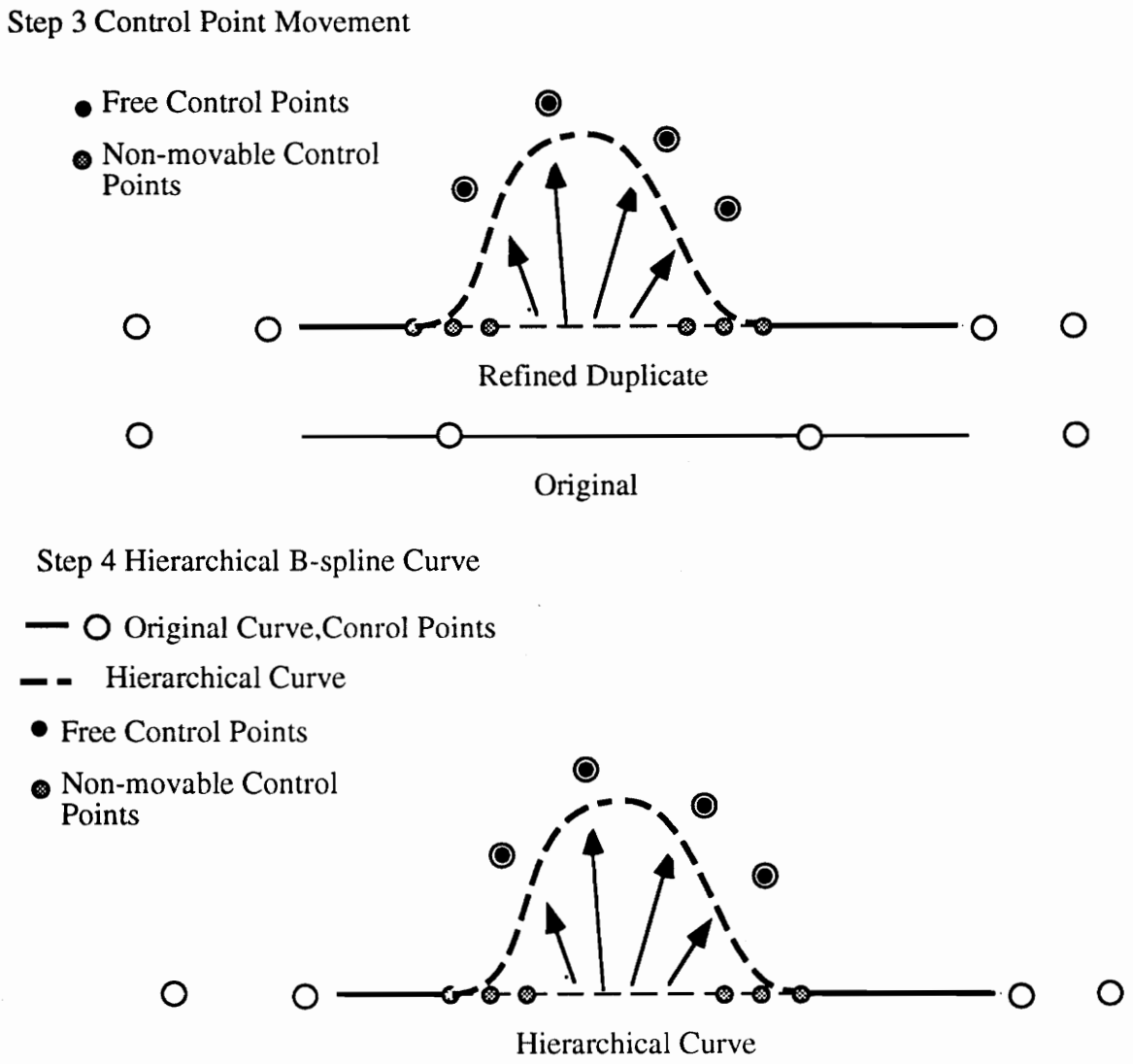
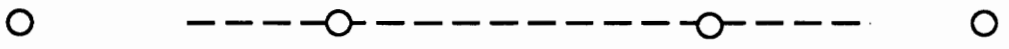



**Figure 7. Free-form Hierarchical Curve Creation - Steps 3 and 4**

now a hierarchical combination of the original and the overlay B-spline curves. As the parameter $u$ is varied, the hierarchical data structure keeps track of which control points and knots are to be used for evaluating the B-spline curve. This process is suitable for graphical applications and been extensively developed by Forsey.

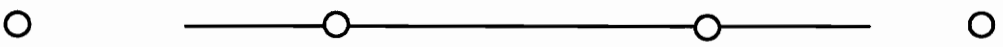
In this research the four-step process has been modified for the engineering problem domain. Specifically, the knot insertion and control point movement are defined by an auxiliary B-spline curve which ensures that the auxiliary B-spline curve is exactly reconstructed in the hierarchical curve. Figure 8 shows the first two steps. Step one takes the original curve and duplicates it to create the overlay curve. Step two has been modified. The curve designer needs four free control points and three stationary control points on either side which provide the $C^2$ continuity with the original curve. Thus ten knots must be inserted into the parameter space of the overlay curve to generate the ten new control points. The auxiliary curve's knot sequence provides eight of the ten knots (ratios). The other two knots may be determined by using the same intervals as the first and last intervals of the auxiliary curve's knot sequence. This knot sequence is affinely mapped into the overlay's knot sequence. Step three is also modified. The free control points of the overlay curve are moved to the position of the auxiliary curve's control points. With the same control points and underlying knot sequence, the auxiliary curve is interpolated. Step four remains the same. The duplicated portion of the overlay curve is removed and the hierarchical curve is created. Figure 9 illustrates steps three and four. As the parameter $u$ is varied, the hierarchical data structure keeps track of which control points and knots are to be used for evaluating the B-spline curve.

Step 1. Overlay Creation



u = { -3.0, -2.0, -1.0, 0.0, 11.0, 12.0, 13.0, 14.0 }

Duplicate



u = { -3.0, -2.0, -1.0, 0.0, 11.0, 12.0, 13.0, 14.0 }

Original

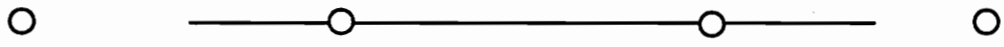
Step 2. Refinement

- Free Control Points
- Non-movable Control Points


u = { -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0 }

u = { -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 }

Refined Duplicate

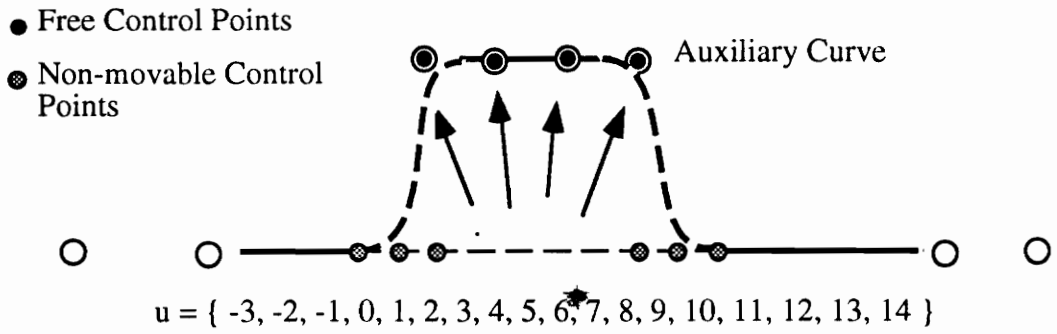


u = { -3.0, -2.0, -1.0, 0.0, 11.0, 12.0, 13.0, 14.0 }

Original

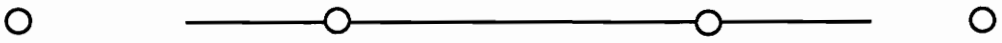
**Figure 8. Hierarchical Curve Creation - Steps 1 and 2**

Step 3 Control Point Movement




u = { -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 }

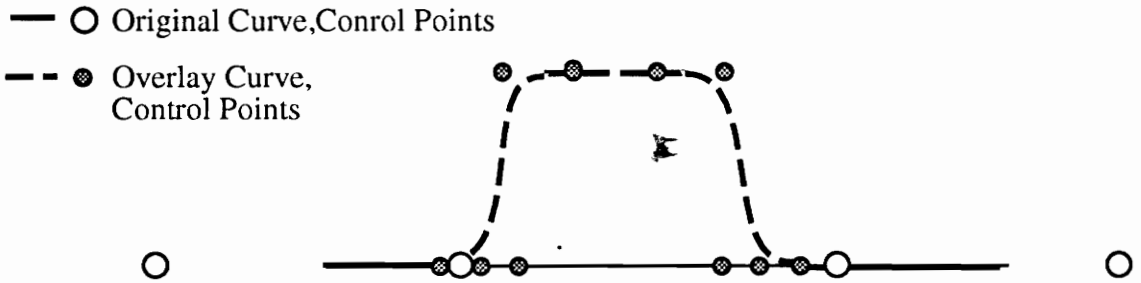
Refined Duplicate



u = { -3.0, -2.0, -1.0, 0.0, 11.0, 12.0, 13.0, 14.0 }

Original

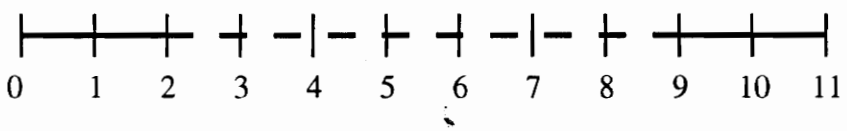
Step 4 Hierarchical B-spline Curve




$u_{overlay}$ = { -1, 0, 1, **2, 3, 4, 5, 6, 7, 8, 9**, 10, 11, 12 }

$u_{original}$ = { -3.0, -2.0, -1.0, **0.0, 11.0**, 12.0, 13.0, 14.0 }

Original and Overlay Curves with Knot Sequences




Hierarchical Curve Parameter Space

**Figure 9. Hierarchical B-spline Curve - Steps 3 and 4**

# 4.0 Tensor-Product B-Spline Surfaces

Tensor-product B-spline surfaces can be thought of as a "two-dimensional" parametric extension of B-spline curves. Instead of a single cubic curve segment defined by *m* vertices and a single knot sequence, a family of B-spline curves is defined by an *m* by *n* mesh of vertices and two independent knot sequences. Mathematically, a tensor product B-spline surface is defined by:

$$\mathbf{Q}(u,v) = \sum_i \sum_j \mathbf{V}_{i,j} B_i(u) B_j(v) \qquad (4.1)$$

The family of B-spline curves defining the surface points is easily seen by constructing the *u* or *v* isoparametric lines . For a single bi-cubic tensor product patch, a four-by-four mesh of vertices is required along with knot sequences on the *u* and *v* domain. Figure 10 schematically shows the sixteen control points and the associated parameter space.

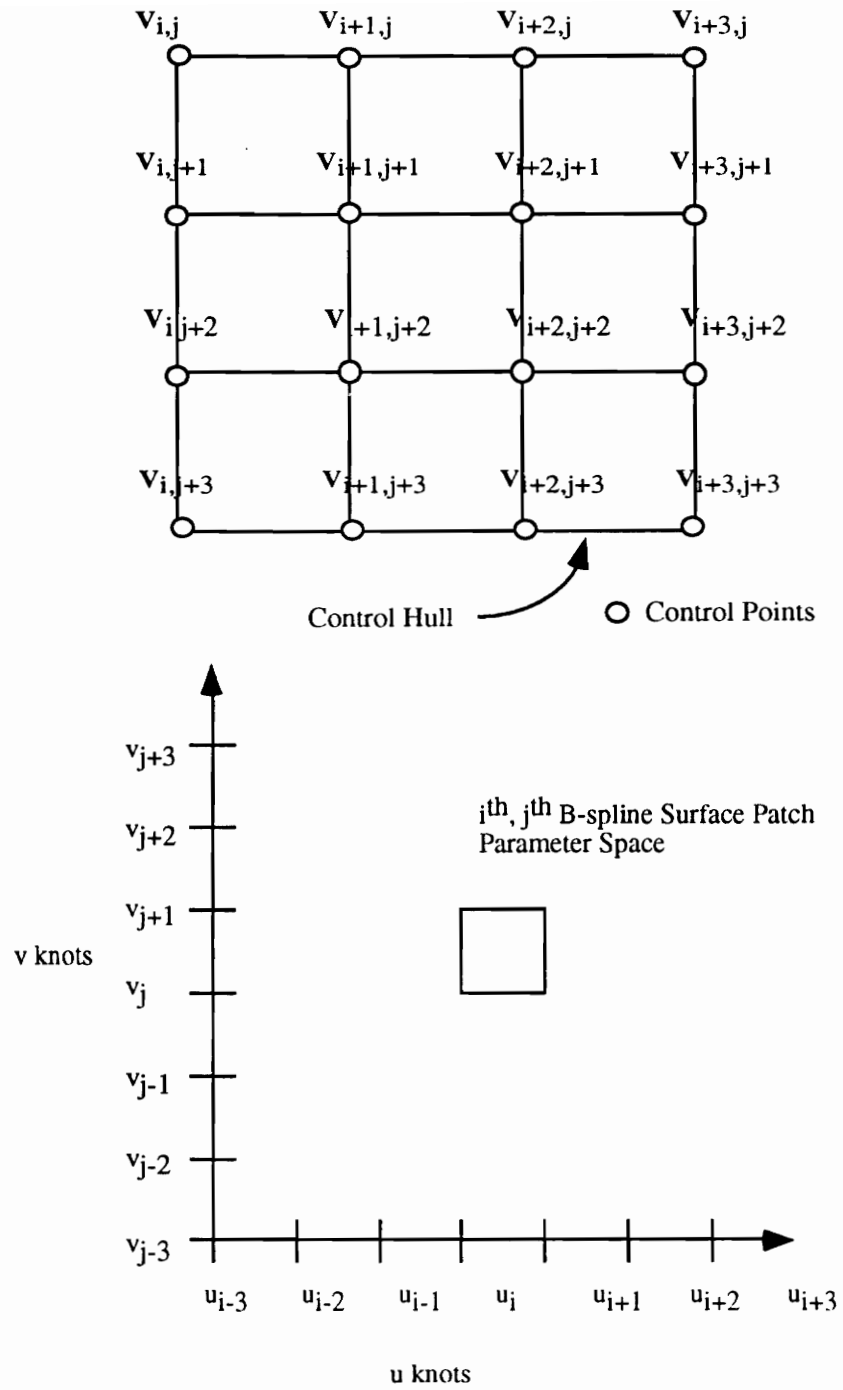
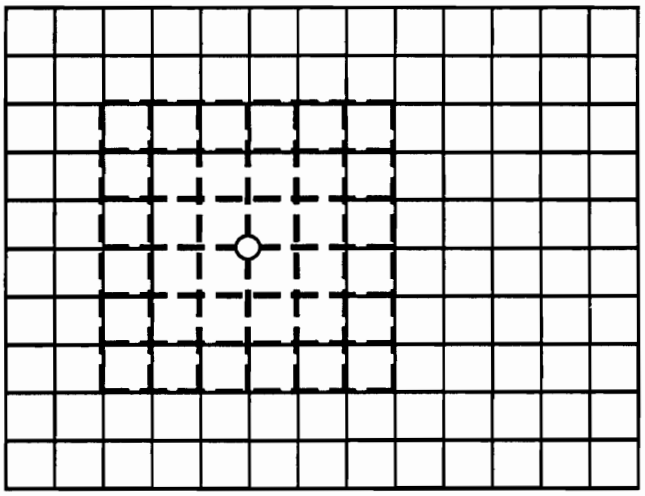
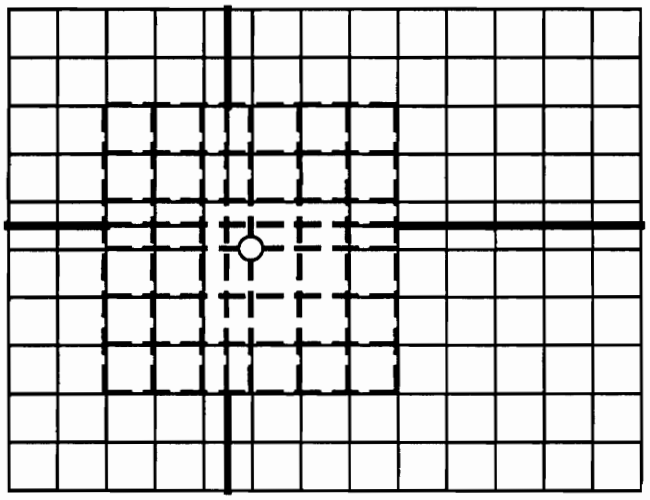



**Figure 10 Single, Bi-cubic Tensor Product B-spline Surface**

The tensor-product B-spline surfaces possess the same properties as B-spline curves. The refinement process is straightforward, but the insertion of a new knot in the knot sequence results in a new row and column of control vertices. Thus , the insertion process is not local but global. Figure 11 shows the effect of inserting one knot in the $u$ and $v$ knot sequences. The surface designer affects the entire control point net although that was not his intent. The area influenced by one control point is also affected. In the following chapter this thesis develops non-uniform hierarchical B-spline surfaces to solve these problems.



Before Refinement



After Refinement

**Figure 11 Control Vertices Before and After Knot Insertion**

# 5.0 Hierarchical B-Spline Surfaces

## 5.1 *Background*

The uniform, hierarchical B-spline surface is a free-form surface design method developed by Forsey in 1988 [Fors88] to provide localized refinement of a tensor-product B-spline surface. Localized refinement permits more detailed features to be added to a surface without globally affecting the control points of the surface. The uniform hierarchical B-spline surface was originally designed for graphical animation and free-form surface design - a problem domain without engineering constraints. An animator using Forsey's uniform hierarchical B-spline surface can refine and move free control points to create a surface with both general and specific features. For example, an animator can create a face with highly detailed eyes and nose features while maintaining the cheek (figure 12). In contrast to the animator's goals, the engineering designer often requires tight constraints on a surface - such as interpolating cross-sections. A designer wishing to loft a surface through cross-sections, a relatively simple task, will generate constraints which are only addressed by non-uniform B-spline surfaces. Attempting to use Forsey's uniform hierarchical B-spline surfaces to meet these types of constraints would be difficult as the control points are not, in general, interpolated.

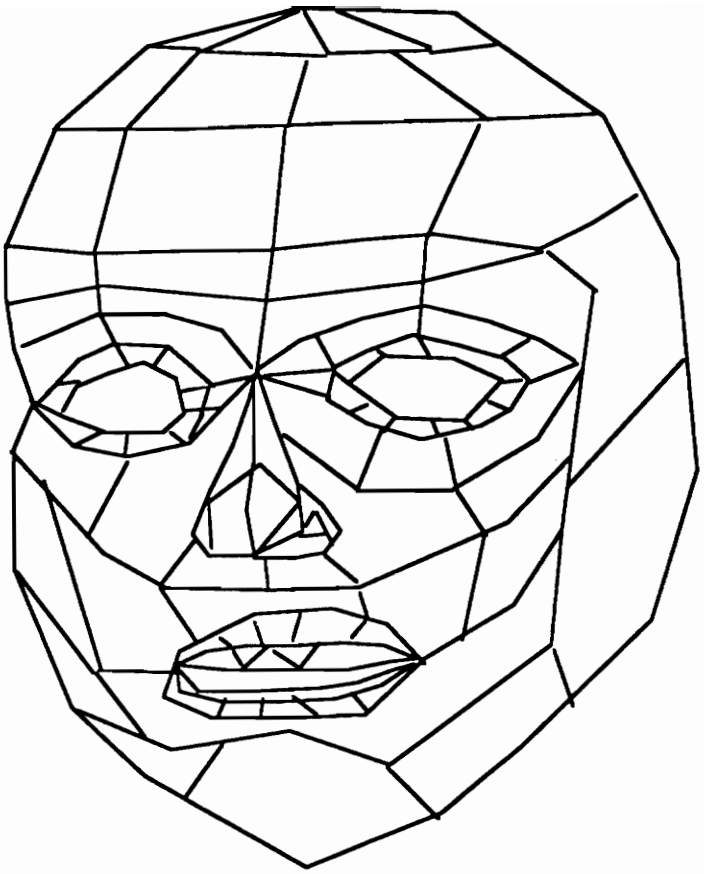


**Figure 12. Surface with General and Specific Features**

By developing algorithms for non-uniform hierarchical B-splines, the hierarchical free-form surface-design method can be utilized in the engineering problem domain. Specifically, the ability to add parametrically defined components to a free-form surface enhances the design process (using bicubic surfaces) by providing a $C^2$ model for use in the engineering conceptual design phase. Moreover it provides a design method which permits finer details to be added throughout (and at any point in the) design process while still maintaining a $C^2$ surface. The following sections describe the research accomplished in this thesis - creating a non-uniform hierarchical B-spline surface.

## 5.2 *Non-Uniform, Hierarchical B-Spline Surfaces*

The hierarchical B-spline surface development is characterized by a multi-step process: overlay surface creation, base surface refinement, control point movement and hierarchical surface creation. First a duplicate (overlay) surface is created. This surface must be refined to free control points. Next the free control points are moved to achieve the desired shape. The overlay and the original surface are then compared; those parts of the overlay surface which duplicate the original surface are discarded. Finally, the hierarchical data structure for the hierarchical surface is created. This data structure keeps track of hierarchical level, knot sequences, and control nets for each surface. This data structure permits every $u,v$ point to be properly evaluated. The simplest form of this process is the freeing and moving one control point.

### 5.2.1 *"Point" Component Example*

This example illustrates the basic steps in creating a non-uniform, hierarchical B-spline surface. An overlay surface is created, the surface is refined, a single control point is

moved, the overlay and original surfaces are compared (discarding duplicate information from the overlay surface), and the hierarchical surface is defined. This example also demonstrates how the refinement process is localized.

The base surface used in this example is a planar bi-cubic B-spline patch defined by a four-by-four mesh of control points defined over the following domains (knot sequences).

$$u = \{-3,-2,-1,0,1,2,3\}$$

$$v = \{-3,-2,-1,0,1,2,3\}$$

The surface is duplicated to create the overlay surface. The overlay surface is refined by inserting seven new knots in both the *u* and *v* knot sequences. The modified knot sequences are :

$$u = \{-3,-2,-1,0,0.2,0.3,0.4,0.5,0.6,0.7,0.8,1,2,3\}$$

$$v = \{-3,-2,-1,0,0.2,0.3,0.4,0.5,0.6,0.7,0.8,1,2,3\}$$

The corresponding control vertices of the refined overlay surface are schematically depicted in figure 13. The middle control point of the overlay is now moved to create the point feature. The overlay and original surfaces are now compared. The surface defined by the middle control point and the 48 surrounding control points define a surface which is different from the original. Outside this region the original and overlay surfaces are exactly the same. The duplicated area of the overlay surface is removed. Two surfaces now exist: the original surface and the overlay surface as shown in figure 14. By keeping track of their hierarchical relationship, the two surfaces are a single hierarchical surface. This approach exploits the local control and refinements properties of the B-

spline to ensure surface continuity with the base surface. In short, no control points on the overlay surface are moved which will affect the continuity. The knot sequences for each surface and the effective domains are presented below.

Original Surface

$u = \{-3,-2,-1,0,1,2,3\}$ $\quad [0 \leq u \leq 0.2]$ and $[.8 \leq u \leq 1.0]$

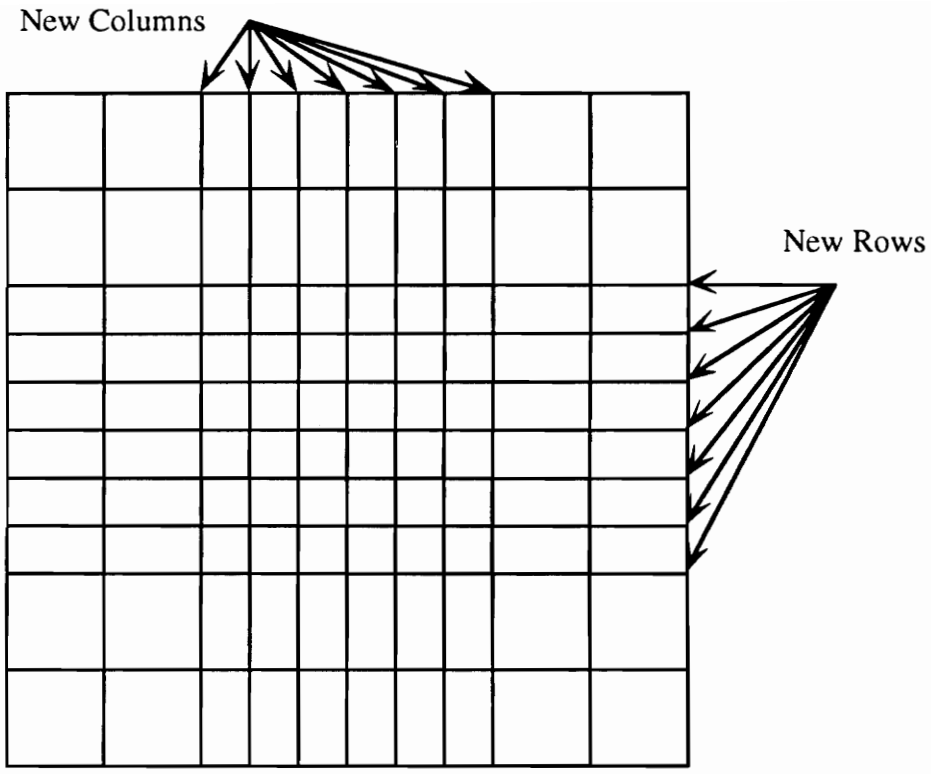
$v = \{-3,-2,-1,0,1,2,3\}$ $\quad [0 \leq v \leq 0.2]$ and $[.8 \leq v \leq 1.0]$

Overlay Surface

$u = \{-1,0,0.2,0.3,0.4,0.5,0.6,0.7,0.8,1,2\}$ $\quad [0.2 \leq u \leq 0.8]$

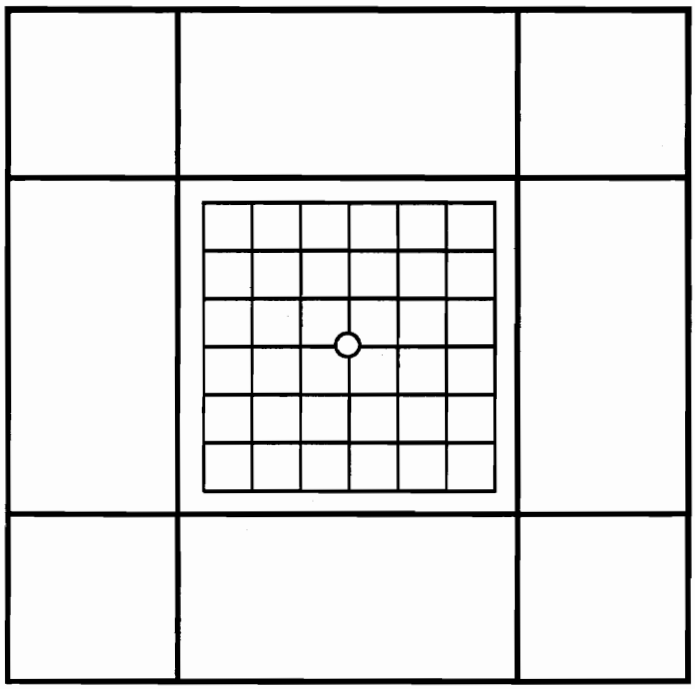
$v = \{-1,0,0.2,0.3,0.4,0.5,0.6,0.7,0.8,1,2\}$ $\quad [0.2 \leq v \leq 0.8]$

With this information every $u,v$ point can be evaluated with the appropriate control net and knot sequence. The $u,v$ point (0.1,0.1) is in the original surface. The $u.v$ point (0.4,0.6) is in the overlay surface. Rendering isoparametric curves can be accomplished by noting which surface information to use when evaluating a particular $u,v$ point. Figure 15 shows the isoparametric curves of the "point" component. A more complex form of this process is described in the next section where a box component is added to a B-spline surface.




Schematic of Control Vertices after Refinement

**Figure 13. Overlay Surface Refinement**



Original Surface Control Net

Overlay Surface Control Net

Moved Control Point

**Figure 14. Original and Overlay Surfaces**

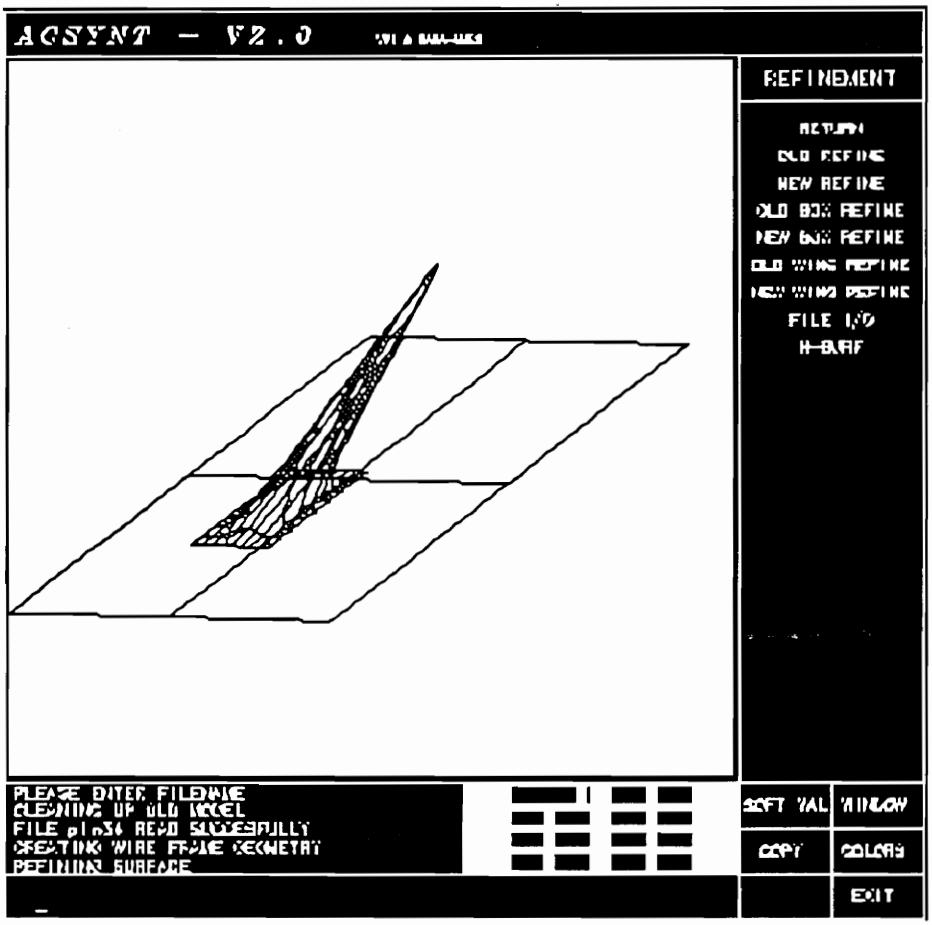



**Figure 15. Isoparametric Curves of the "Point" Component**

### 5.2.2 *Box Component Example*

This example follows the same multi-step process discussed in the previous section. The difference is that a predefined box component determines the number of knots to be inserted into the $u$ and $v$ knot sequences and provides the locations for the control point movement. As in the previous example, the base surface will be a planar bi-cubic patch defined by a four-by-four mesh of control points. The overlay surface is refined by inserting knots. If the box component's number of $u$ knots is $m$ and its number of $v$ knots is $n$, then $(m\text{-}4)\text{x}(n\text{-}4)$ control points must be freed on the base surface. To free those control points, $(m\text{-}4+6)$ knots must be affinely mapped and inserted into the overlay surface's $u$ knot sequence and $(n\text{-}4+6)$ knots in the $v$ knot sequence. The six extra knots are needed to maintain the $C^2$ continuity with the base surface. The affine mapping preserves the original parameterization of the predefined component in the overlay. For example, if the overlay $u$ knot sequence is

$$u = \{0,-3,-2,-1,0,1,2,3\},$$

and the box component $u$ knot sequence is

$$u_{box} = \{-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10\},$$

then one possible overlay $u$ knot sequence that preserves the box component's shape would be

$$u = \{-3,-2,-1,0,.01,.02,.03,.04,.05,.06,.07,.08,.09,.10,.11,.12,.13,.14,.15,16,1,2,3\}.$$

After the knot insertion in the overlay surface is complete, the new control points are moved to the box component's control point locations. The duplicate surface area in the

overlay surface is trimmed and the hierarchical data structure is created to manage the hierarchical surface information. Figure 16 shows a hierarchical surface with a box component.

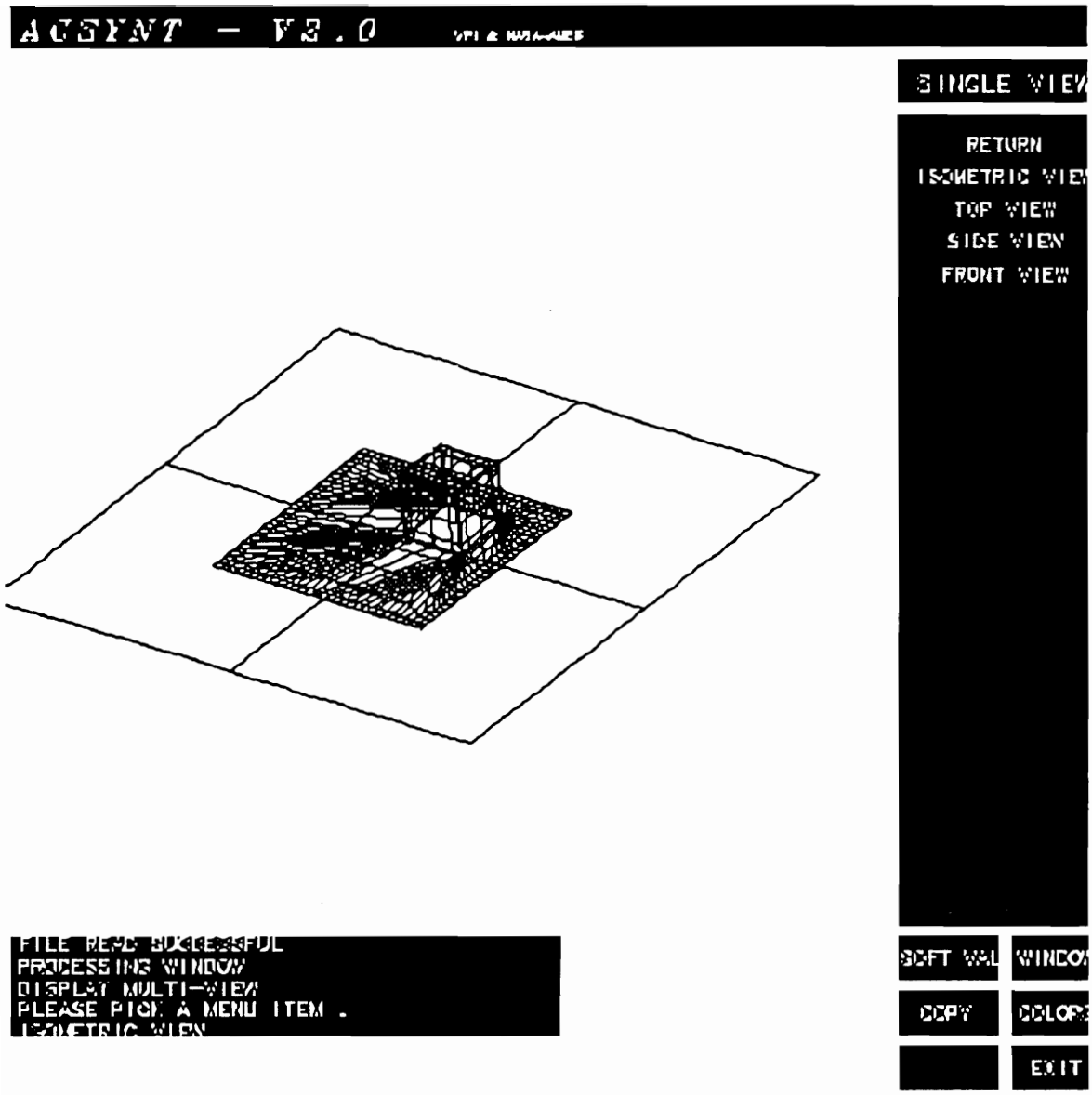



**Figure 16. Hierarchical Surface with a Box component**

## 5.3 *Hierarchical Surface Advantages*

The advantages of the non-uniform, hierarchical B-spline surface are two-fold. The hierarchical data structure localizes the refinement procedure and the non-uniform B-splines surfaces accommodate typical engineering features and components. Localization of the B-spline surface refinement operations reduces the number of control points needed to describe a tensor-product surface. If the base surface has $m \times n$ control points and has a feature with $k \times l$ control points added to it, then the traditional tensor-product surface would need to store $m \times n \times k \times l$ control points. In contrast, the hierarchical B-spline surface needs only $m \times n + (k+6) \times (l+6)$ control points to represent the same surface. The geometric implications of localizing the refinement procedure are also important. Features and components can be added to a surface without "getting in each other's way." In this way, parametrically defined components can be easily attached to a B-spline surface. This capability is demonstrated in ACSYNT by attaching parametrically defined wings to a fuselage.

# 6.0 ACSYNT (AirCraft SYNThesis)

## 6.1 *Background*

ACSYNT is a standards-based system for feature-based, parametric, computer aided conceptual design of aircraft. Since 1986 ACSYNT has been jointly developed by NASA Ames and the Virginia Tech CAD Laboratory's ACSYNT Institute. ACSYNT's geometric modeling capability is extensive, but does not provide a single $C^2$ surface. Instead, each component is a $C^2$ surface. A $C^2$ surface model can be readily analyzed for its radar backscatter signature (RCS) [Rein90]. A $C^2$ surface is also needed for the good mesh generation required for computational fluid dynamic (CFD) and infrared (IR) analysis. With a complete, $C^2$ non-uniform hierarchical B-spline surface within ACSYNT, the CFD, IR, and RCS analysis, which is typically performed during the preliminary design phase, may now be addressed in the conceptual design phase. The following sections describe ACSYNT's model and geometric data structures and explains how the non-uniform hierarchical B-spline data structure, developed in this thesis, is blended into these structures.

## 6.2 *ACSYNT Geometric Data*

ACSYNT employs a linked list of components to store all data in a centralized format. The beginning of the linked list starts with the model data structure, figure 19. This structure controls the pointer to the linked list of components. The model data structure is designed to minimize the need to pass variables between functions; only a pointer to the model data structure is needed to pass all the geometric information to a particular function. The component data structure (figure 20) contains the geometric information of each component. Figure 21 depicts the linked-list structure of a simple aircraft model.

## 6.3 *Hierarchical Surfaces and ACSYNT*

The hierarchical B-spline surface within ACSYNT utilizes the same geometric model data structure and adds a combination linked-list/tree-data structure, figures 20 and 21, to maintain the hierarchical surface information. The parametric space information of each component is mapped into this new data structure. The process of creating a single, $C^2$ surface aircraft is described in the following steps which mirrors the process of creating hierarchical B-spline curves.

```
typedef struct {
        int num_comp;          /* number of components contained in model */
        int acs_root;          /* structure id executes all other structures */
        int nubs_root;         /* Non-uniform B-spline root id */
        int int_root;          /* structure id containing intersection curves */
        int fillet_root;
        int shade;
        int gauss_k;
        int mean_k;
        comp_data *comp;       /* pointer to linked list of components */
        struct intersection_type *intlist;      /* list of intersections */
        struct strukct_comp *ptrcomp;           /* list of intersections */
        } MODEL;
```

**Figure 17. Model Data Structure**

```
typedef struct      compdata_type{
        int comp_number;                    /* component number */
        char comp_name[20];                 /* component name */
        int acs_id;                         /* structure id */
        int nubs_id;
        int *hull_id;
        int fillet_id;
        int glob_flag;
        int open[2];                        /* open flag 1 closed 0 */
        int color;                          /* component color */
        int comp_type;                      /* component type*/
        int nu;                             /* number of u isoparametric lines */
        int nw;                             /* number of w isoparametric lines */
        int acs_ncross;                     /* number of cross sections */
        int acs_npts;                       /* # of points per x-section */
        float ***acs_pts;                   /* pointer to component points*/
        float ***acs_utan;                  /* pointer to tangents in u dir */
        float *** acs_wtan;                 /* pointer to tangents in w dir */
        int nu_knots;                       /* number of u knots */
        int nw_knots;                       /* number of w knots */
        float *u_knots;                     /* u knot array */
        float *w_knots;                     /* w knot array */
        float ***hull;                      /* control hull *;
        Nurbs * nurbs_data;                 /* NURBS  data */
        struct compdata_type *next;         /* pointer to next component */
        } comp_data;
```

**Figure 18. Component Data Structure**

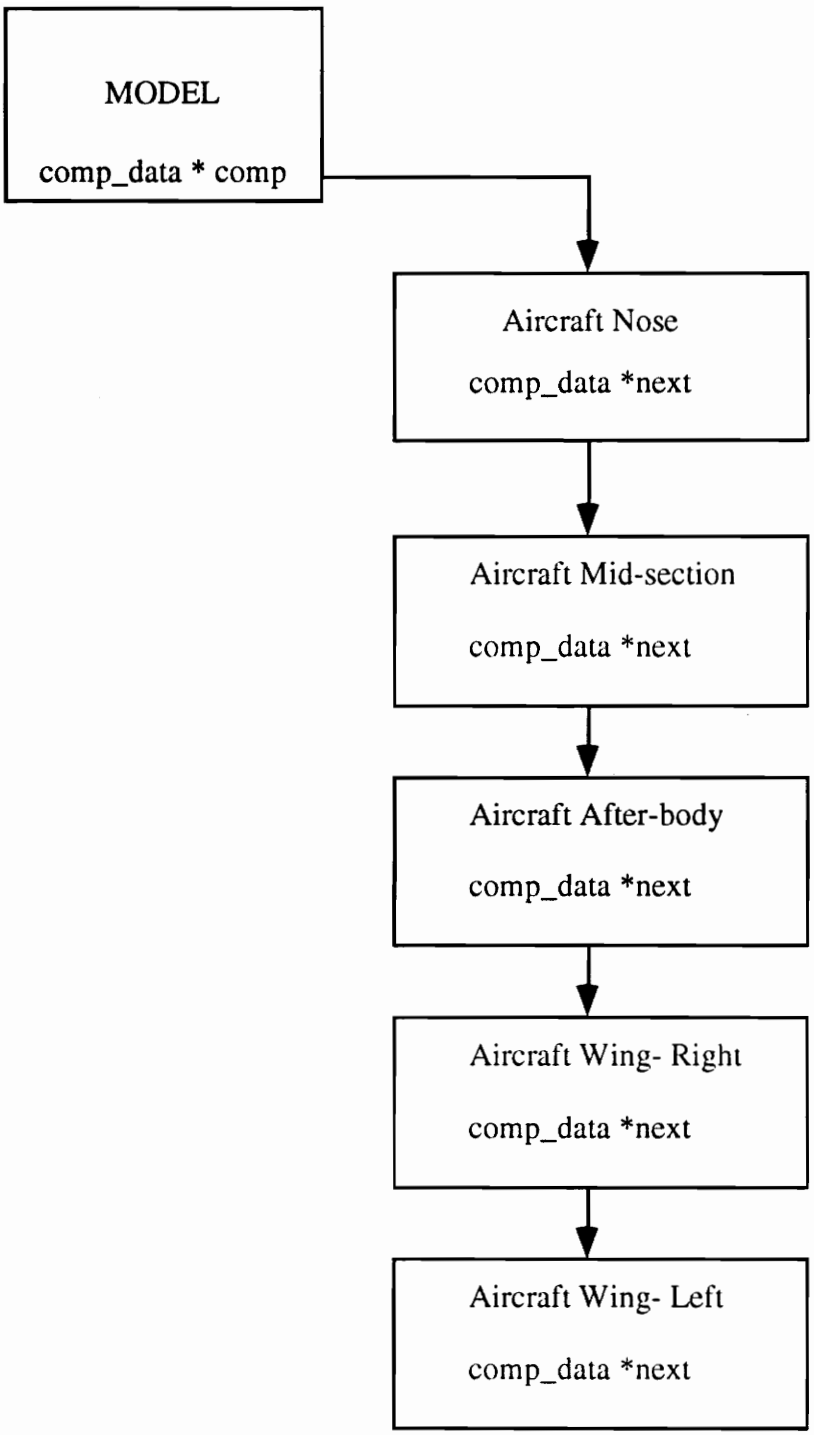
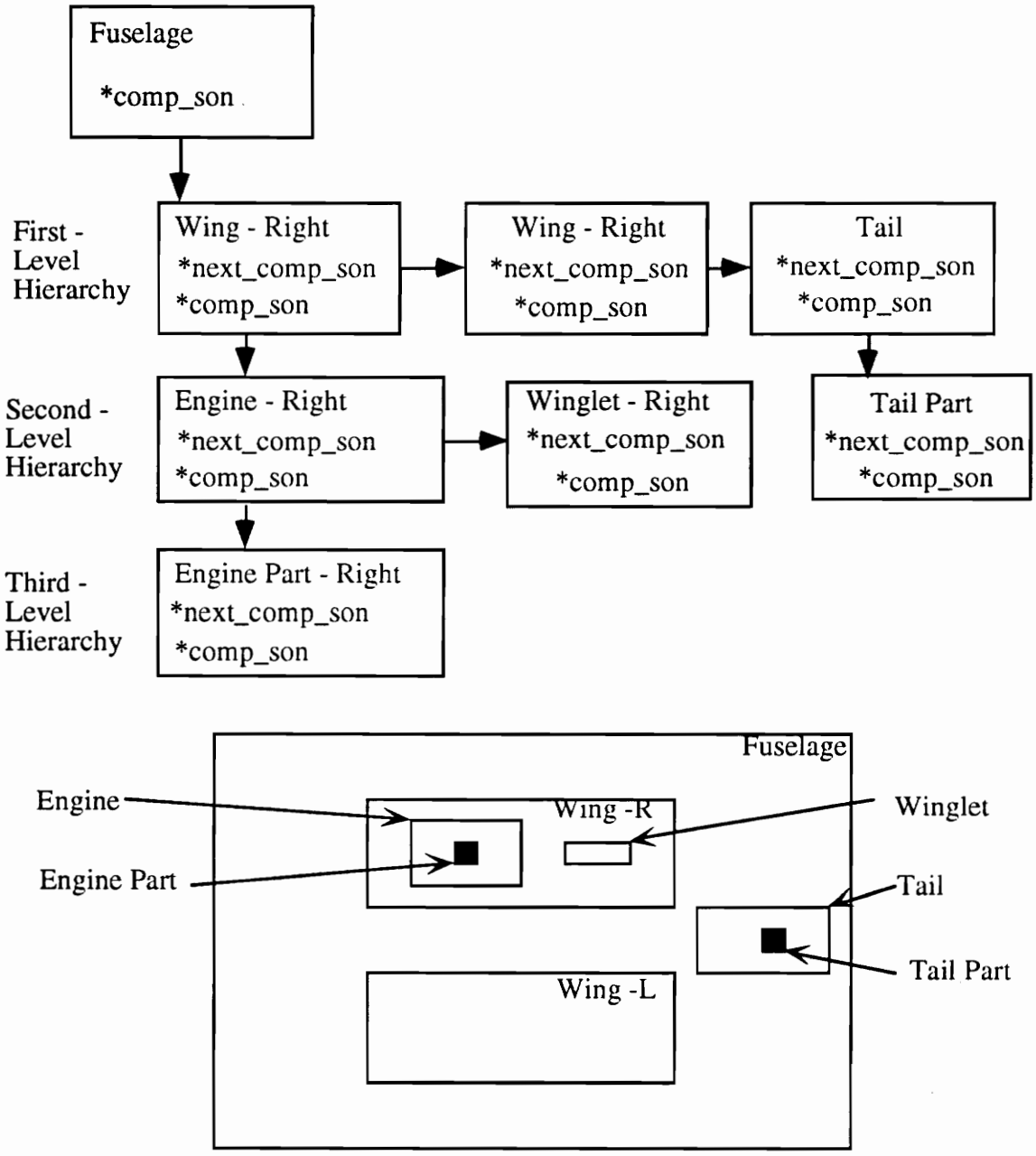



**Figure 19. Aircraft Model Linked List**

```
typedef struct node{
        char *info;                     /* ptr to component name */
        struct node *next_comp_son;     /* next comp for this level */
        struct node *comp_son;          /* next level H-surface */
        float u_top;                    /* parametric space top */
        float u_bottom;                 /* parametric space bottom */
        float  v_left;                  /* parametric space left */
        float v_right;                  /* parametric space right */
        } COMP_NODE;
```

**Figure 20. Non-uniform, Hierarchical B-spline Data Structure**







Parameter Space of Hierarchical Aircraft Model

**Figure 21. Hierarchical Tree of an Aircraft's Parameter Space**

**Step 1.** - The aircraft fuselage - composed of the nose, mid-section, and aft-body components - is inverted as a single non-uniform B-spline surface. This surface (control points and knot sequence) is loaded into a new Model data structure. The bounding $u,v$ parameters of the fuselage are placed in the root position of the non-uniform hierarchical B-spline surface data structure, figure 22.

**Step 2.** - The right wing component is modified so that its control points lie outside the control net of the fuselage. After modification, the right wing is added to the new Model data structure. To add the wing to the hierarchical data structure, the "bounding" $u,v$ space on the fuselage must be found. This is accomplished by finding the intersection of four line with the fuselage. The first line is defined by control points on the leading edge of the wing. The second line is determined by the trailing edge control points. The third and fourth lines are determined by the upper and lower control points of the wing. The intersection of the these lines with the fuselage is found by searching the fuselage's $u,v$ space for points whose distance between these lines and B-spline surface points at $u$, $v$ is "zero". The algorithm follows the following logic.

(1) Start at fuselage parameter point $u_0$, $v_0$

(2) Evaluate the distance between the line and the fuselage surface at $u_0$, $v_0$

(3) If the distance is "zero", then the intersection point is found

(4) If the distance is not "zero", then increment $u$, $v$ to decrease the distance

(5) Go to step 2

**Step 3.** - Once these four intersection points are found, then refinement and control point movement may be performed.

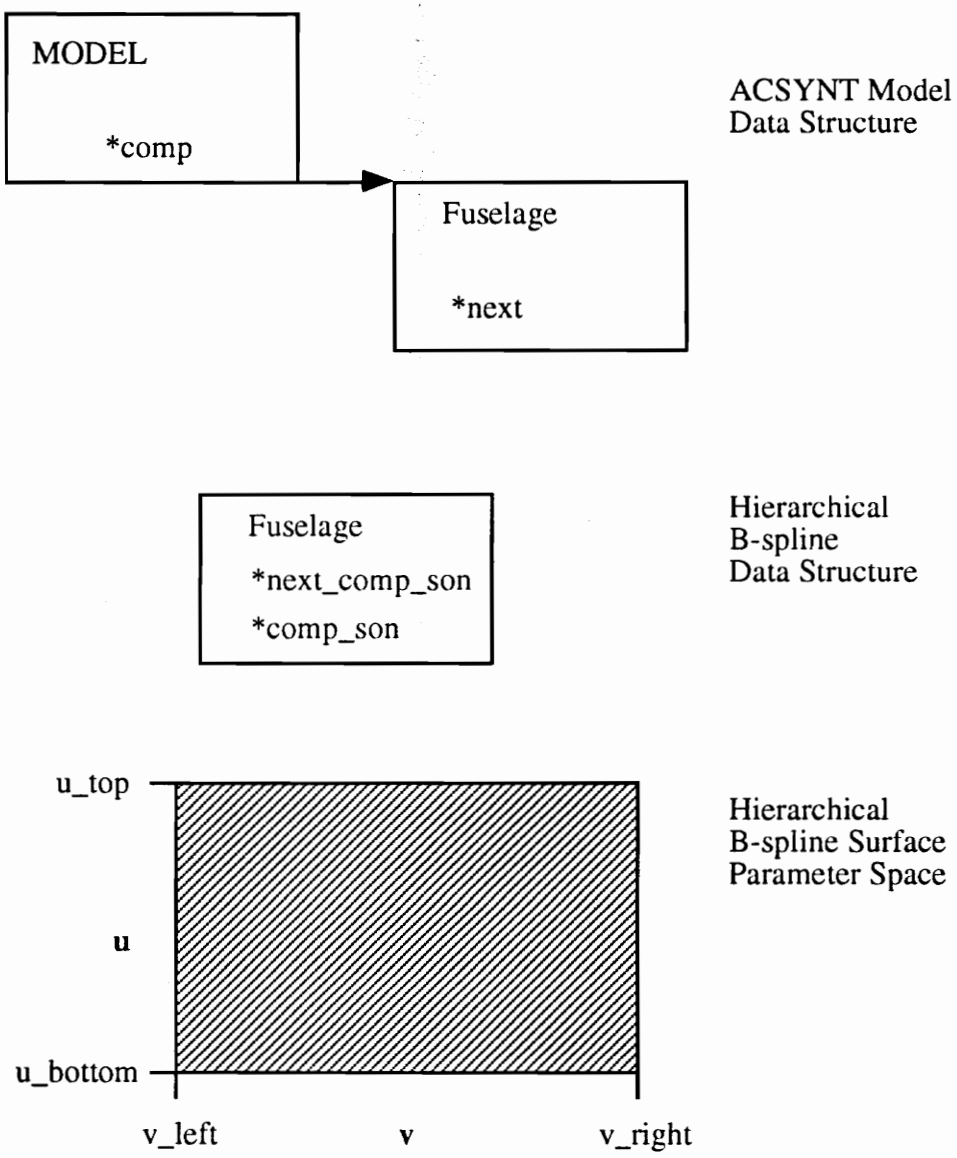



**Figure 22. The Fuselage in Hierarchical Data Structure**

**Step 4.** - Next, the non-uniform hierarchical B-spline surface data structure is updated. Figure 23 schematically represents this data structure after one component addition. This step is repeated until all the desired components have been added. Figure 24 presents the sequence of steps executed to skin an airplane composed of nose, mid-section, aft-body, wing, horizontal tail and vertical tail components.

Skinning an aircraft within ACSYNT is straight-forward. The aircraft geometry is loaded. The user proceeds down the menu hierarchy, figure 25, and selects AUTO-SS, AUTOmatic Single-Surface. At this point the user may select FILE I/O to write an ACSYNT B-spline file; select SHADE SS to shade the plane; choose SHADE MULTI to shade the original B-spline model; or pick SS NET to view the hierarchical B-spline control net.

## 6.4 *Rendering Isoparametric Curves of Non-uniform Hierarchical B-spline Surfaces*

ACSYNT rendering algorithms were modified to accommodate the non-uniform hierarchical B-spline data structure. ACSYNT isoparametric curves are simply poly-lines connecting pints on the surface. Typically a $v$ isoparametric curve of an ACSYNT component is rendered by evaluating points on the surface for a given $v$ and a varying $u$ parameter. The $v$ parameter usually begins at $v_{begin}$ and ends at $v_{end}$ where $v_{step}$ is equal to $(v_{end} - v_{begin})/constant$. This algorithm is modified for rendering non-uniform hierarchical B-spline isoparametric curves by generating a list of parameter values at which $v_{step}$ must be modified. In figure 23 isoparametric curve A lies completely within the fuselage's parameter space. Isoparametric curve B goes through the fuselage and wing's parameter space. The $v_{step}$ is isoparametric curve B must change when $u$ passes

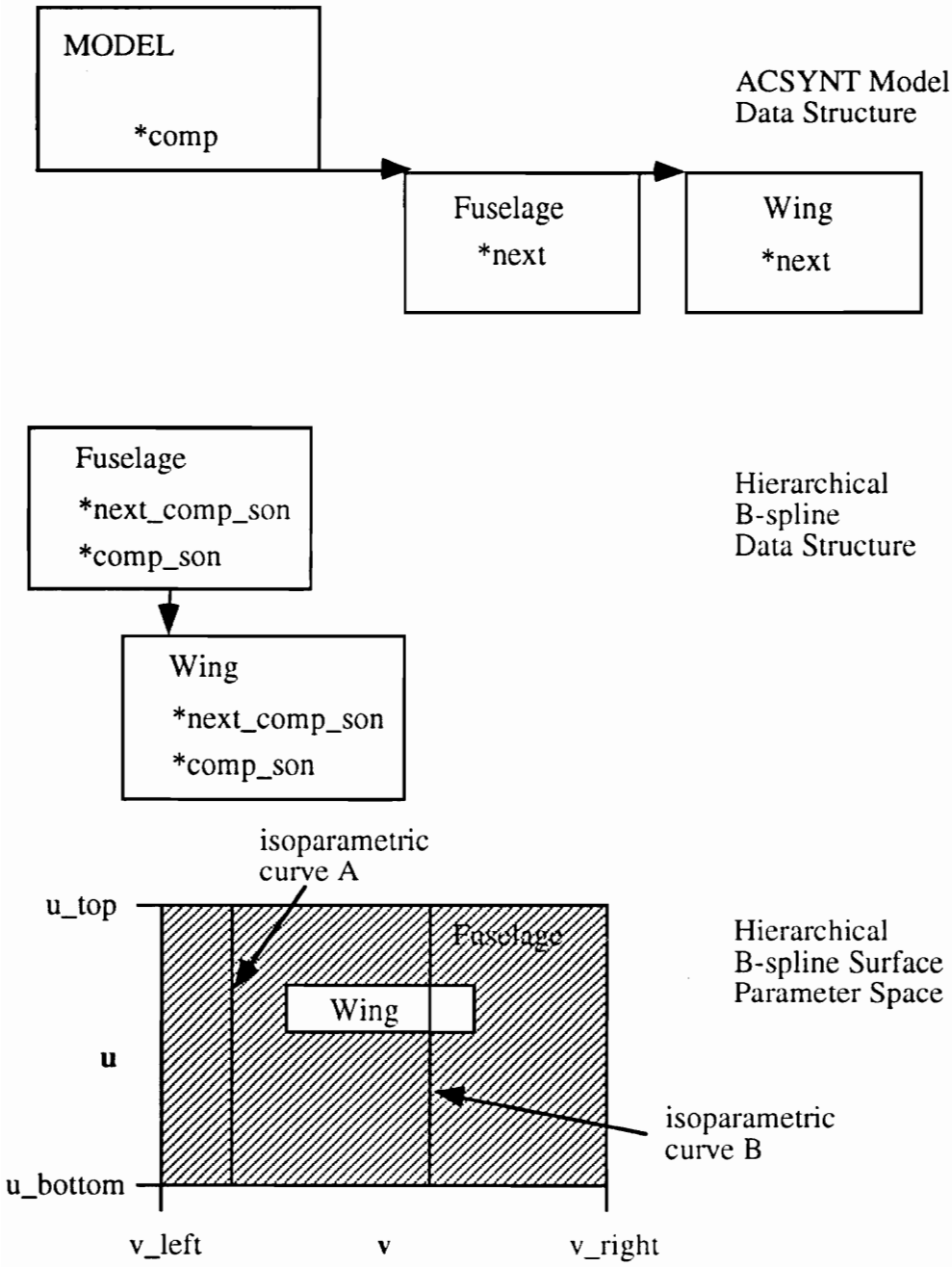



**Figure 23. Adding a Wing to the Fuselage in the Hierarchical Data Structure**

*Begin Aircraft Skinning*

*Read ACSYNT Aircraft File*

*Compute Fuselage Component from Nose, Mid-section and Aft-body components*

*Do Until all Components Added*

*Begin Compute Refinement Parameter Space*

*Compute Bounding Lines of Wing Component Type*

*Do Until Convergence of Maximum Iterations*

*Compute Distance Between Line and Fuselage Surface Pt*

*Compute Next Distance Between Line and Fuselage Pt*

*Perform Convergence Test*

*End Do Until Convergence or Maximum Iterations*

*End Compute Refinement Parameter Space*

*Refine Fuselage Surface*

*Modify Wing Type Component for Control Point Movement*

*Move Fuselage Control Points*

*Remove Redundant Surface Information*

*Update Hierarchical data Structure*

*End Do Until all Components Added*

*End Aircraft Skinning*

**Figure 24. Procedural Steps in Skinning an Aircraft**

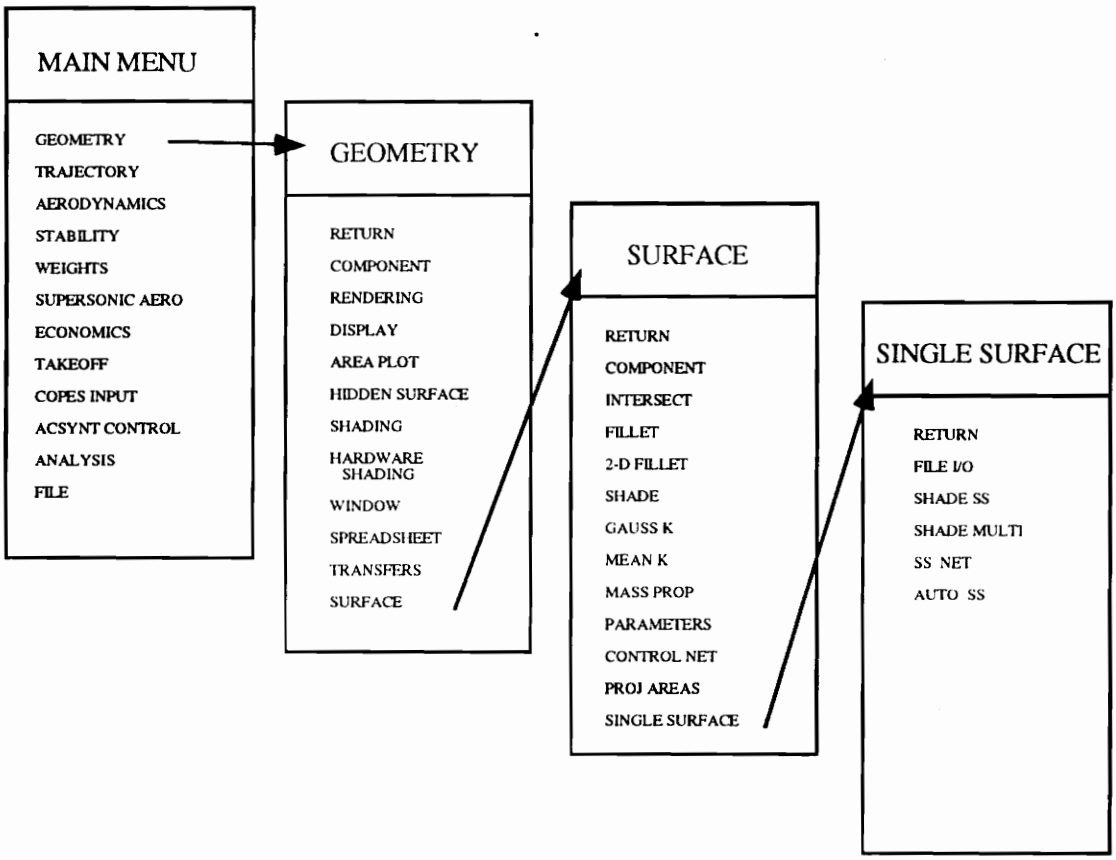



**Figure 24. Menu Hierarchy**

through the wing's parameter space. Figure 26 shows a wing/plane non-uniform hierarchical B-spline surface rendered with hierarchical isoparametric curves.

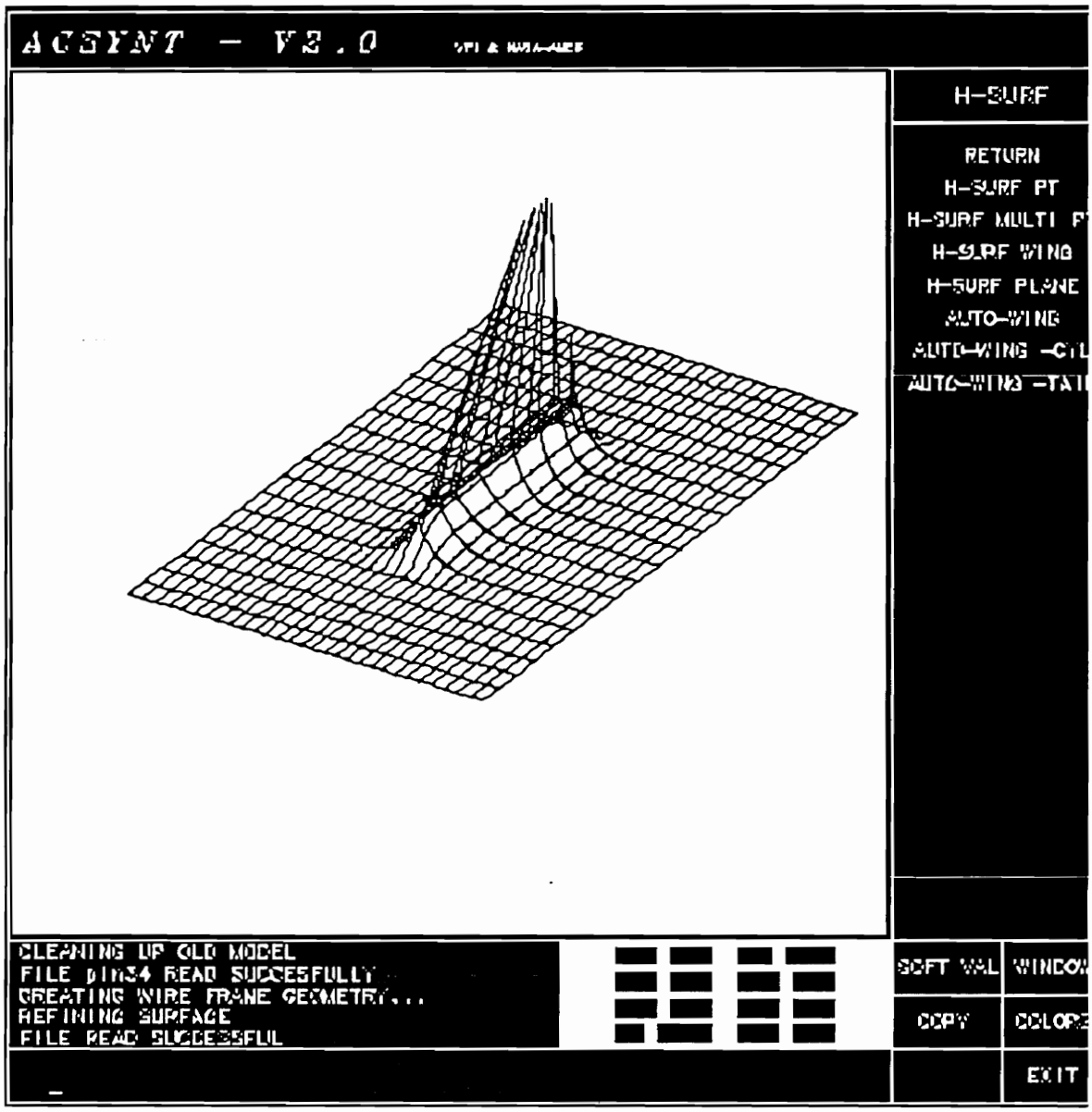



**Figure 26. Hierarchical Isoparametric Curves**

# 7.0 Results and Conclusions

The central development and result of this thesis research is the non-uniform hierarchical B-spline surface with its associated algorithms and data structures. These hierarchical surfaces localize the affected region of the control net when a component is added to a surface, reduce the amount of data stored in a single, $C^2$ B-spline surface, and provide a single *u, v* space for the entire surface skin. These qualities make non-uniform hierarchical B-spline surfaces useful in the engineering domain by providing the needed geometry for preliminary analysis in the conceptual phase. These algorithms have been incorporated into ACSYNT to further validate the methodology. In ACSYNT the non-uniform hierarchical B-spline surfaces provide a $C^2$ continuous surface of an aircraft skin. By integrating these surfaces with the ACSYNT data structures, the skin of a conceptual aircraft can be a "single-button" operation. Figures 27 and 28 show conceptual airplanes with differing wing parameters before and after skinning. These results demonstrate the utility of a non-uniform hierarchical B-spline surfaces in the mechanical engineering surface design domain.

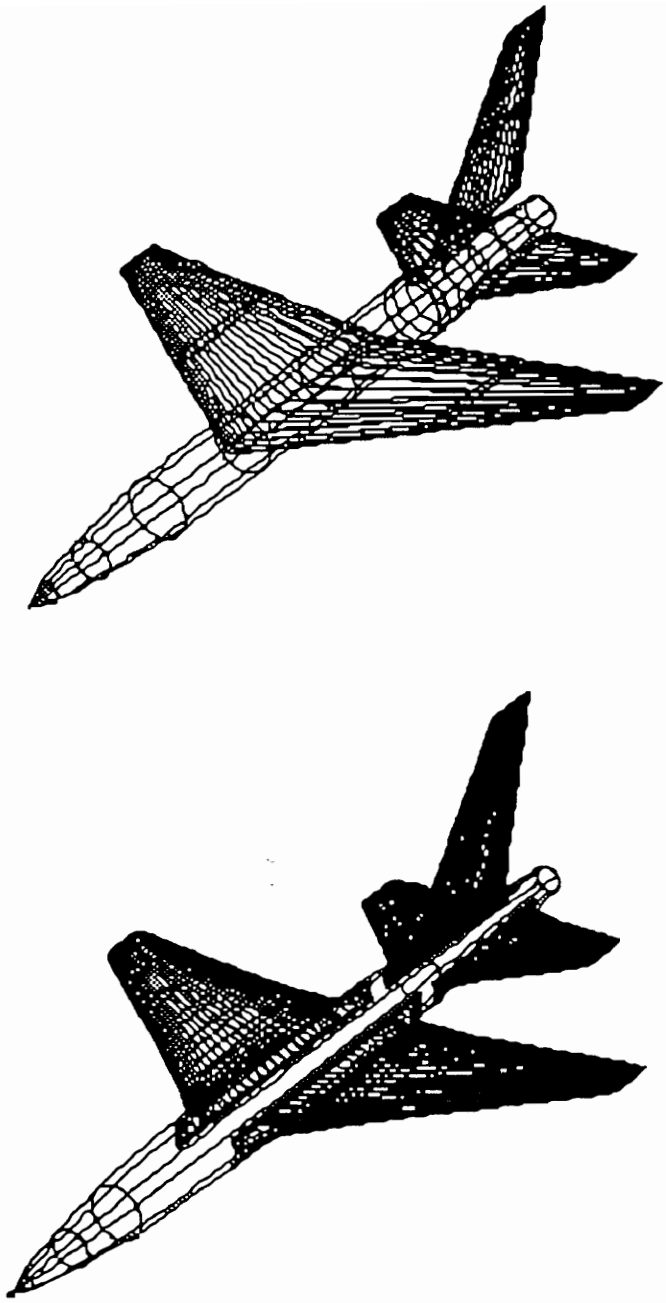


**Figure 27. Airplane I Before and After Skinning**

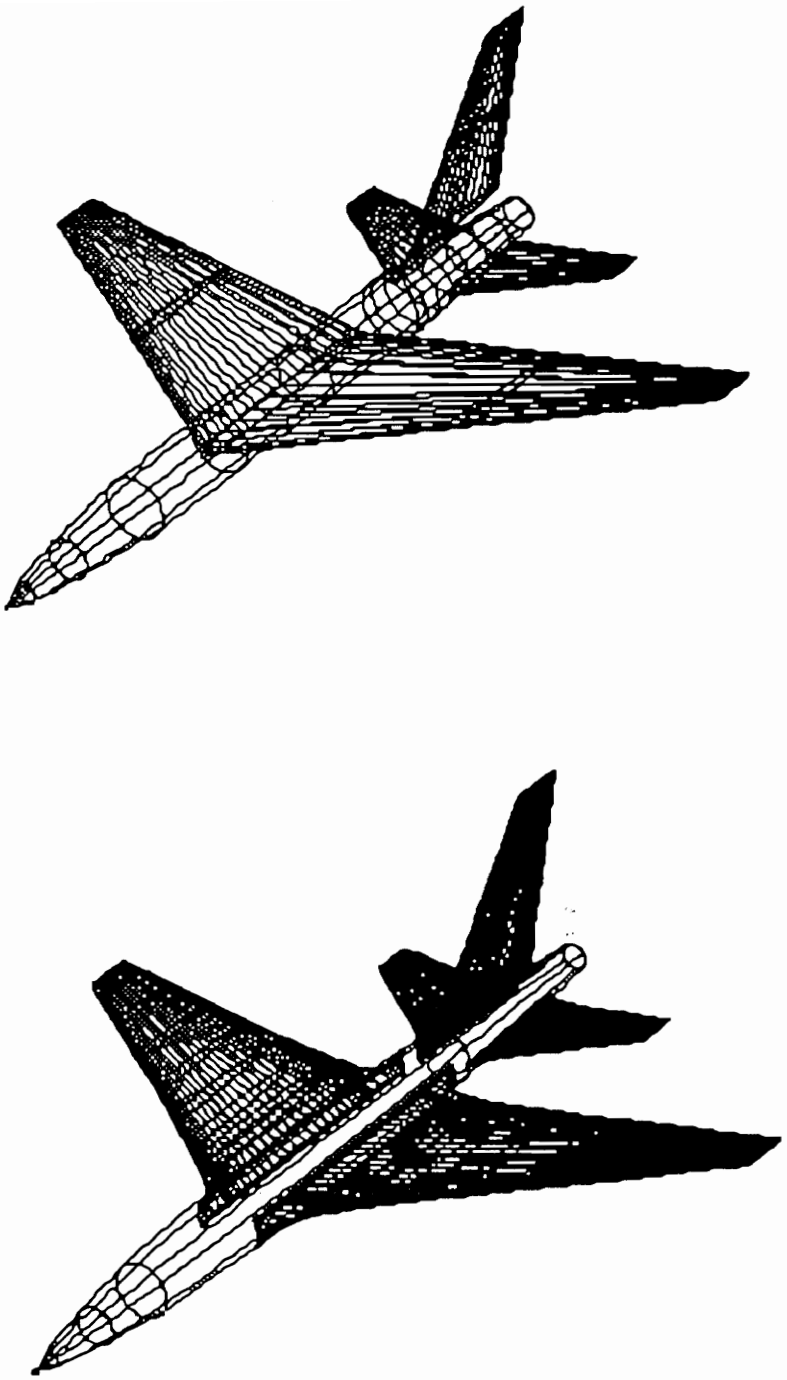


**Figure 28. Airplane II Before and After Skinning**

The development and implementation of non-uniform, hierarchical B-spline surfaces may be viewed from the mechanical engineer's perspective to observe the role that this surface form will have in the CAD field. First, the non-uniform, hierarchical B-spline surface offers a new surface geometry which may be employed in conceptual, preliminary, and final design phases. The ability to add both small and large component surfaces to an existing surface during any design phase ensures the utility of this surface in the CAD field. The hierarchical nature of the surface fits naturally with the CAD design process. The designer begins with a general surface design and adds more specific surface components at each level of hierarchy. If the designer wishes to change a surface component, then that particular hierarchical level is simply eliminated. Finally, the non-uniform knot sequences provide the needed control parameters for designing engineering surfaces. In short, the development and introduction of non-uniform, hierarchical B-spline surfaces into the CAD field significantly augments the mechanical engineer's ability to design engineering surfaces.

# 8.0 References


**[Acsy93]** "ACSYNT V2.0 Installation Manual and User Guide", *ACSYNT Institute / Computer Aided Design Laboratory, VPI&SU*, January 1993.

**[Barn90a]** Barnhill, R. E., Kersey, S. N., "A Marching Method for Parametric Surface/Surface Intersection", *Computer Aided Geometric Design*, vol. 7, no. 1-4, June 1990, pp. 257 - 280.

**[Barn90b]** Barnhill, R. E., Ou, H. S., "Surfaces Defined on Surfaces", *Computer Aided Geometric Design*, vol. 7, no. 1-4, June 1990, pp. 323 - 336.

**[Bart87]** Bartels, R., Beatty, J., Barsky, B., "An Introduction to Splines for Use in Computer Graphics & Geometric Modeling", *Morgan Kaufman Publishers, Inc.*, Los Altos, California, 1987.

**[Böhm84]** Böhm, W., Farin, G., Kahmann, J., "A Survey of Curve and Surface Methods in CAGD", *Computer Aided Geometric Design*, vol. 1, no. 1, July 1984, pp. 1 - 60.

**[Chen91]** Cheng, F., Barsky, B. A., "Interapproximation: interpolation and approximation using cubic spline curves", *Computer-Aided Design*, vol. 23, no. 10, December 1991, pp. 700-706.

**[deBo72]** de Boor, C., "On Calculating With B-Splines", *Journal of Approximation Theory*, vol. 6, no. 1, July 1972, pp. 50 - 62.

**[deBo78]** de Boor, C., " Practical Guide to Splines", Springer, 1978.

**[Fari90]** Farin, G., "Curves and Surfaces for Computer Aided Geometric Design - 2nd ed.", *Academic Press*, San Diego, California, 1990.

**[Faux79]** Faux, I. D., Pratt, M. J., "Computational Geometry for Design and Manufacture", *John Wiley & Sons*, New York, 1979.

**[Ferg64]** Ferguson, J., "Multivarible curve interpolation", *JACM*, vol. 2, no. 2, 1964, pp. 221-228.

**[Fili89]** Filip, D. J., Ball, T. J., "Procedurally Representing Lofted Surfaces", *IEEE CG&A*, November 1989, pp 27 -33.

**[Flem91]** Fleming, S., Myklebust, A., "Utilizing the graPHIGS API for CAD Applications", Proceedings of the Second International graPHIGS User's Group Conference and Workshop, Blacksburg, Virginia, October 20-23, 1991, pp. 3-10.

**[Flem92a]** Fleming, S., "The Enhancement of PHIGS Plus B-spline Functionality for Geometric Modeling in CAD", *M.S. Thesis,* VPI&SU, April 1992.

**[Flem92b]** Fleming, S., Myklebust, A., "The Enhancement of PHIGS Plus B-spline Functionality for Geometric Modeling", Fourth IFIP WG5.2 Workshop on Geometric Modeling in Computer Aided Design, Rensselaerville, New York, September 27-October 1, 1992.

**[Fole90]** Foley, J., van Dam, A., Feiner, S., Hughes, J., "Computer Graphics: Principles and Practice - 2nd ed.", *Addison-Wesley Publishing Company*, Reading, Massachusetts, 1990.

**[Fors88]** Forsey, D., Bartels, R. H., " Hierarchical B-Spline Refinement", *Computer Graphics*, vol. 22, no. 4, August 1988, pp. 205-212.

**[Glou89]** Gloudemans, J. R., "Filleting of Aircraft Components Using Non-Uniform B-Spline Surfaces", *M.S. Thesis,* VPI&SU, May 1989.

**[Glou90]** Gloudemans, J. R., Myklebust, A., "Filleting of Aircraft Components Using Non-Uniform B-Spline Surfaces", presented at *the International Federation for Information Processing WG 5.2 Workshop on Geometric Modeling*, Rensselaerville, New York, June 17-21, 1990.

**[Hohm91]** Hohmeyer, M. E., Barsky, B. A., "Skinning Rational B-Spline Curves to Construct and Interpolatory Surface", *CVGIP: Graphical and Image Processing*, vol. 53, no.6, November 1991, pp. 511-521.

**[Jaya91]** Jayaram, U., "Extracting Dimensional Geometric Parameters from B-Spline Surface Models", *Ph.D. Dissertation*, VPI&SU, November 1991.

**[Jaya92a]** Jayaram, S., Myklebust, A., Gelhausen, P., "ACSYNT - A Standards-Based System for Parametric Computer Aided Conceptual Design of Aircraft", *American Institute of Aeronautics and Astronautics,* AIAA 92-1268, *1992 Aerospace Design Conference*, February 3-6, 1992 / Irvine, CA.

**[Jaya92b]** Jayaram, U., Myklebust, A., Gelhausen, P., "Extracting Dimensional Geometric Parameters from B-Spline Surface Models of Aircraft", *American Institute of Aeronautics and Astronautics,* AIAA 92-4283, *AIAA Aircraft Design Systems Meeting*, August 24-26, 1992 / Hilton Head, SC.

**[Jone91]** Jones, R., "Intersection and Filleting of Non-Uniform B-Spline Surfaces", *M.S. Thesis,* VPI&SU, January 1991.

**[Kank91]** Kankainen, P., Abramian, M., Bedi, S., " Surface Modelling of a Fighter Aircraft Fuselage with B-Spline Skeletal Lines", *Canadian Aeronautics and Space Journal*, vol. 17, no. 1, March 1991, pp 35-41.

**[Koch84]** Kochanek, D. H. U., Bartels, R. H., " Interpolating Splines with Local Tension, Continuity and Bias Control", *Computer Graphics*, vol. 18, no. 3, July 1985, pp. 33-41.

**[Mort85]** Mortenson, M. E., "Geometric Modeling", *John Wiley & Sons*, New York, 1985.

**[Pegn90]** Pegna, J., Wolter, F., "Designing and Mapping Trimming Curves on Surfaces Using Orthogonal Projection", presented at *the ASME Design Technical Conferences - 16th Design Automation Conference*, Chicago, Illinois, September 16-19, 1990. Published by *ASME Design Engineering Division (Publication) DE* vol 23, pt. 1, 1990, pp. 235 - 245.

**[Pieg89]** Piegl, L., "Key Developments in Computer-Aided Geometric Design", *CAD*, vol. 21, no, 5, June 1989, pp. 262 - 274.

**[Rein90]** Reinig, K.D., "Rapid Radar Backscatter Simulation of Detailed Targets", *IEEE Transactions on Aerospace and Electronic systems""*, vol. 26, no. 5, September 1990, pp. 858-865.

**[Sarr84]** Sarraga, R. F., Waters, W. C., " Free-Form Surfaces in GMSolid: Goals and Issues", *IEEE CG&A*, January, 1985, pp21-40.

**[Till83]** Tiller, W., "Rational B-Splines for Curve and Surface Representation", *IEEE CG&A*, Septrember 1983, pp 61-69.

**[Tjun93]** Tjung, Jie Wen, "Projection, Design and Representation of Curves on B-spline Surfaces", *M.S. Thesis,* VPI&SU, January 1993.

**[Wamp88]** Wampler, S. G., Myklebust, A., Jayaram, S., Gelhausen, P., "Improving Aircraft Conceptual Design - A PHIGS Interactive Graphics Interface for ACSYNT", *American Institute of Aeronautics and Astronautics,* AIAA 88-4481, *AIAA Aircraft Design, Systems, and Operations Meeting*, September 7-9, 1988, Atlanta, Georgia.

**[Wamp88]** Wampler, S., "Development of a CAD system fo Automated Conceptual Design of Supersonic Aircraft", *M.S. Thesis,* VPI&SU, May 1988.

**[Wood88]** Woodward, C. D., "Skinning techniques for Interactive B-spline surface Interpolation", *Computer -Aided Design,* vol. 20, no. 8, October 1988, pp 441-450.

**[Wong90]** Wong, C. K., "Intersection of B-Spline Surfaces By Elimination Method", *M.S. Thesis,* VPI&SU, September 1990.

**[Wong91]** Wong, C. K., Feustel, C. D., Myklebust, A., "Intersection of Bicubic B-Spline Surfaces by Eliminants", presented at *International Conference on Curves, Surfaces, CAGD, and Image Processing*, Biri, Norway, June 20-25, 1991.

**[Yama88]** Yamaguchi, F., "Curves and Surfaces in Computer Aided Geometric Design", *Springer-Verlag*, Berlin, 1988.

**[Zeid91]** Zeid, I., "CAD/CAM Theory and Practice", *McGraw-Hill, Inc.*, New York, 1991.

# Appendix A. ACSYNT Hierarchical Surface User's Guide

## A.1 *Introduction*

This user's guide is designed to lead the ACSYNT user through the software developed for hierarchical surface creation. The step-by-step instructions provide clear examples to illustrate hierarchical B-splines. This user's guide assumes that the user is familiar with the ACSYNT software and has gone through the examples in ACSYNT V2.0 Installation Guide and User Guide [Acsy93].

## A.2 *"Point" Feature Creation*

1. Traverse the ACSYNT menu path to the B-spline (SURFACE) module.

2. Select **REFINE** to enter the hierarchical B-spline module.

3. Select **FILE I/O** to read in an ACSYNT B-spline file.

4. Select **READ NURBS**.

5. Select **B-SPLINE** to determine the file type to be read.

6. Type in **PLN34** as the name of the file to be read.

7. Select **RETURN.**

8. Select either **OLD REFINE** or **NEW REFINE.** Old refine illustrates the traditional method of refining a surface. New refine shows the hierarchical surface. To see the control nets, select **RETURN, CONTROL HULL**.

9. Select **FILE I/O** and read in the same file.

10. Select **H-SURF** ( Hierarchical surface ).

11. Select either **H-SURF PT** for a single point or **H-SURF MULTI PT** for three points. These features are created and rendered with hierarchical B-spline surfaces.

12. Select **RETURN** so that a new file may be read.

## A.3 *Box Component Creation*

1. Traverse the ACSYNT menu path to the B-spline (SURFACE) module.

2. Select **REFINE** to enter the hierarchical B-spline module.

3. Select **FILE I/O** to read in an ACSYNT B-spline file.

4. Select **READ NURBS**.

5. Select **B-SPLINE** to determine the file type to be read.

6. Type in **PLN14** as the name of the file to be read.

7. Select **RETURN.**

8. Select either **OLD BOX REFINE** or **NEW BOX REFINE.** Old box refine uses the traditional methods to add the box component. New box refine uses the hierarchical surfaces to add the component.

## A.4 *Wing Component Surface*

1. The wing component can be created and modified in ACSYNT. The wing component should be modeled with twenty-nine points per cross-section and should lie outside the control net of the base surface ( in this case the fuselage ). Once defined, select **SURFACE** to enter the B-spline module.

2. Select **REFINE** to enter the hierarchical surface module.

3. Select **FILE I/O** to write out the wing component B-spline file.

4. Select **WRITE NURBS** to write the file.

5. Select **B-SPLINE** as the file type to write.

6. Type in a name for the wing component, e.g. **WING.**

7. Select **READ NURBS** to read in the fuselage file.

8. Select **B-SPLINE** as the file type to be read.

9. Type in **try2b.show** as the filename.

10. Select **RETURN.**

11. Select **H-SURF.**

12. Select **AUTOWING-CYL** to add the wing component to the fuselage

13. Type **WING** , ( the name of the wing component file ) for the filename.

14. Type **WING-1** for component name.

Steps 12 - 14 can be repeated to add other wing components. At this time files must be created for each component to be added. To add the aircraft tail, use AUTOWING-TAIL instead of AUTOWING-CYL.

# Appendix B -Source Code Functional Descriptions

## B.1 *Functional Descriptions*

The following sections contain descriptions of the source code functions developed for this thesis. The function name is given followed by the basic purpose of the function. Next an exact description of the function call is listed along with input arguments, output arguments and the function's return output.

## B1.1 *add_u_list_node*

**PURPOSE:** This function is used in creating a linked list of $u,v$ points for a $u$ isoparametric curve. These points represent the boundaries of the different hierarchical surfaces that are crossed when rendering the isoparametric curve.

**DESCRIPTION:**

KNOT_LIST *add_u_list_node(COMP_NODE *rp, KNOT_LIST *start, float v)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *rp | pointer to the hierarchical surface data structure node |
| *start | pointer to the beginning of the list |
| v | parameter value |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

Returns a pointer to the beginning of the list

## B1.2 *add_v_list_node*

**PURPOSE:** This function is used in creating a linked list of $u,v$ points for a $v$ isoparametric curve. These points represent the boundaries of the different hierarchical surfaces that are crossed when rendering the isoparametric curve.

**DESCRIPTION:**

KNOT_LIST *add_v_list_node(COMP_NODE *rp, KNOT_LIST *start, float u)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *rp | pointer to the hierarchical surface data structure node |
| *start | pointer to the beginning of the list |
| u | parameter value |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

Returns a pointer to the beginning of the list

## B1.3 *add_comp_son*

**PURPOSE:** This recursive function is used in creating the hierarchical data structure. It adds a component to the hierarchical data structure.

**DESCRIPTION:**

COMP_NODE *add_comp_son(COMP_NODE *rp, COMP_NODE *temp)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *rp | pointer to current hierarchical surface data structure node |
| *temp | pointer to the node to be added to the data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

Returns a pointer to the node where the component is added

## B1.4 *addnode*

**PURPOSE:** This function begins the process of adding a component node to the hierarchical data structure.

**DESCRIPTION:**

COMP_NODE *addnode(COMP_NODE *root, comp_data *comp)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *root | pointer to root of the hierarchical surface data structure |
| *comp | pointer to the component data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

Returns a pointer to the root node of the hierarchical surface data structure

## B1.5 *box_refine_surface_old*

**PURPOSE:** This function refines and adds a box feature to the Model. It uses traditional methods of surface modeling.

**DESCRIPTION:**

int box_refine_surface_old(MODEL * Model)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to the ACSYNT model data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

## B1.6 *box_refine_surface_new*

**PURPOSE:** This function refines and adds a box feature to the Model. It uses hierarchical surface design methods to add the box component.

**DESCRIPTION:**

int box_refine_surface_new(MODEL * Model)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to the ACSYNT model data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

## B1.7 *copy_comp*

**PURPOSE:** This function copies a component.

**DESCRIPTION:**

comp_data *copy_comp(comp_data *comp)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *comp | pointer to the component data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

Returns a pointer to the copy of the component

## B1.8 *draw_HI_nubs*

**PURPOSE:** This function draws isoparametric curves for a hierarchical surface.

**DESCRIPTION:**

void draw_HI_nubs(MODEL *Model, COMP_NODE * root)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to the ACSYNT model data structure |
| *root | pointer to the root node of the hierarchical surface |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

## B1.9 *draw_Hu_nubs*

**PURPOSE:** This function draws the *u* isoparametric curves for a hierarchical surface.

**DESCRIPTION:**

void draw_Hu_nubs(MODEL *Model, COMP_NODE * root , float u)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to the ACSYNT model data structure |
| *root | pointer to the root node of the hierarchical surface |
| u | isoparametric value |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

## B1.10 *draw_Hw_nubs*

**PURPOSE:** This function draws the $v$ isoparametric line for a hierarchical surface.

**DESCRIPTION:**

void draw_Hw_nubs(MODEL *Model, COMP_NODE * root , float w)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to the ACSYNT model data structure |
| *root | pointer to the root node of the hierarchical surface |
| w | isoparametric value |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

## B1.11 *draw_H_bspline*

**PURPOSE:** This is the main function for rendering the hierarchical surfaces. This function is called when the programmer wants to display a hierarchical surface.

**DESCRIPTION:**

void draw_H_bspline(MODEL *Model, COMP_NODE * root )

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to the ACSYNT model data structure |
| *root | pointer to the root node of the hierarchical surface |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

## B1.12 *hsurf_autowing_cyl*

**PURPOSE:** This is the main function for adding a wing component to the fuselage. The first component in the Model component list is the fuselage.

**DESCRIPTION:**

int hsurf_autowing_cyl(MODEL *Model)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to the ACSYNT model data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

## B1.13 *hsurf_autowing_tail*

**PURPOSE:** This is the main function for adding a tail (wing) component to the fuselage. The first component in the Model component list is the fuselage.

**DESCRIPTION:**

int hsurf_autowing_tail(MODEL *Model)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to the ACSYNT model data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

## B1.14 *hsurf_multi_pt*

**PURPOSE:** This function creates three hierarchical point features on a surface.

**DESCRIPTION:**

int hsurf_multi_pt(MODEL *Model)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to the ACSYNT model data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

## B1.15 *hsurf_pt*

**PURPOSE:** This function creates a single hierarchical point feature on a surface.

**DESCRIPTION:**

int hsurf_pt(MODEL *Model)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to the ACSYNT model data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

## B1.16 *load_HSURF_menu*

**PURPOSE:** This function loads the hierarchical surface menu.

**DESCRIPTION:**

int load_HSURF_menu(MODEL *Model)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to the ACSYNT model data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

## B1.17 *load_hsurf_data_structure*

**PURPOSE:** This function loads the hierarchical surface data structure.

**DESCRIPTION:**

COMP_NODE *load_hsurf_data_structure(MODEL *Model)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to the ACSYNT model data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

Returns a pointer to the root node of the hierarchical surface data structure

## B1.18 *load_refinement_menu*

**PURPOSE:** This function loads the refinement menu.

**DESCRIPTION:**

int load_refinement_menu(MODEL *Model)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to the ACSYNT model data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

## B1.19 *load_NURBS*

**PURPOSE:** This function loads the NURBS data structure (shading routines).

**DESCRIPTION:**

int load_NURBS(MODEL *Model)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to the ACSYNT model data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

## B1.20 *make_u_knot_list*

**PURPOSE:** This recursive function adds a node to linked list of u knots.

**DESCRIPTION:**

KNOT_LIST *make_u_knot_list(COMP_NODE *rp, KNOT_LIST *start, float v)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *rp | pointer a hierarchical data structure node |
| *start | pointer to the beginning of the u knot list |
| v | parametric value |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

Returns pointer to the beginning of the u knot list

## B1.21 *make_v_knot_list*

**PURPOSE:** This recursive function adds a node to linked list of v knots.

**DESCRIPTION:**

KNOT_LIST *make_v_knot_list(COMP_NODE *rp, KNOT_LIST *start, float u)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *rp | pointer a hierarchical data structure node |
| *start | pointer to the beginning of the v knot list |
| u | parametric value |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

Returns pointer to the beginning of the v knot list

## B1.22 *make_u_list_node*

**PURPOSE:** This function makes a node and assigns its values.

**DESCRIPTION:**

KNOT_LIST *make_u_list_node(COMP_NODE *root)

**INPUT ARGUMENTS:**

*root pointer to the beginning of the u knot list data structure

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

Returns pointer to the new u knot list node

## B1.23 *make_v_list_node*

**PURPOSE:** This function makes a node and assigns its values.

**DESCRIPTION:**

KNOT_LIST *make_v_list_node(COMP_NODE *root)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *root | pointer to the beginning of the u knot list data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

Returns pointer to the new v knot list node

## B1.24 *mknode*

**PURPOSE:** This function makes a hierarchical data structure node and assigns its values.

**DESCRIPTION:**

COMP_NODE *mknode(comp_data *comp)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *comp | pointer to the component data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

Returns pointer to the new hierarchical surface node

## B1.25 *read_in_comp*

**PURPOSE:** This function reads in one component from an ACSYNT B-spline file.

**DESCRIPTION:**

comp_data *read_in_comp(void)

**INPUT ARGUMENTS:**

NONE

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

Returns pointer to the new component

## B1.26 *refine_surface_old*

**PURPOSE:** This function adds a point feature to a surface without hierarchical surfaces.

**DESCRIPTION:**

int refine_surface_old(MODEL *Model)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to ACSYNT model data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

## B1.27 *refine_surface_new*

**PURPOSE:** This function adds a point feature to a surface with hierarchical surfaces.

**DESCRIPTION:**

int refine_surface_new(MODEL *Model)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to ACSYNT model data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

## B1.28 *whereis_uv*

**PURPOSE:** This function determines which hierarchical surface is appropriate for a given *u,v* point.

**DESCRIPTION:**

COMP_NODE *whereis_uv(COMP_NODE *root, float u , float v)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *root | pointer to beginning of hierarchical surface data structure |
| u | parameter value |
| v | parameter value |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

Returns pointer to the hierarchical surface data structure node

## B1.29 *wing_refine_surface_new*

**PURPOSE:** This function adds a wing component using hierarchical surfaces

**DESCRIPTION:**

int wing_refine_surface_new(MODEL *Model)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to ACSYNT model data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

## B1.30 *wing_refine_surface_old*

**PURPOSE:** This function adds a wing component without using hierarchical surfaces

**DESCRIPTION:**

int wing_refine_surface_old(MODEL *Model)

**INPUT ARGUMENTS:**

| | |
|---|---|
| *Model | pointer to ACSYNT model data structure |

**OUTPUT ARGUMENTS:**

NONE

**FUNCTION OUTPUT:**

NONE

# Vita

The author was born on 31 August 1962 in Pasadena, Texas. He attended the University of Texas at Austin and received his B.S.M.E. in December, 1984. After working in the Department of Defense for six years, he returned to school at Virginia Tech to pursue a Master's degree in mechanical engineering. His future plans include finding a job in the CAD field, restoring an old farm house, playing with his daughters, and continuing to research the application of projective geometry to practical, computer aided design problems.