PHOTO-REALISTIC GRAPHICAL REPRESENTATION OF HELICAL CABLE STRUCTURE

A DISSERTATION SUBMITTED TO THE GRADUATE DIVISION OF THE UNIVERSITY OF HAWAI'I IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

MECHANICAL ENGINEERING

DECEMBER 2007

By
Chang Huang

Dissertation Committee:

Ronald H. Knapp, Chairperson
Weilin Qu
Anyuan Cao

We certify that we have read this dissertation and that, in our opinion, it is satisfactory in scope and quality as a dissertation for the degree of Master of Science in Mechanical Engineering.

DISSERTATION COMMITTEE

Chairperson

For my family, who offered me unconditional love and support throughout the course of this thesis.

Acknowledgements

First and foremost I would like to thank my thesis advisor Ronald, H, Knapp, who has shown a large and consistent interest in my project during the times. Our numerous scientific discussions and her many constructive comments have greatly improved this work.

I would also like to thank Dr. Anyuan Cao and Dr. Weilin Qu whose steadfast support of this project was greatly needed and deeply appreciated.

ABSTRACT

Cables and wire ropes frequently incorporate strands consisting of double and triple helical wires. Stranded conductors are helically wrapped into larger units that again are helically laid into cables. A structural analysis of such complex geometries requires knowledge of the net elongation, bending and twist of the individual wires in these strands. Equations that describe the centerlines of these wires and the wire surface areas are derived in this paper. These equations are useful in the development of structural models as well as accurate three-dimensional plots of the cable geometry.

Three parametric equations are used to describe the center line of the double and triple helical wires. A companion parametric equation, using two parameters, describes the double and triple helical wire surfaces. These equations are used to produce plots of the three-dimensional shape of individual helical wires and second, third degree helical cable. Such plots are useful in designing cables where geometrical interrelationships are easily understood.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

High strength wire ropes are highly efficient structures used to transmit tension along straight paths and around circular sheaves. They are widely used in mechanical and structural applications that require flexibility and strength. For Electrical-Optical-Mechanical (EOM) cables, helically-laid wires provide flexibility and provide external armor to protect electrical and optical core components. To model such cables for specific applications, computation of change of helical wire elongation, curvature and twist is needed to compute stress magnitudes in the helical wires and to produce deformed plots. In addition to using the helical equations for stress analysis, industry has expressed a need for realistic solid models of cable constructions to help in packaging components, use of color coding and to convey new concepts to design team members and customers. Moreover, such geometrical detail will help with calculation of cable weight and cost.

Single, double and triple helical cable geometries are developed as parametric equations that describe wire centerlines and their surfaces. Single, double and triple helical parameters, u_s,u_d and u_t, are used to formulate all three parametric relations. In this study, a Graphics Library - OpenGL is used to demonstrate 3D solid cable models. Examples that demonstrate cable modeling are given.

1.1 Description of Cable Structures

A wire rope is an assembly of round wires that are helically served around a core which results in a flexible metallic rope capable of resisting high tensile loads. A typical

wire rope is composed of a helical strand that is helically laid along a straight core wire as shown in Fig 1.1. The outer layer wires in this configuration form a double helix.

Figure 1.1: A typical double-helix rope with double-helical structure

Modern wire rope was invented by the German mining engineer Wilhelm Albert in the years between 1831 and 1834 for use in mining in the Harz Mountains in Clausthal, Lower Saxony, Germany. It was quickly accepted because it proved superior to ropes made of hemp or to metal chains, such as had been used in the past.

Wilhelm Albert's first ropes consisted of wires twisted about a hemp rope core, six such strands then being twisted around another hemp rope core in alternating directions for extra rotational stability. Earlier forms of wire rope had been made by covering a bundle of wires with hemp. In America wire rope was later manufactured by John Roebling, forming the basis for his success in suspension bridge building. Roebling introduced a number of innovations in the design, materials and manufacture of wire rope.

Manufacturing a wire rope is similar to making one from natural fibers. The individual wires first are twisted into a strand, then six or so such strands again twisted

around a core. This core may consist of steel, but also of natural fibres such as sisal, manila, henequen, jute, or hemp. This is used to radically cushion wires and thus reduce stress when bending the rope.

This flexibility is particularly vital in ropes used in machinery such as cranes or elevators as well as ropes used in transportation modes such as cable cars, cable railways, funiculars and aerial lifts. It is not quite so essential in suspension bridges and similar uses.

The lay of a wire rope describes the manner in which either the wires in a strand, or the strands in the rope, are laid in a helix. Left and right hand lay Left hand lay or right hand lay describe the manner in which the strands are laid to form the rope. To determine the lay of strains in the rope, look at the rope as it points away from you. If the strands appear to turn in a clockwise direction, or like a right-hand thread, as the strands get further away from you, then the rope is a right hand lay. The picture of steel wire rope on this page shows a rope with right hand lay. If the strands appear to turn in an anti-clockwise direction, or like a left-hand thread, as the strands get further away from you, then the rope is a left hand lay.

1.2 List of Symbols

Symbols used to develop the single, double and triple helical centerlines are defined below and also in Fig 2.5 and 2.6.

(R) **Pitch radius** The perpendicular distance between the centroidal axis of a strand and the cable axis.

(r) **Wire radius** The radius of an individual wire

(α) **Strand lay angle** The lay angle of the strand about the cable axis (single helix).

(β) **Substrand wire lay angle** The lay angle of a substrand about the strand axis (double helix).

(γ) **Wire lay angle** The lay angle of a wire about the substrand axis (triple helix).

(θ_s) **Position angle along strand axis** The angle measured from the global X-axis to a point on the strand axis.

(θ_d) **Position angle along substrand axis** An angle similar to (θ_s) to determine position of an individual substrand helically laid about the strand axis.

(θ_t) **Position angle along wire axis** An angle similar to (θ_s) to determine position of an individual wire helically laid about the substrand axis.

(λ) **Wire rotation direction** The wire rotate direction about its rotation axis. (left or right-hand lay)

(L) **Strand lay length** The pitch or lay length measured parallel to the cable axis around which the centroidal axis of a strand makes one complete helical revolution.

Chapter 2

Single, Double and Triple Helical Parametric Equations

2.1 Curve Representation

Since the topic of curves is quite important to this study, a brief introduction of to geometric curve descriptions is given below. The advantages and disadvantages of several methods of curve generation are given. Later we will extend some of these methods to helical curves.

Curve generating can be achieved in several ways. A curve can be described pointwise with a series of coordinate data or by using an analytic equation. The coordinate data method obviously is difficult to change shapes of current curves. The exact shape also is hard to know. Analytic equations of curves can provide more information, such as curve behavior, continuity and curvature, etc, and is easier to control the curve.

Curves are widely generated in computer graphics and CAD/CAM software using analytic geometry or approximation theories. Many applications in the automotive and aerospace industries require general curves to meet various shape requirements. Such curves can be described mathematically by nonparametric or parametric equations. For a nonparametric curve, the coordinates y and z are expressed by two separate functions, and x is the independent variable.

Nonparametric equations can be explicit or implicit. In three-dimensional space, explicit nonparametric equations usually are expressed as follows:

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ f(x) \\ g(x) \end{bmatrix} \tag{2.1.1}$$

where X is a three-dimensional curve. From Equation (2.1.1) we know that the x-coordinate of a point on a curve only has one corresponding y and z coordinate. Thus this form can't express multi-valued curves such as a circle. The implicit nonparametric representation can solve this problem, see Eq (2.1.2),

$$\begin{aligned} \mathrm{F(x,y,z)} &= 0 \\ \mathrm{G(x,y,z)} &= 0 \end{aligned} \tag{2.1.2}$$

but this representation still has some limitations. For example when the value of x is given, at some conditions calculation of the values of y and z is complex in finding roots. Secondly if the slope of a curve at a point is vertical or near vertical, its value could approach infinity or at least result in a large number. For this case it is difficult to obtain a solution. Another condition is the need for many computations when we want to display the curve through a series of points or short lines.

Parametric representation of curves can avoid the above-mentioned limitations. It also can handle closed and multiple-valued functions such as circles and conics, and it is easy to calculate the value of a point on the curve. When the slope of the curve is near vertical, we can use a tangent value for calculation. The important point is that the parametric equations are independent of the coordinate system. The equations determine the shape of an object through the relationship between its data points. So the parametric form is not only more general but also better suited to fitting for computations and display.

A parametric curve in three dimensional space (Fig. 2) is given by the parametric representation

$$X = X(u) = \begin{cases} x(u) \\ y(u) \\ z(u) \end{cases} , u \in [a, b] \tag{2.1.3}$$

The parametric form can have many advantages in geometric modeling. Points on the curve can be computed by choosing the proper parametric value in Eq (2.1.3). Geometrical transformations can be performed directly on parametric equations. In addition, common forms for curves which are extendable to a description of the surface can be found. Furthermore, curves defined by Eq (2.1.3) included initial and end value, there is not additional geometric data to define the boundary. Finally, the parametric equation is good for curve fitting on a display with special graphics hardware.

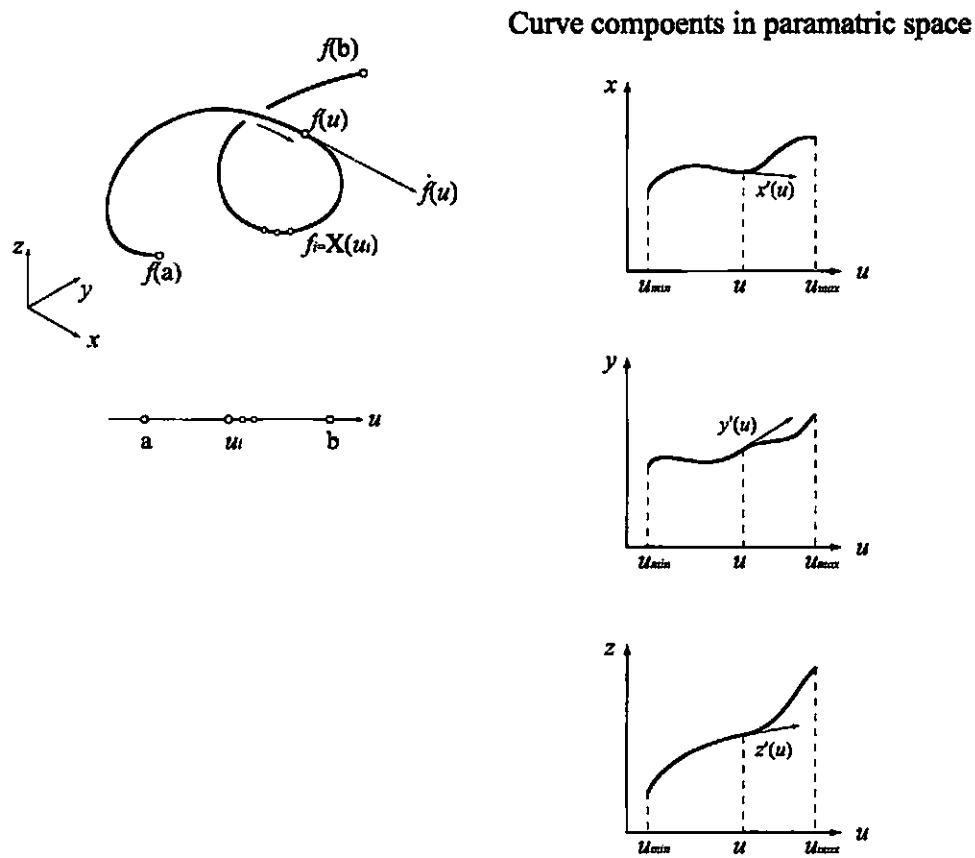


Figure 2.1: Parametric representation of a three-dimensional curve

To evaluate the slope of a parametric curve at an arbitrary point, the concept of the tangent vector is introduced. The Cartesian coordinates, x,y,z, are differentiable functions of the parameter, u. It is assumed further that

$$\dot{\mathrm{X}}(u) = \begin{bmatrix} \dot{x}(u) \\ \dot{y}(u) \\ \dot{z}(u) \end{bmatrix} \neq 0, u \in [a, b] \tag{2.1.4}$$

where derivatives are taken with respect to u. This condition avoids problems concerning the parametrization of the curve.

Arc length can be found from

$$s = s(u) = \int_a^u \left\| \dot{\mathrm{X}} \right\| du \tag{2.1.5}$$

A space curve X(t)=($x(u)$, $y(u)$,$z(u)$) is associated with an orthonormal coordinate frame called the Frenet frame given by the three vectors: the tangent vector $\vec{T}(u)$, the normal vector $\vec{N}(u)$ and the bionormal vector $\vec{B}(u)$ (Fig. 3).

We assume that the first three derivatives are linearly independent. Then $\dot{x}$, $\ddot{x}$,$\dddot{x}$ form a local affine coordinate system with position X. Through the Gram-Schmidt process of ortho-normalization this local affine coordinate system is converted to a Frenet frame system (Farin, 1990), as shown in Fig.(2.2) and given by Eqs. 4-6.

$$\vec{T} = \frac{\dot{x}}{\left\| \dot{\mathrm{X}}(u) \right\|}\vec{i} + \frac{\dot{y}}{\left\| \dot{\mathrm{X}}(u) \right\|}\vec{j} + \frac{\dot{z}}{\left\| \dot{\mathrm{X}}(u) \right\|}\vec{k} \tag{2.1.6}$$

$$\vec{B} = \frac{\dot{\mathrm{X}} \otimes \ddot{\mathrm{X}}}{\left\| \dot{\mathrm{X}} \otimes \ddot{\mathrm{X}} \right\|} \tag{2.1.7}$$

$$\vec{N} = \vec{B} \otimes \vec{T} \tag{2.1.8}$$

where

$$\left\| \dot{\mathrm{X}}(u) \right\| = \sqrt{\left(\frac{dx}{du}\right)^2 + \left(\frac{dy}{du}\right)^2 + \left(\frac{dz}{du}\right)^2}$$

$$\left\| \ddot{\mathrm{X}}(u) \right\| = \sqrt{\left(\frac{d^2x}{du^2}\right)^2 + \left(\frac{d^2y}{du^2}\right)^2 + \left(\frac{d^2z}{du^2}\right)^2}$$

and $\otimes$ denotes the cross product.

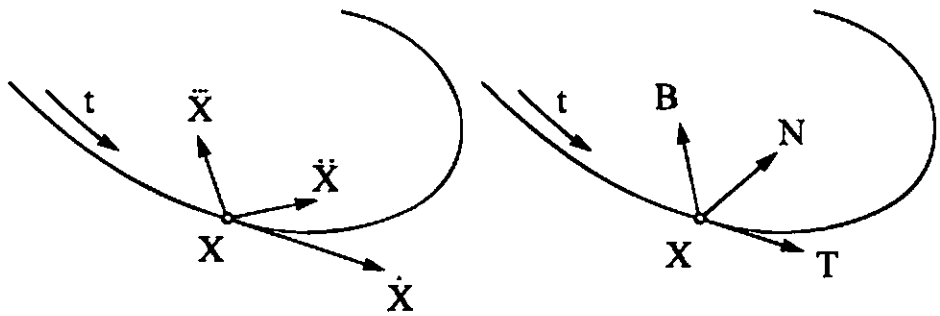


Figure 2.2: Local affine system (left) and Frenet frame (right).

2.2 Single Helical Parametric Equation

We first assume a simple single helix whose parametric variables include θ and R_s or θ and u_s. Other parametric variables that describe a starting position along a curve and lay direction will be added later. For a single helix (Elata, 2003) with an initial strand position angle of 0 at z = 0 and a right-hand lay direction shown in Fig.2.3, the vector equation of the centroidal axis is

$$\mathrm{X}(u) = \begin{cases} x = R_s \cos\theta = R_s \cos 2\pi u_s \\ y = R_s \sin\theta = R_s \sin 2\pi u_s \\ z = \theta R_s / \tan\alpha = 2\pi u_s R_s / \tan\alpha \end{cases} \tag{2.2.1}$$

The Cartesian coordinates, x,y and z of a single helical centerline also are differentiable functions of u_s. The subscript "s" indicates variables that are associated with a single helix. The coordinates of the helical centerline is lying on a circle of radius, R_s, with the helical lay angle, α.

As u_s varies from 0 to 1, the single helix makes exactly one turn in a right-hand lay (RHL) or left-hand lay (LHL) direction specified by λ for a RHL=1 and for a LHL= -1, see Fig (2.3). From Fig (2.4), λ affects the x and y coordinates, but does not change the z coordinate. Thus, Eq (2.2.2) can be rewritten as

$$\mathrm{X}(u_s) = \begin{cases} x = R_s \cos 2\pi\lambda u_s \\ y = R_s \sin 2\pi\lambda u_s \\ z = 2\pi u_s R_s / \tan\alpha \end{cases} \tag{2.2.2}$$

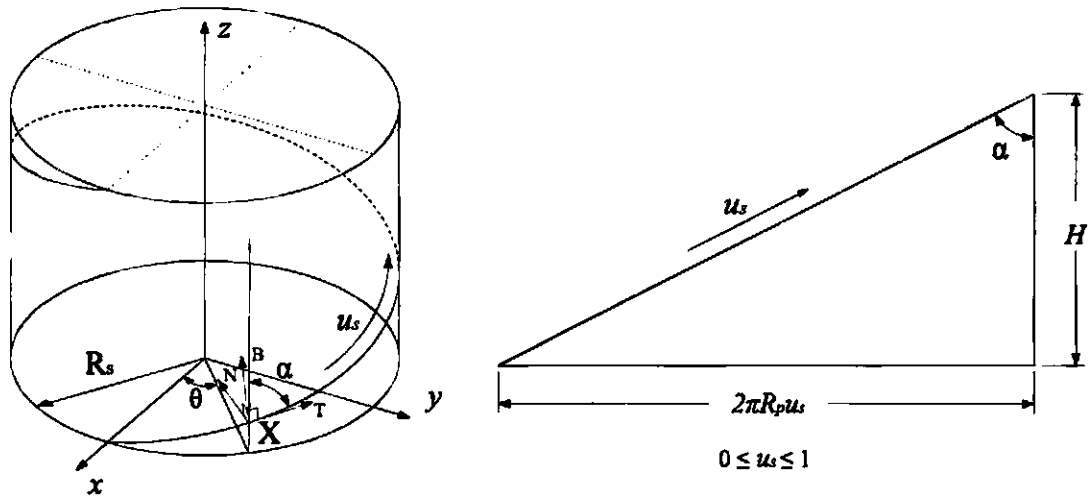


Figure 2.3: Global coordinates of single helix

For a helical curve that starts (z = 0) at a different position than that shown in Fig (2.4), a variable φ, is introduced as a new starting point. If we let the starting parameter, φ, vary from 0 to 1, the corresponding start angle is 0 2π. In this case the new coordinate position becomes

$$\mathrm{X}(u_s) = \begin{cases} x = R_s \cos[2\pi\lambda(u_s + \varphi)] \\ y = R_s \sin[2\pi\lambda(u_s + \varphi)] \\ z = 2\pi u_s R_s / \tan\alpha \end{cases} \tag{2.2.3}$$

For the Frenet frame system, the first and second derivatives of Eq (2.2.3) are needed. These are given by

$$\dot{\mathrm{X}}(u) = \begin{cases} \frac{dx}{du} = -2\pi\lambda R_s \sin[2\pi\lambda(u_s + \varphi)] \\ \frac{dy}{du} = 2\pi\lambda R_s \cos[2\pi\lambda(u_s + \varphi)] \\ \frac{dz}{du} = 2\pi R_s / \tan\alpha \end{cases} \tag{2.2.4}$$

$$\ddot{\mathrm{X}}(u) = \begin{cases} \frac{d^2x}{du^2} = -4\pi^2 R_s \cos[2\pi\lambda(u_s + \varphi)] \\ \frac{d^2y}{du^2} = -4\pi^2 R_s \sin[2\pi\lambda(u_s + \varphi)] \\ \frac{d^2z}{du^2} = 0 \end{cases} \tag{2.2.5}$$

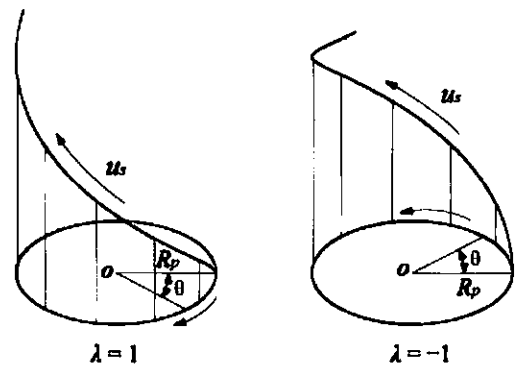


Figure 2.4: Single helix rotation direction.

Also,

$$\left\|\dot{\mathrm{X}} \otimes \ddot{\mathrm{X}}\right\| = \frac{8\pi^3 R_S^2}{\sin\alpha} \tag{2.2.6}$$

From Eqs. (2.1.7) and (2.1.8), we obtain the binormal and normal vectors,

$$\begin{aligned}\vec{B}_s = &-\sin[2\pi\lambda(u_s+\varphi)]\cos\alpha\vec{i} - \cos[2\pi\lambda(u_s+\varphi)]\cos\alpha\vec{j} \\ &+\lambda\sin\alpha\vec{k}\end{aligned} \tag{2.2.7}$$

$$\vec{N}_s = -\cos[2\pi\lambda(u_s+\varphi)]\vec{i} - \sin[2\pi\lambda(u_s+\varphi)]\vec{j} + 0\vec{k} \tag{2.2.8}$$

2.3 Relationships among Single, Double and Triple Helix

With the pitch radius, start angle, lay angle and direction, and the parameter, u_s , known, the wire centerlines for single, double and triple helices as shown in Fig (2.5) can be drawn. This requires that u_d and u_t be found in terms of u_s.

For circular helices, the strand (single helix), substrand (double helix) and wire (triple helix) centroidal axes can be developed into the planar geometry shown in Fig (2.6).

From this geometry,

$$\frac{2\pi R_t u_t}{L_t} = \sin\gamma \tag{2.3.1}$$

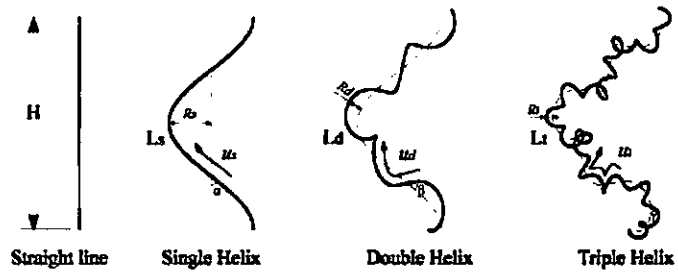


Figure 2.5: Single, double And triple helical centerlines.

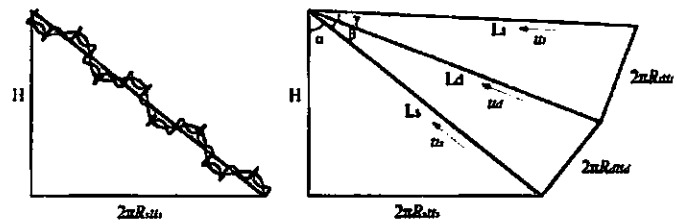


Figure 2.6: Developed helical geometries

$$\frac{2\pi R_d u_d}{L_d} = \sin\beta \tag{2.3.2}$$

$$\frac{2\pi R_s u_s}{L_s} = \sin\alpha \tag{2.3.3}$$

$$\frac{2\pi R_t u_t}{L_d} = \tan\gamma \tag{2.3.4}$$

$$\frac{2\pi R_d u_d}{L_s} = \tan\beta \tag{2.3.5}$$

Combining Eq (2.3.1) to (2.3.5), relations between u_d, u_t and u_s are obtained as follows:

$$u_d = \frac{\tan\beta R_s u_s}{\sin\alpha R_d} \tag{2.3.6}$$

$$u_t = \frac{\tan\gamma R_d u_d}{\sin\beta R_t} = \frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} \tag{2.3.7}$$

From Eqs (2.3.6) and (2.3.7), we find that position parameters for the double and triple helical centerlines depend on the lay angles, pitch radii and the position parameter for a single helix.

for the curve parameter, u, and for $v \in [0, 1)$. From Fig (2.8), the rotation at any position along the double-helical centerline is

$$\theta_d = 2\pi u_d = \frac{L_s \tan\beta}{R_d} \tag{2.3.8}$$

Eqs. (2.3.1) to (2.3.8) combine to give

$$\theta_d = \frac{2\pi \tan\beta R_s u_s}{\sin\alpha R_d} \tag{2.3.9}$$

From Fig (2.7), for $u_s > 0$, the position along L_d in local coordinates is $x' = R_d \cos\theta_d$ and $y' = R_d \sin\theta_d$,

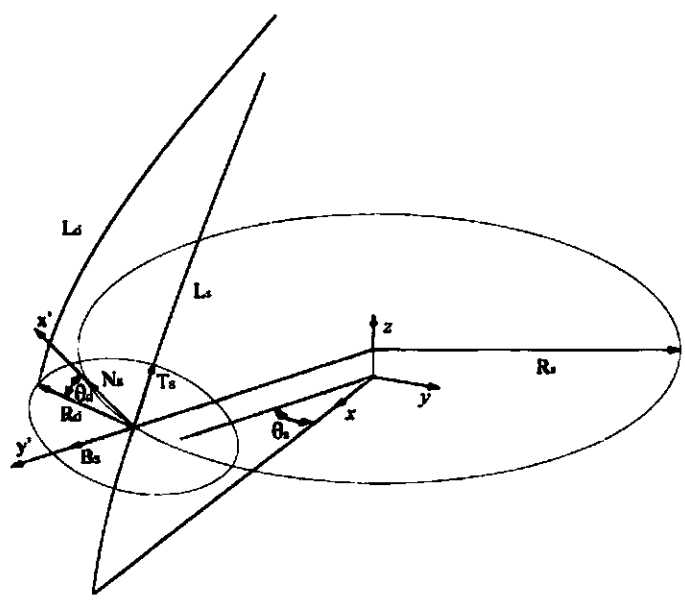


Figure 2.7: Frenet frame, T_s, N_s and B_s, for single helix

Based on the single helical centerline, Eq (2.2.10), the double helix centerline equation in global coordinates is

$$\vec{X}_d = \vec{X}_s + R_d \cos 2\pi u_d \vec{N}_s + R_d \sin 2\pi u_d \vec{B}_s \tag{2.3.10}$$

Shifting the starting point by an amount, φ_d , on the double helix, and including the lay direction parameter, λ_d, Eq (2.3.11) can be rewritten as

$$\begin{aligned}\vec{X}_d &= \vec{X}_s + R_d[\cos 2\pi\lambda_d(u_d + \varphi_d)]\vec{N}_s \\ &+R_d \sin[\cos 2\pi\lambda_d(u_d + \varphi_d)]\vec{B}_s\end{aligned} \tag{2.3.11}$$

Similarly, the triple helical centerline equation becomes

$$\begin{aligned}\vec{X}_t &= \vec{X}_d + R_t[\cos 2\pi\lambda_t(u_t + \varphi_t)]\vec{N}_d \\ &+R_t \sin[\cos 2\pi\lambda_t(u_t + \varphi_t)]\vec{B}_d\end{aligned} \tag{2.3.12}$$

Eq (2.3.6) is used to express Eqs. (2.3.11) and (2.3.12) in terms of u_s, u_d and u_t with the following results:

$$\begin{aligned}\vec{X}_d &= \vec{X}_s + R_d[\cos 2\pi\lambda_d(\tfrac{\tan\beta R_s u_s}{\sin\alpha R_d} + \varphi_d)]\vec{N}_s \\ &+R_d \sin[\cos 2\pi\lambda_d(\tfrac{\tan\beta R_s u_s}{\sin\alpha R_d} + \varphi_d)]\vec{B}_s\end{aligned} \tag{2.3.13}$$

$$\begin{aligned}\vec{X}_t &= \vec{X}_d + R_t[\cos 2\pi\lambda_t(\tfrac{\tan\gamma R_d u_d}{\sin\beta R_t} + \varphi_t)]\vec{N}_d \\ &+R_t \sin[\cos 2\pi\lambda_t(\tfrac{\tan\gamma R_d u_d}{\sin\beta R_t} + \varphi_t)]\vec{B}_d\end{aligned} \tag{2.3.14}$$

Eqs (2.3.7) and (2.3.14) yield the triple helical centerline coordinates in terms of u_s according to

$$\begin{aligned}\vec{X}_t &= \vec{X}_d + R_t[\cos 2\pi\lambda_t(\tfrac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t)]\vec{N}_d \\ &+R_t \sin[\cos 2\pi\lambda_t(\tfrac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t)]\vec{B}_d\end{aligned} \tag{2.3.15}$$

2.4 Surface Models

Surface representation is considered an extension of curve representation covered. The nonparametric and parametric forms of curves can be extended to surfaces. Similarly, the use of surface in computer graphics and CAD/CAM requires developing the proper equations and algorithms for both computation and programming purposes. Additionally, surface description is usually related to machining requirements to manufacture the surface.

Surface can be described mathematically in three-dimensional space by nonparametric or parametric equations. There are several methods to describe non-parametric surfaces to a given set of data points. These fall into two categories. The equation of the surface can be given by

$$S = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \\ f(x,y) \end{bmatrix} \tag{2.4.1}$$

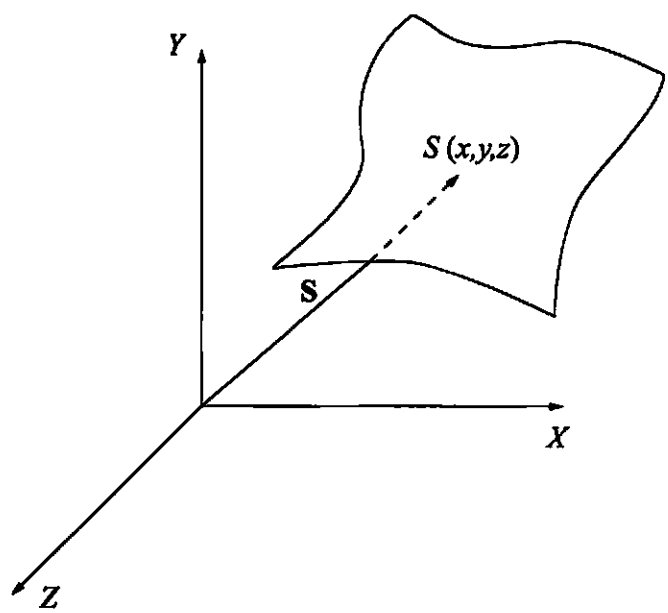


Figure 2.8: Point S on a nonparametric surface patch

where **S** is the position vector of a point on the surface as shown in Fig (2.8). Another form of the function $f(x,y)$ for a surface to pass through all the given data points is a polynomial,

$$z = f(x,y) = \sum_{m=0}^{p} \sum_{n=0}^{q} a_{mn} x^m y^n \tag{2.4.2}$$

where the surface is described by an XY grid of size $(p+1)(q+1)$ points. In this format the data points are used to develop a series of surface patches that are connected together with at least position and first-derivative continuity.

The nonparametric surface representation also have disadvantages when compared with parametric surface representations. These disadvantages are similar with those in nonparametric curves.

The parametric representation of a surface means a continues, vector valued function $S(u, v)$ of two variables, u and v. $S(u, v)$ assumes every position on the surface. The function $S(u, v)$ at certain u and v values is the point on the surface at these values. The most general way to describe the parametric equation of a three-dimensional curved surface in space is

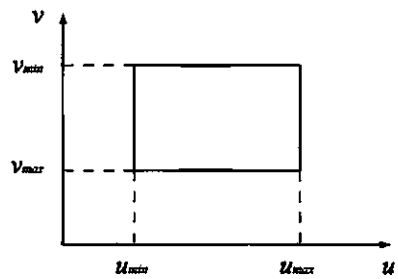
$$S = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{bmatrix}, u_{min} \leq u \leq u_{max}, v_{min} \leq v \leq v_{max} \tag{2.4.3}$$

Eq 2.4.3 gives the coordinates of a point on the surface as the components of its position vector. It converts the parametric space (E^2 in u and v values) to the cartesian space (E^3 in x,y and z) shown in Fig (2.9). The parametric variables u and v are constrained to intervals bounded by minimum and maximum values.

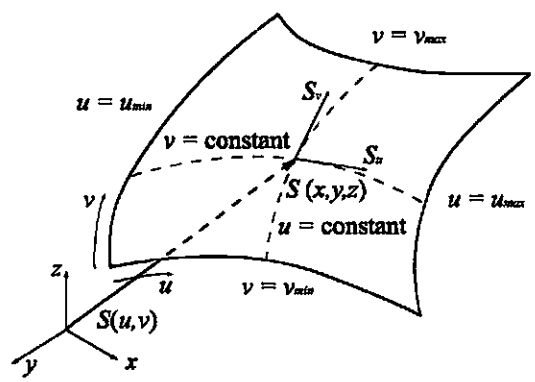
Eq 2.4.3 suggests that a general three-dimensional surface can be modeled by dividing it into an assembly of topological patches. A patch is considered the basic mathematical element to model a composite surface. Some surfaces may consist of one patch only while others are connected together.

The topology of a patch may be rectangular or triangular as shown in Fig (2.10). To generate curves of a surface patch, one can fix the value of one of the parametric variables, u, to obtain a curve in terms of the other variable v. By continuing this process first for one variable and then for the other using a certain set of arbitrary values in the permissible domain, a network of two parametric families of curves are generated. We can specify a mesh size, say $m \times n$, to display a surface on the graphics display.

A tube surface is generated by sweeping an area of space by a planar circle along a specified central curve $\vec{x}(u)$. The planar circle can change its radius as u varies. Given an orientation of objects on curved paths with vectors $\vec{T}(u)$, $\vec{N}(u)$ and $\vec{B}(u)$, the tube surface (Eberly, 2001) is defined by

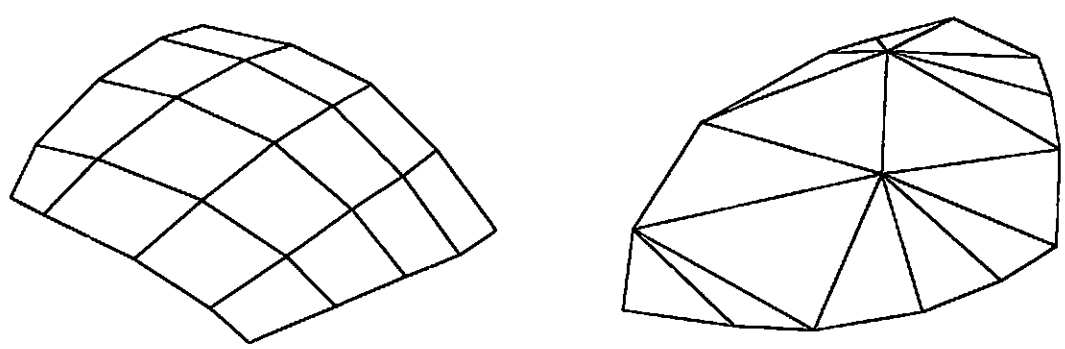


Parametric Space



Cartesian Space

Figure 2.9: Parametric representation of a three-dimensional surface



(a) Rectangle Patches (b)Triangular Patches

Figure 2.10: Surfaces composed of rectangular and triangular patches

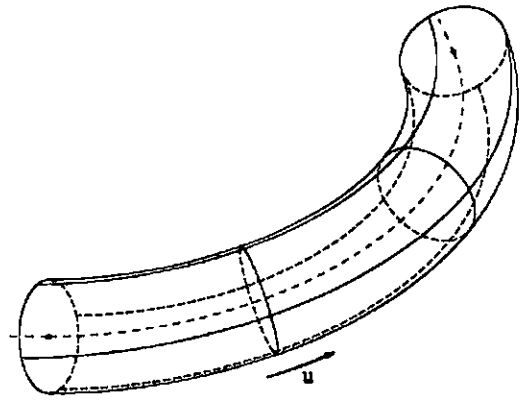


Figure 2.11: Tube surface

$$\vec{S}(u,v) = \vec{\mathrm{X}}(u) + r[\cos 2\pi v \vec{N}(u) + \sin 2\pi v \vec{B}(u)] \tag{2.4.4}$$

A single helix surface equation, *S(u,v)*, based on Eq. 7, becomes

$$\begin{cases} x = R_s \cos[2\pi\lambda(u_s+\varphi)] - r\cos 2\pi v \cos[2\pi\lambda(u_s+\varphi)] \\ +r\sin 2\pi v \sin[2\pi\lambda(u_s+\varphi)]\cos\alpha \\ y = R_s \sin[2\pi\lambda(u_s+\varphi)] - r\cos 2\pi v \sin[2\pi\lambda(u_s+\varphi)] \\ -r\sin 2\pi v \cos[2\pi\lambda(u_s+\varphi)]\cos\alpha \\ z = 2\pi R_s u/\tan\alpha + r\lambda \sin 2\pi v \cos\alpha \end{cases} \tag{2.4.5}$$

Similiarly, the surface equations for the double and triple helical wires are found with Eq (2.4.4) where all terms are expressed in terms of the two parameters, u_s and v.

$$\vec{S}_d(u_s,v) = \vec{\mathrm{X}}_d(u_s) + r[\cos 2\pi v \vec{N}_d(u_s) + \sin 2\pi v \vec{B}_d(u_s)] \tag{2.4.6}$$

$$\vec{S}_t(u_s,v) = \vec{\mathrm{X}}_t(u_s) + r[\cos 2\pi v \vec{N}_t(u_s) + \sin 2\pi v \vec{B}_t(u_s)] \tag{2.4.7}$$

where

$$\vec{T}_m = \frac{\dot{x}_i}{\left\|\dot{\mathrm{X}}_m(u_s)\right\|}\vec{i} + \frac{\dot{y}_j}{\left\|\dot{\mathrm{X}}_m(u_s)\right\|}\vec{j} + \frac{\dot{z}_k}{\left\|\dot{\mathrm{X}}_m(u_s)\right\|}\vec{k} \tag{2.4.8}$$

$$\vec{B}_m = \frac{\dot{\mathrm{X}}_m \otimes \ddot{\mathrm{X}}_m}{\left\| \dot{\mathrm{X}}_m \otimes \ddot{\mathrm{X}}_m \right\|} \tag{2.4.9}$$

$$\vec{N}_m = \vec{B}_m \otimes \vec{T}_m \tag{2.4.10}$$

where $m = d$ (double helix) or t (triple helix).

Chapter 3

Cable Model Visualization

To visualize the cable model we need decide how and what to draw on the computer screen. Generally this includes two aspects. One is a high-level decision that provides scene graphs and their manipulation and specific types of objects and algorithms that are part of the scene graph system. Another is a low-level decision called *renderer*. It is responsible for how to draw objects in the 3D world on a 2D computer screen.

The first step is to transform the 3D data in world space to 3D data in view space. View space specifies a coordinate system that supports what to draw. A second transformation converts the data in the view space to 2D data in screen space, called a projection. During this process, the data will be drawn as pixels on the computer screen.

The second step of a renderer is to eliminate portions of the invisible data to the observer. Two concepts are involved. The concept of culling is a process that determines whether an object is completely out of view. The clipping is to split an object into smaller pieces, visible parts are processed and invisible parts are discarded.

The third step in this process of drawing the 2D data that has been transformed to screen space is called rasterization. Most of time for rendering is spent on rasterization. Current-generation graphics cards are designed to accelerate rasterization, although a general-purpose processor still can work on the process.

3.1 The Graphics Rendering Pipeline

3.1.1 Object Space, World Space and Camera Space Transform

In order to transform geometrical data into a two-dimensional viewport we need to use several different coordinate systems. These different coordinate systems includes object space, world space and camera space. These different coordinate systems are associated with the rendering pipeline-their relationships are shown in Figure 3.1.

The vertices of a cable model are stored in object space, where object space expresses vertices relative position. The cable is made of several types of wires. After turning and bending these wires single, double and triple helical structures are formed. We assume these helical structures of the cable model have a common center that are always located on origin point of the cable's local coordinate system and the wires is extended along the z-axis of the cable's local coordinate system, (See figure 3.2). Through the parametrical mathematical equation we can generate graphical geometrical vertices of these wires based on the above-mentioned assumption. In the object space the vertices relative position would not be changed.

The position and orientation of the cable model are stored in world space. A global coordinate system ties all of the object spaces together. When we pan or rotate the cable, the cable will generate a new object coordinate by matrix transformation in the World space. World space also includes other objects information like light and camera position in addition to the cable.

Before a cable is rendered, its vertices need to be transformed into camera space, In space, the x and y axes are aligned to the display screen and the z-axis is parallel to the viewing direction. The rendering pipeline will transform vertices from object space into camera space by connecting the matrices representing the transformations from object space to world space and from world space to camera space. The product of these transformations can be called the model-view transformation. The total transformation matrix from the model space coordinates to the view space coordinates of the objects can be written $M_{total} = M_{project} M_{view} M_{world}$. This detailed application will be discussed in section 3.5.

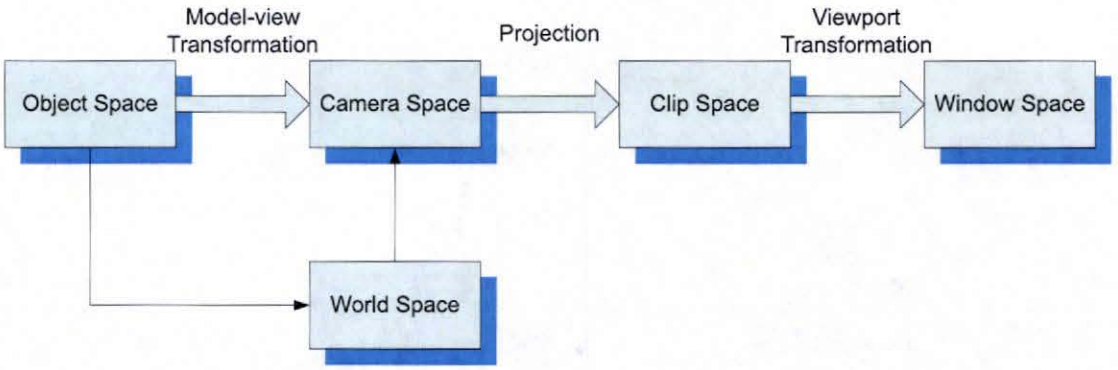


Figure 3.1: The coordinate spaces appearing in the rendering pipeline

3.1.2 Culling and Clipping

World space is a very big space. We don't need to process all objects in the world since we only can see limit space from computer screen. We will regulate a region of space called *view volume*. All objects that are completely outside the view volume are not processed. The objects totally inside the view volume are processed for display on the computer screen. If the objects intersect the boundary of the view volume, they need be clipped along the boundary, then processed for display on the screen. Culling and clipping of objects can reduce the amount of processing data before sending them to rasterization.

And the space in which the vertices exist after projection is called homogeneous clip space. Homogeneous clip space is so named because it is in this space that graphics primitives are clipped to the boundaries of the visible region of the scene, ensuring that no attempt is made to render any part of a primitive that falls outside the viewport. In homogeneous clip space, vertices have their own coordinates. The term normalized pertains to the fact that the x, y, and z-coordinates of each vertex fall in the range of the culling view volume, but reflect the final positions in which they will appear in the viewport. The vertices must undergo one more transformations, called the viewport transformation, that maps the normalized coordinates to the actual range of pixel coordinates covered by the viewport. The z-coordinate is usually mapped to the floating-point range [near, far], but this is subsequently scaled to the integer range corresponding to the number of bits per

pixel utilized by the depth buffer. After the viewport transformation, vertex positions are said to lie in window space.

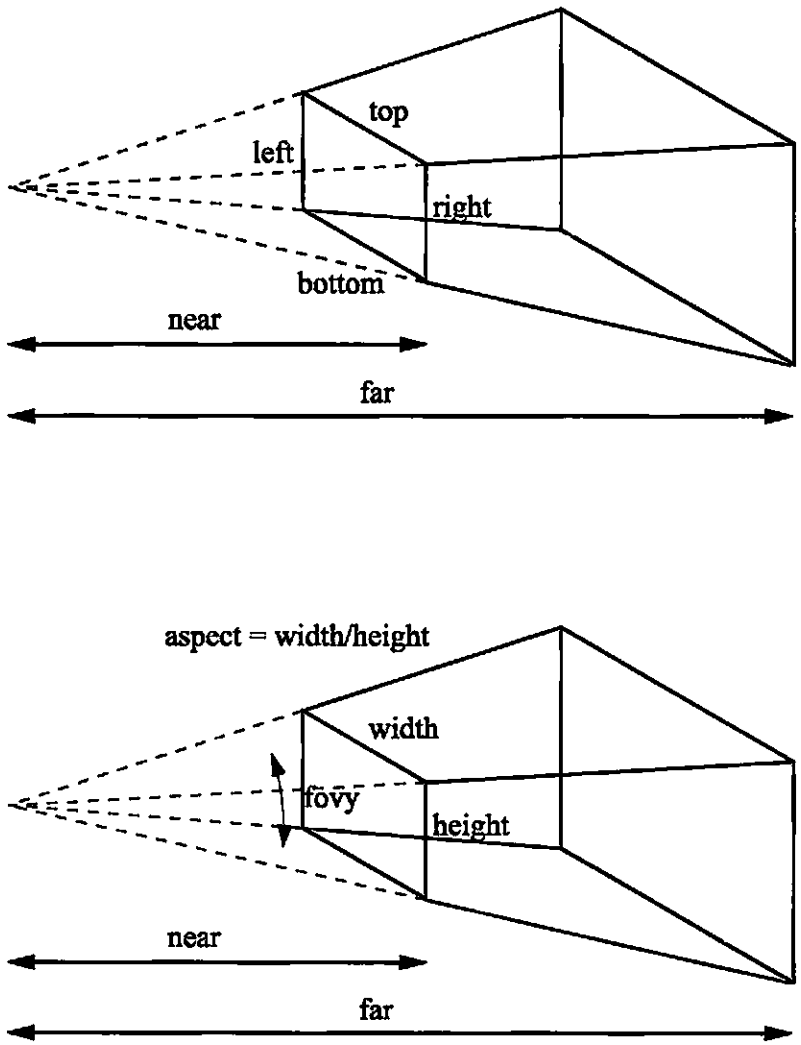


Figure 3.2: Perspective culling volume

Culling refers to eliminating portions of an object, possibly the entire object which are out of the view volume. Culling usually includes object culling and back face culling. Clipping refers to computing the intersection of an object with the view frustum so that only the visible portion of the object is sent to the rasterizer. An application may specify that face culling be performed as the first stage of this process. Face culling applies

only to polygonal graphics primitives and removes either the polygons that are facing away from the camera or those that are facing toward the camera.

3.1.3 Rasterization

Rasterization is the process of displaying a geometric entity in screen space and selecting those pixels to be drawn. Once model vertices have been clipped and transformed into window space, the renderer must determine what pixels in the viewport are covered by each graphics primitive. The process of filling in the horizontal spans of pixels belonging to a primitive is called rasterization. The CPU calculates the depth, interpolated vertex colors, and interpolated texture coordinates for each pixel. This information, combined with the location of the pixel itself, is called a fragment. The process through which a graphics primitive is converted to a set of fragments is illustrated.

3.2 Graphics Library

The last section described the relevant process in building a renderer without considering whether the work is done by a general-CPU or hardware-accelerated graphics card. The reality of building a real-time computer graphics library requires an understanding of what platforms are to be supported and what other existng systems can be used. If we want to build a graphics library, it cost much time to think of many rendering techniques, such as space transformation, culling and clipping, building a light model and other algorithms. If we want to let the hardware support our graphics library, we also need to research and use hardware instructions. For most graphical systems a mature graphics library is used, such as OpenGL and direct3D, etc,. These graphics library provide an application programmer interface (API). We can develop our own application program by connecting the graphics library without regard to low-level hardware. Here we use OpenGL library. We chose OpenGL as our base graphics language for a number of reasons. It is designed to be full featured, to run efficiently on a wide range of graphics architectures, and is clean and straightforward to use. It has a clear specification convenient to the application. OpenGL also is supported by many different graphics hardware and driver implementations. So we

don't need care so much about a rendering pipeline. We can put focus on modeling and setting environment variables to get a realistic cable model plot.

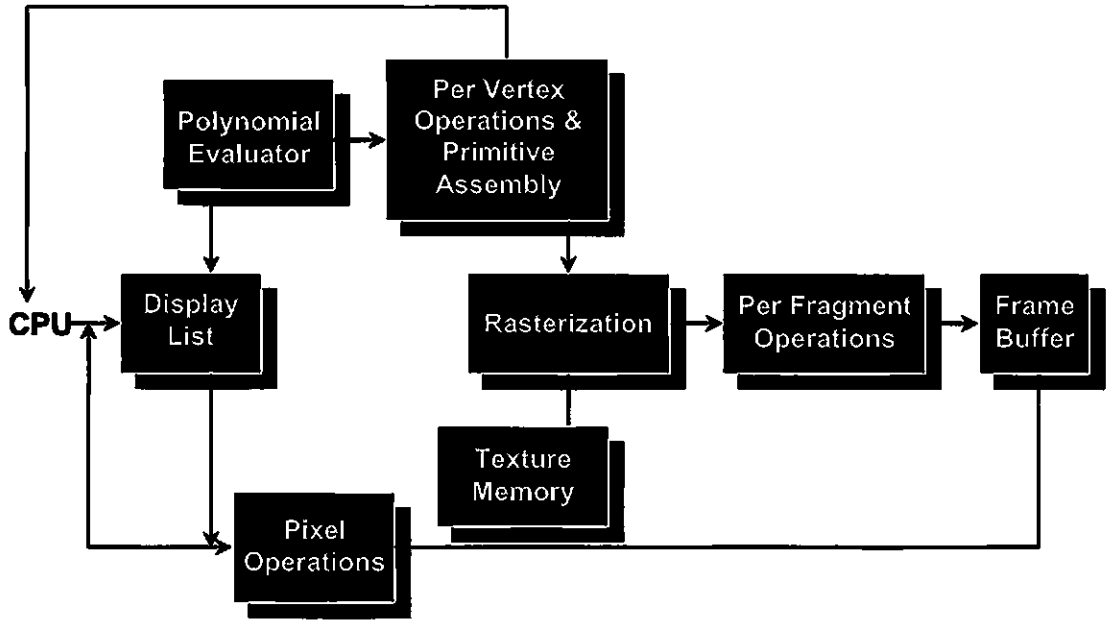


Figure 3.3: OpenGL block diagram

In our graphics system we need to use very high level objects and operations in computer graphics, low level data structures such as arrays or files are often utilized to write programs and implement algorithms.

3.3 Interactive Computer Programming

In many engineering applications, analysis requires a geometric model (data). In such a case programming is a useful tool. For our cases there are many different geometries such as single, double and triple helices in cable models. At the same time the same wire is repetitively used since in one lay there are several wires with the same geometrical parameter, except that starting angle is different.

These wires themselves also have different cross-sections including circles, keystones and rectangles. They can be used to generate many kinds of cable structures. We not only use similar meshing methods for these single, double and triple helical structures

combining with different cross-sections, but also we need to create a program that accepts the geometric parameters and generates the model automatically.

This section indicates programming that processes both graphics (entity data) and nongraphics (cable parameters) data on a cable graphics system, and that controls the operation of the graphics system. The major advantage of interactive programming is to automate some procedures of the design process, hence improving the user's productivity. The following visualizing applications of the cable to be used include:

1. Automatic graphics creation. Repetitive graphics work can be performed automatically via interactive programming. Consider that the structure of the cable is made of many repetitive wires. Wires in one cable usually have the similar shapes with different dimensions. To write a program to generate the cable structures, the first step is to parameterize the wire that belong to the cable. This step involves describing the wire's shape by a set of variables. When these variables are assigned certain values, the corresponding model of the wire can be generated. The second step is to write a program in terms of these variables. When we executes the program, it generates the part after we supply the desired values for the all the variables.

2. Create interactive menus. Here we will create some customized menus and icons to help understand cable structure from different view and change partial parameters of the cables.

3. Create cable and parametric design. By changing geometric and other input data for a given program, we can study the effect of certain parameters on a particular design.

3.3.1 Object-Oriented Programming

In the real world every object has its own action, properties and conditions. Sometimes between these objects there are some similar properties and conditions. We can describe these objects with similar actions, properties and condition into one class. In traditional programming, the languages can't distinguish between the general properties of any shape or object (a shape has a color, it can be drawn, etc.) and the properties of a

specific shape (a single helical wire is a shape with a radius and lay angle, is drawn by a single helical function, etc.).

When we use traditional programming languages to describe high level objects and operations in computer graphics, it will need many sub-functions since we have to write a particular programs for particular problem and condition. These functions can't express the relations between the objects. It also increases programming complexity. For example, the single helical round wire and the single helical tube have the same centerline when lay radius and lay angle are the same. We can write a function to calculate the centerline of the single helix, but it doesn't indicate this centerline is used for a round wire or tube wire. It also doesn't indicate input value coming from a single helical round or tube. When we want to modify the input value, we have to check where input values come from. When the different objects and the object's characteristics become more, the sub-function for particular properties and problems will increases remarkably.

In object-oriented programming, we use similar methods - building a class. The same description of a group of properties (data elements) and the same behavior (functions) target. Class is actually data types, for example, also has a float characteristics and behavior. The difference is the definition of categories of programmers to adapt to specific problems, rather than be forced to use the existing data types.These data types exist only motive is to describe the design of the machine storing unit. Programmers can add new data types need to expand the programming language. The Commission welcomed the creation of the Department of Design, a new category of concern to them with the same type of internal type checking.

Object-oriented programming is a technique or paradigm for writing "good" programs for a set of problems. Object-oriented programming has the ability to express this distinction and take advantage of it. Advantages of object-oriented programming include easier program design, as the objects correspond closely to the behavior of items being simulated or calculated; Relative fewer program errors, as objects promote modularity and encapsulation; and easier program extension, when new objects need to added. We often meet a problem between sophisticated algorithms, using very high level data structures, and poor methods of program and object construction. Particular programming languages

are too easily found for solving problems before a precise specification of the problem to be solved has been formulated.

A cable is two or more wires or optical fibers bound together, typically in a common protective jacket or sheath. We know the wire cross-section has the shapes: round, tubular or a jacket round wire, keystone and rectangle. Such components then are combined as straight, single helical, double helical and triple helical characteristics. Combining these wires characteristics, we build our own class which use saved wire basic parameters and calculate the geometric properties, (see Figure 3.4).

Here we also define our own-defined types of data. They provide users with constructs that let us express the distinction discussed in the above paragraph. In object-oriented languages, programs are based on objects, which are record-like data structures. Each type, or class, of object is associated with a particular set of procedure-like operations called "method", and methods are performed when objects are invoked by "messages". Each item data within a program is regarded as an attribute of some objects and only accessed by invoking one of the methods defined for that object's class.

We classify the wire into seventeen classes. Every class has its own geometric parameters. For example a single helix includes six main parameters: start angle, start angle,wire radius, wire direction, wire length and wire lay angle. These parameters are enough to describe the geometry characteristics of the single helical line. Other wires we also assign different parameters according to their geometrical structure.

In object-oriented languages there is an important concept - inheritance. During development it involves a large number of classes like above-mentioned wire class. These are very different compared with others, such as cross-section and helical structure. But these characteristics are not unique. They also have some similar characteristics. To control the resulting potential complexity, we need a classification mechanism, known as inheritance. A class will be an heir of another if it uses the other features in addition to its own. Using inheritance we don't need to write the code for every wire. It can share their common geometrical structure or other properties. A descendant is a direct or indirect heir; the reverse notion we can call ancestor. In this application we need to generate geometric class to describe these wires. We can assume one class called TGeometry as basic geometry class. On the basis it is extended to an other geometrical class: TPoint3D (3d point),TPlane

(geometry plane) and TWireObject (basic wire class). From Figure 3.3 we can understand the hierarchy structure of the geometry class.

Since OpenGL's library isn't using object-oriented languages to write, in order to connect OpenGL display data, we also need to specify an OpenGL wire class. OpenGL wire class only saves surface points, center point data, and other graphic information.

3.3.2 Data Structure

In section (3.3.1) we describe the cable geometric characteristic and also build build corresponding wire class. The cable is made of a serial wires. So in our program we need think how to organize these wires together to generate the whole cable model.

Here we analyze the cable structure more deeply. The cable can be separated into different layer. In the same lay the wires are same except initial starting angle. Some of wires are individual wires. Some other wires are made of other wires which are double wires. Double wire also can have his own wire which is triple wire. So the cable can be assumed a tree structure. See figure (3.10)

For individual wire we also need provide points and surface normal direction to display graphic model of the cable. So our task have two steps. One is convert cable tree structure to linear structure. Another is connect each point in every wire. We also hoping the data structure could provide flexible function, such as modify its owns parameters.

Although there are many, many data structures used in standard programming, a large majority of them are built upon some variant of two fundamental containers: the array and the linked list. They are important not only because of their simplicity, but also because of their efficiency.

Arrays can be thought the simplest data structure; anything simpler would be a primitive data type like an integer or a Boolean. An array is a sequential list of elements, fixed in number. The elements are all of the same type, usually stored in one memory block, so that each element immediately follows the previous in memory. The elements are said to be contiguous in memory. The elements of an array by their numeric index: the first element is element 0; the second one is found at the number after that, and so on. In code, the element at index i is referred to as A[i], where A is the identifier for the array.

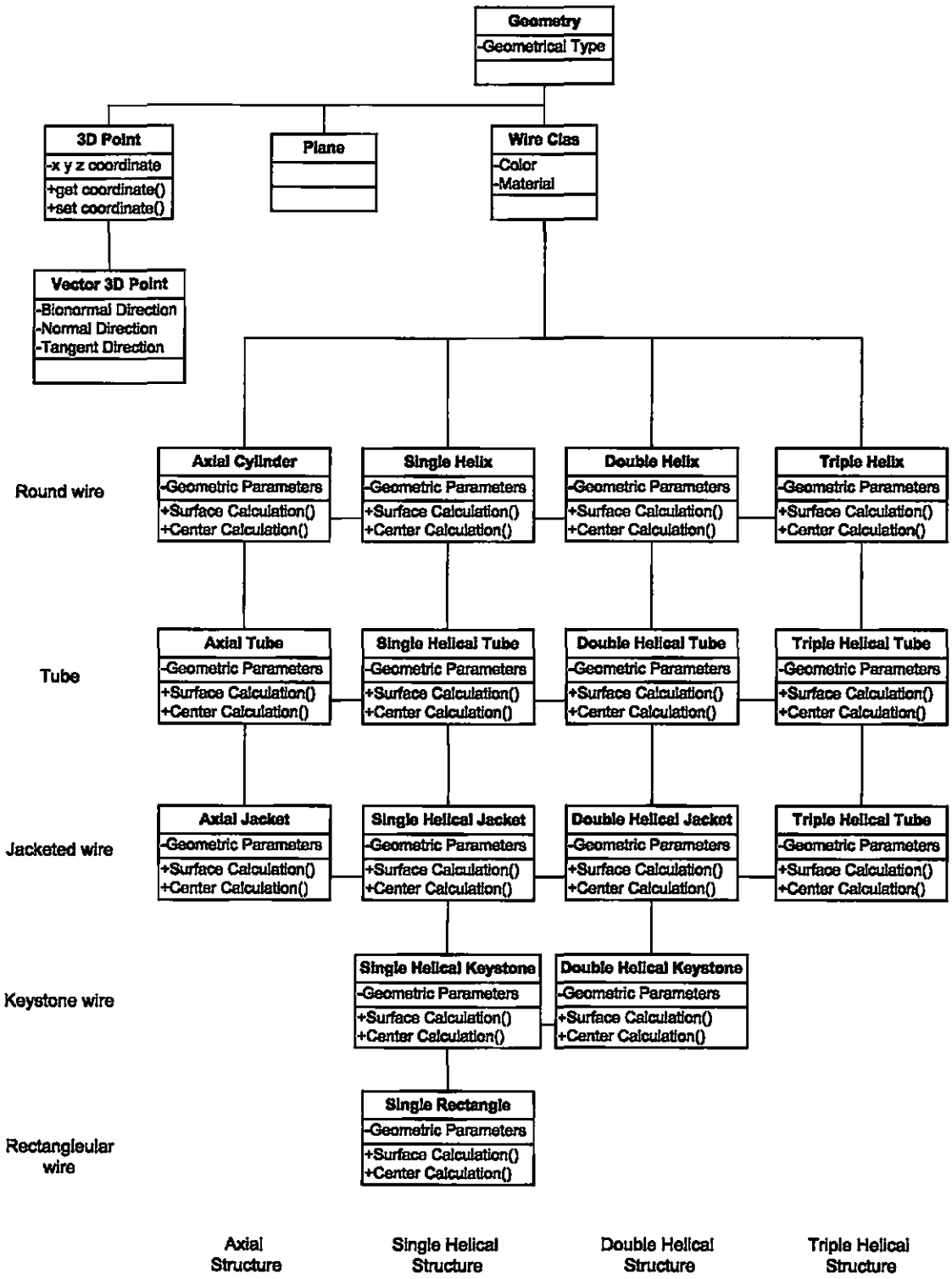
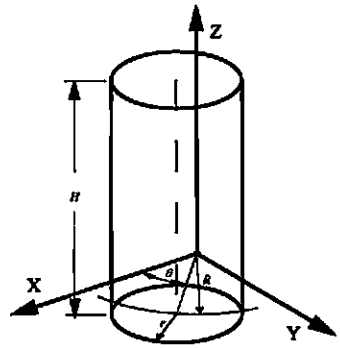
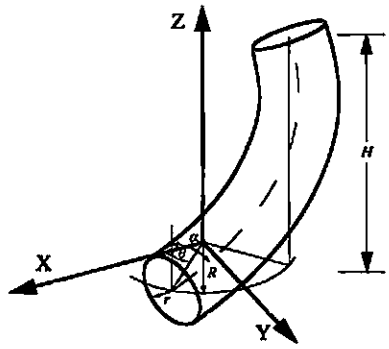


Figure 3.4: Wire class of the cable

Axical Cylinder
+Start Angle θ: double +Start Radius R : double +Wire Radius r : double +Wire Height H : double
+Set Basic Parameter() +Calculate CenterLine Points() +Calculate Surface Points()



Single Helix
+Start Angle : double +Start Radius : double +Wire Radius : double +Wire Direction : int +Wire Height : double +Wire Lay Angle : double
+Set Basic Parameter() +Calculate CenterLine Points() +Calculate Surface Points()

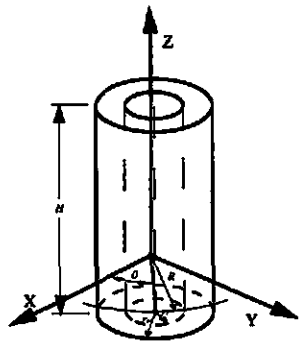


Double Helix
+Single Helical Start Angle : double +Single Helical Radius : double +Single Helical Wire Directioin : int +Single Helical Wire Lay Angle : double +Double Helical Start Angle : double +Double Helical Radius : double +Wire Radius : double +Wire Height : double +Double Helical Wire Directioin : int +Double Helical Wire Lay Angle : double
+Set Basic Parameter() +Calculate CenterLine Points() +Calculate Surface Points()

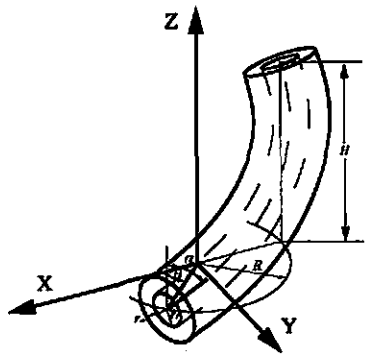
Triple Helix
+Single Helical Start Angle : double +Single Helical Radius : double +Single Helical Wire Directioin : int +Single Helical Wire Lay Angle :double +Double Helical Start Angle : double +Double Helical Radius : double +Wire Radius : double +Wire Height : double +Double Helical Wire Directioin : int +Double Helical Wire Lay Angle :double +Triple Helical Start Angle : double +Triple Helical Radius : double +Triple Helical Wire Direction : int +Triple Helical Wire Lay Angle : double
+Set Basic Parameter() +Calculate CenterLine Points() +Calculate Surface Points()

Figure 3.5: Round wire class

Axical Tube
+Start Angle : double +Start Radius : double +Wire Inner Radius : double +Wire Outer Radius : double +Wire Height : double
+Set Basic Parameter() +Calculate CenterLine Points() +Calculate Surface Points()



Single Helical Tube
+Start Angle : double +Start Radius : double +Wire Inner Radius : double +Wire Outer Radius : double +Wire Direction : int +Wire Height : double +Wire Lay Angle : double
+Set Basic Parameter() +Calculate CenterLine Points() +Calculate Surface Points()

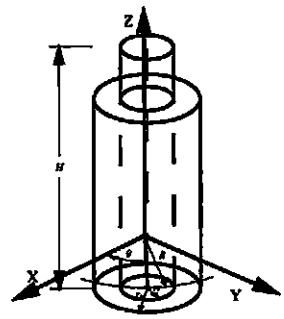


Double Helical Tube
+Single Helical Start Angle : double +Single Helical Radius : double +Single Helical Wire Directioin : int +Single Helical Wire Lay Angle : double +Double Helical Start Angle : double +Double Helical Radius : double +Wire Inner Radius : double +Wire Height : double +Double Helical Wire Directioin : int +Double Helical Wire Lay Angle : double +Wire Outer Radius : double
+Set Basic Parameter() +Calculate CenterLine Points() +Calculate Surface Points()

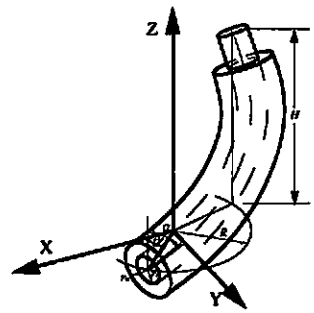
Triple Helical Tube
+Single Helical Start Angle : double +Single Helical Radius : double +Single Helical Wire Directioin : int +Single Helical Wire Lay Angle : double +Double Helical Start Angle : double +Double Helical Radius : double +Wire Inner Radius : double +Wire Height : double +Double Helical Wire Directioin : int +Double Helical Wire Lay Angle : double +Triple Helical Start Angle : double +Triple Helical Radius : double +Triple Helical Wire Direction : int +Triple Helical Wire Lay Angle : double +Wire Outer Radius : double
+Set Basic Parameter() +Calculate CenterLine Points() +Calculate Surface Points()

Figure 3.6: Tube wire class

Axial Jacket
+Start Angle : double +Start Radius : double +Wire Inner Radius : double +Wire Outer Radius : double +Wire Height : double
+Set Basic Parameter() +Calculate CenterLine Points() +Calculate Surface Points()



Single Helical Jacket
+Start Angle : double +Start Radius : double +Wire Inner Radius : double +Wire Outer Radius : double +Wire Direction : int +Wire Height : double +Wire Lay Angle : double
+Set Basic Parameter() +Calculate CenterLine Points() +Calculate Surface Points()

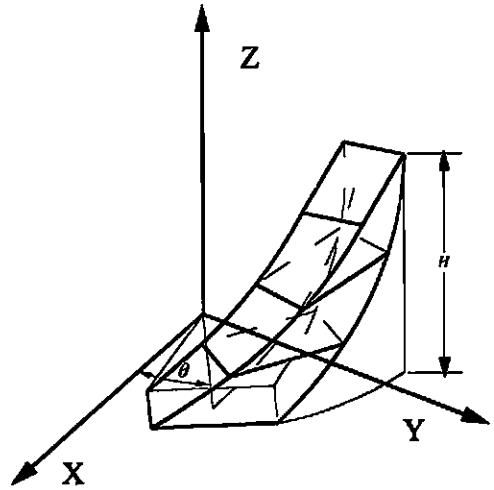


Double Helical Jacket
+Single Helical Start Angle : double +Single Helical Radius : double +Single Helical Wire Directioin : int +Single Helical Wire Lay Angle : double +Double Helical Start Angle : double +Double Helical Radius : double +Wire Inner Radius : double +Wire Outer Radius : double +Wire Height : double +Double Helical Wire Directioin : int +Double Helical Wire Lay Angle : double
+Set Basic Parameter() +Calculate CenterLine Points() +Calculate Surface Points()

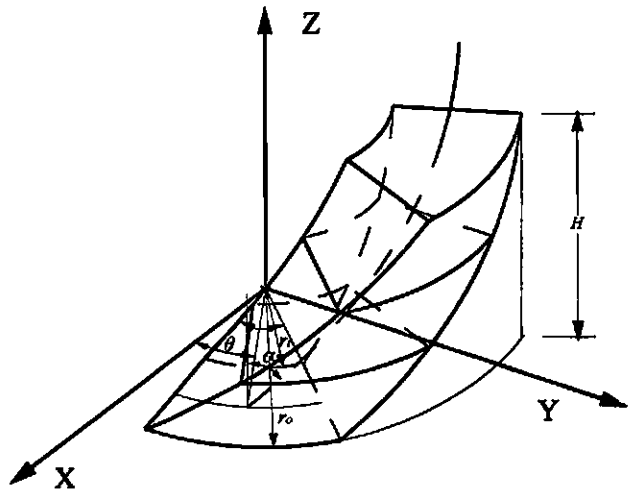
Triple Helical Jacket
+Single Helical Start Angle : double +Single Helical Radius : double +Single Helical Wire Directioin : int +Single Helical Wire Lay Angle : double +Double Helical Start Angle : double +Double Helical Radius : double +Wire Inner Radius : double +Wire Outer Radius : double +Wire Height : double +Double Helical Wire Directioin : int +Double Helical Wire Lay Angle : double +Triple Helical Start Angle : double +Triple Helical Radius : double +Triple Helical Wire Direction : int +Triple Helical Wire Lay Angle : double
+Set Basic Parameter() +Calculate CenterLine Points() +Calculate Surface Points()

Figure 3.7: Jacket wire class

Single Helical Rectangle
+Start Angle : double +Start Radius : double +Wire Radius : double +Wire Direction : int +Wire Height : double +Wire Lay Angle : double
+Set Basic Parameter() +Calculate CenterLine Points() +Calculate Surface Points()



Single Helical Keystone
+Start Angle : double +Start Radius : double +Wire Radius : double +Wire Direction : int +Wire Height : double +Wire Lay Angle : double
+Set Basic Parameter() +Calculate CenterLine Points() +Calculate Surface Points()



Double Helical Keystone
+Single Helical Start Angle : double +Single Helical Radius : double +Single Helical Wire Directioin : int +Single Helical Wire Lay Angle : double +Double Helical Start Angle : double +Double Helical Radius : double +Wire Radius : double +Wire Height : double +Double Helical Wire Directioin : int +Double Helical Wire Lay Angle : double
+Set Basic Parameter() +Calculate CenterLine Points() +Calculate Surface Points()

Figure 3.8: Keystone and rectangle wire class

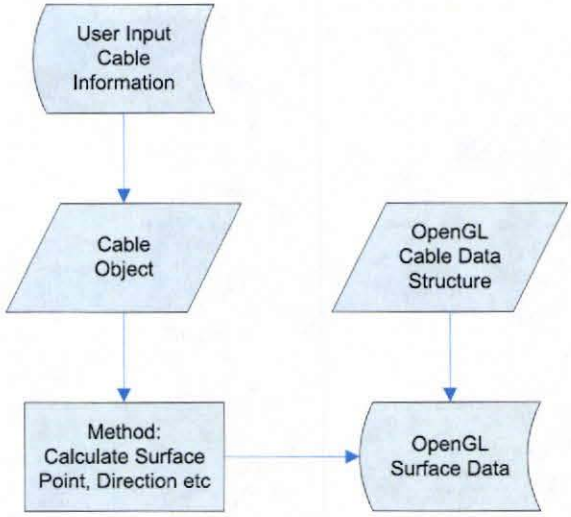


Figure 3.9: Cable object connect OpenGL data structure

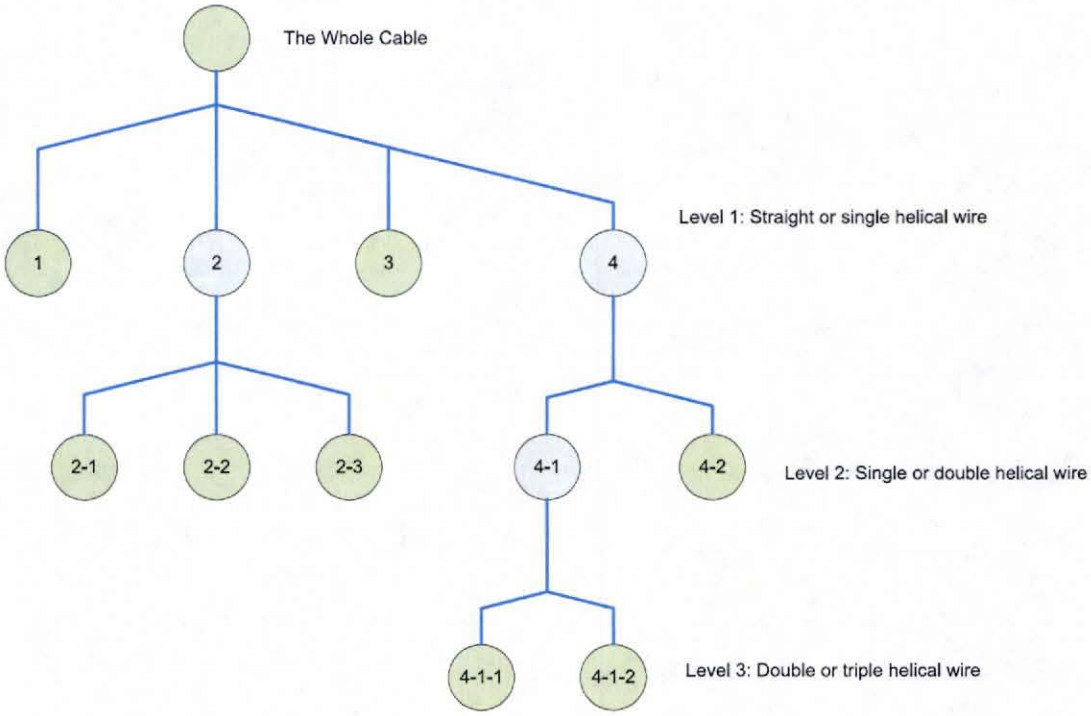


Figure 3.10: Cable tree structure

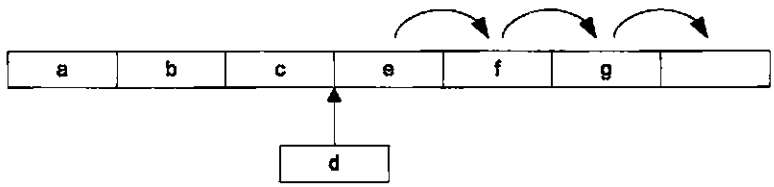


Figure 3.11: Insert a array

The figure. (3.11)demonstrate the array structure in memory and how to insert a value. We find that there are several shortcomings if we use array to arrange the cable structure. First every type of wire has different data type. But array usually save the same type data. Even we use array to save wire data, we have to calculate the whole array capacity in order that array is big enough to take the whole cable data. But when parametric structure of some wires changes, it means the data capacity change to new value. We have to recalculate the whole array capacity again and set all new data again. From the figure (3.11) we can find more serious problem which means we need move back the data from the original position. Since the cable structure is complex and huge data. It will takes computer long time to moving these data. Obviously it is not efficient.

Like an array, a linked list is composed of many cells that contain data, although they are called nodes when referring to linked lists. In an array, cells are packed right next to each other in memory, and cells contain nothing but the data in the array.

A linked list is a chain of items or objects of some description (usually called nodes), with each item containing a pointer pointing to the next item in the chain. This is known as a singly linked list - every item has a single link or pointer to the next. The list itself is identified by the first node, from which all other nodes can be found (or visited) by following the links one by one. Notice the difference in definition from an array, where the next item in line is physically adjacent to the current item. With a linked list, the items may be all over the place, their ordering being maintained by the links.

From the figure.(3.12) we can learn the singly linked list insert one data only need two steps. First step set the Next pointer in our new node to the node after the given node. Second step set the Next pointer of the given node to our new node. So the singly provide a more flexible data structure than an array. We can conserve memory or be able to insert and remove data quickly. For example when we change some layer parametric value, we

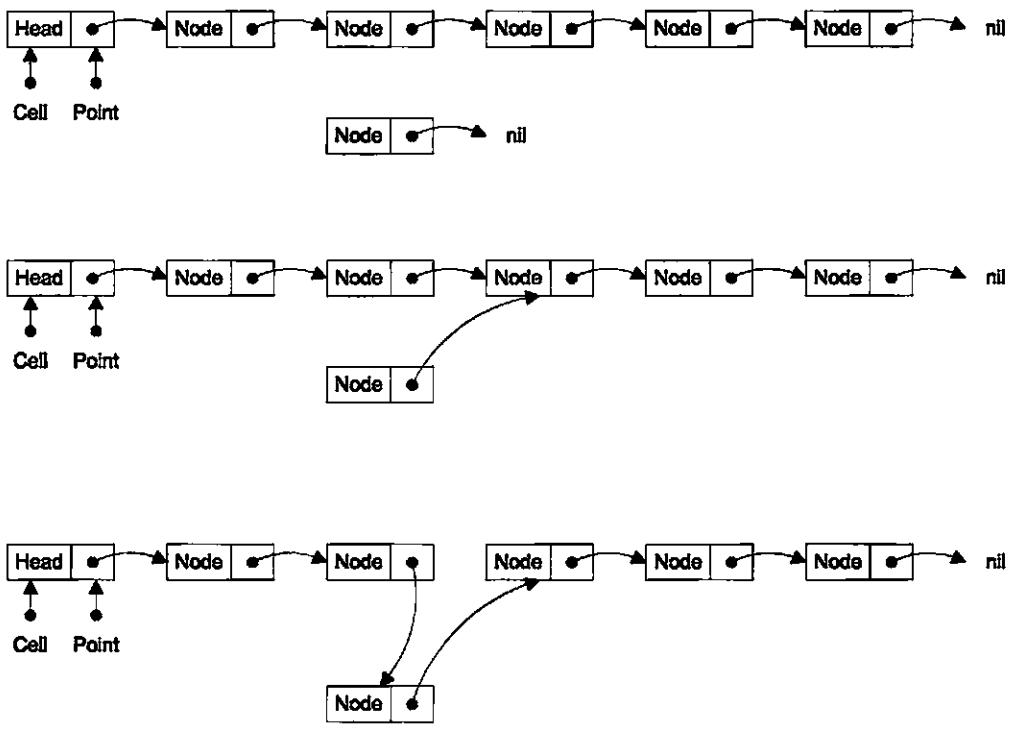


Figure 3.12: Insertion into a singly linked list

only need delete corresponding layer geometric data then insert new data. If the cable have a lot of layer this structure can improve execute efficient evidently.

we can change the cable structure to singly linked list. For example the figure. (3.13) shows the new structure compared with figure. (3.10). We can see some wires which include sub-wires disappear since these type wires have not corresponding wire class and are changed to sub-wires. So our program can convert any kind of cable to a serial of wires then save into the singly linked list.

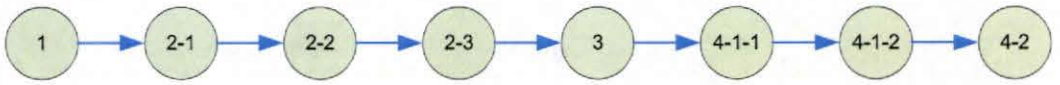


Figure 3.13: A singly linked list of the whole cable

Inside the wire we also use singly linked list to connect centerline points and surface points.

3.3.3 Cable Height Setting

In order to see the inside cable the better way is setting different length for different wire layer like Christmas tree. The program can automatically assign the different height for wire which is in different layer. The basic idea is that the height depends on the layer radius. And the similar lay can get same height.

Here we use golden ratio to set height for different layer. The golden ratio, usually denoted , expresses the relationship that the sum of two quantities is to the larger quantity as the larger is to the smaller. The golden ratio is the following irrational number and numerical approximation:

$$\Phi = \frac{1+\sqrt{5}}{2} \approx 1.618$$

The figure of a golden section on the right illustrates the defining geometric relationship.

The program flow of the height setting as followed:

We also provide manually setting height function.

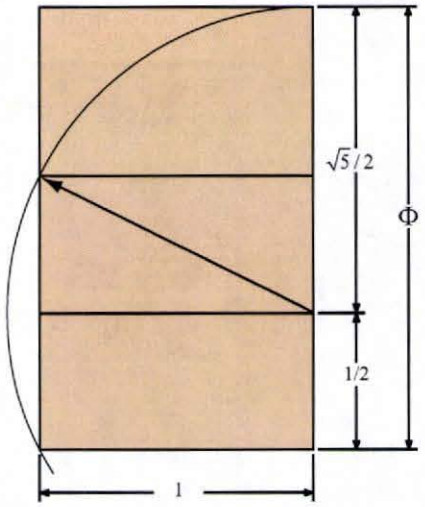


Figure 3.14: Construction of a golden rectangle

3.3.4 Subdivision

A subdivision surface, in the field of 3D computer graphics, is a method of representing a smooth surface via the specification of a coarser piecewise linear polygon mesh. The smooth surface can be calculated from the coarse mesh as the limit of an iterative process of subdividing each polygonal face into smaller faces that better approximate the smooth surface. For drawing purposes, it is necessary to produce a piece linear approximation to a curve with $n+1$ curve points that will be the line segment end points. If t_i are the selected curve parameters for $0 \leq i \leq n$, then the set of points $x_i = x(t_i)$ for $0 \leq i \leq n$ is referred to as a subdivision of the curve.

Commonly, more points can describe more smooth surfaces, but it costs much more calculating time and memory. So we have to make a balance between these two conditions.

One of the simplest ways to speed up an OpenGL program while simultaneously saving storage space is to convert independent triangles or polygons into triangle strips. If the model is generated directly from NURBS data or from some other regular geometry, it is straightforward to connect the triangles together into longer strips.

Since OpenGL specify two types of triangle strips, we can choose GL_TRIANGLE_STRIP and GL_TRIANGLE_FAN to make wire surface. Decide whether the first triangle should

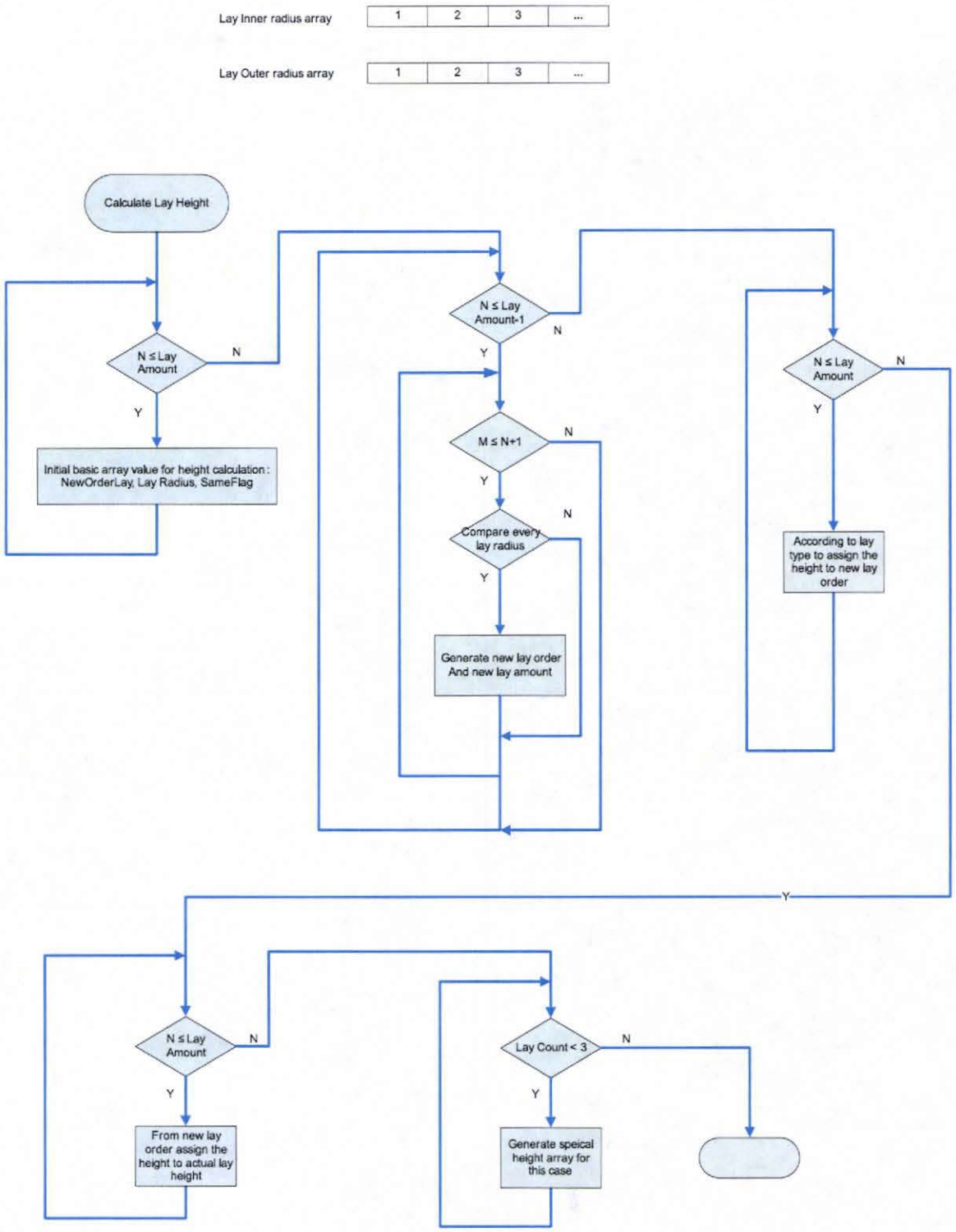


Figure 3.15: Flow plot of the height setting

have a clockwise or counterclockwise winding, then ensure all subsequent triangles in the list alternate windings (as shown in Figure 1.10). Triangle fans must also be started with the correct winding, but all subsequent triangles are wound in the same direction (Figure 1.11). In general, the triangle strip is the more versatile primitive. While triangle fans are ideal for large convex polygons that need to be converted to triangles or for triangulating geometry that is cone-shaped, most other cases are best converted to triangle strips.

Meshing of a helical surface is a straightforward process. The helical surface is defined parametrically by Eq (2.4.4-2.4.5). The parameter space is $[u_{min}, u_{max}] \times [0, 2\pi]$ and can be uniformly subdivided into triangles as shown in figure(). Let $u_i = u_{min} + i(u_{max} - u_{min})/n$ for $0 \leq u \leq n$ and $v_j = j(2\pi)/m$ for $0 \leq v \leq m$. These vertices participating in the meshing are $\vec{S}_{i,j} = \vec{S}(u_i, v_j)$. Since the surface is a tube, so $\vec{S}_{i,m}$ is equal to $\vec{S}_{i,0}$.

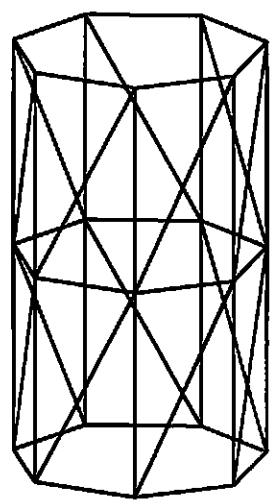


Figure 3.16: A helical mesh made up of multiple triangle strips

3.4 Color, Shading and Lighting

In this section we will discuss the basics of color representation, lighting models, and shading objects. Geometric modeling operations are responsible for accurately reproducing shape, size, position, and orientation of the cables. Color, lighting, and shading

are the next step to produce the visual appearance of an object. Correct settings for these values can produce photo-realistic object in OpenGL.

3.4.1 Representing Color

To produce more realistic images, objects being rendered must be shaded with accurate colors. Modern graphics accelerators can generate colors from a large, but finite palette. In OpenGL, color values are specified to be represented with a triple of floating-point numbers in the range [0, 1]. These values specify the amount of red, green, and blue (RGB) primaries in the color. RGB triples are also used to store pixel colors in the framebuffer, and are used by the video display hardware to drive a cathode ray tube (CRT) or liquid crystal display (LCD) display.

A given color representation scheme is referred to as a color space. The RGB space used by OpenGL is a cartesian space well suited for describing colors for display devices that emit light, such as color monitors. The addition of the three primary colors mimics the mixing of three light sources.

Color buffers consist of either unsigned integer color indices or R, G, B, and, optionally, A unsigned integer values. The number of bitplanes in each of the color buffers, the depth buffer, the stencil buffer, and the accumulation buffer is fixed and window dependent. Should we require minimums? If an accumulation buffer is provided, it must have at least as many bitplanes per R, G, and B color component as do the color buffers.

3.4.2 Lighting

When we look at a physical surface, the eye's perception of the color depends on the distribution of photon energies that arrive and trigger our cone cells, as described in "Color Perception." Those photons come from a light source or combination of sources, some of which are absorbed and some of which are reflected by the surface. In addition, different surfaces may have very different properties - some are shiny, and preferentially reflect light in certain directions, while others scatter incoming light equally in all directions. Most surfaces are somewhere in between.

OpenGL approximates light and lighting as if light can be broken into red, green, and blue components. Thus, the color of light sources is characterized by the amount of red, green, and blue light they emit, and the material of surfaces is characterized by the percentage of the incoming red, green, and blue components that are reflected in various directions. The OpenGL lighting equations are just an approximation, but one that works fairly well and can be computed relatively quickly.

Lighting has four properties: Emitted, Ambient, Diffuse, and Specular Light

Emitted light is the simplest - it originates from an object and is unaffected by any light sources.

The ambient component is the light from that source that's been scattered so much by the environment that its direction is impossible to determine - it seems to come from all directions. Backlighting in a room has a large ambient component, since most of the light that reaches the eye has bounced off many surfaces first. A spotlight outdoors has a tiny ambient component; most of the light travels in the same direction, and since we're outdoors, very little of the light reaches the eyes after bouncing off other objects. When ambient light strikes a surface, it's scattered equally in all directions.

Diffuse light comes from one direction, so it's brighter if it comes squarely down on a surface than if it barely glances off the surface. Once it hits a surface, however, it's scattered equally in all directions, so it appears equally bright, no matter where the eye is located. Any light coming from a particular position or direction probably has a diffuse component.

Finally, specular light comes from a particular direction, and it tends to bounce off the surface in a preferred direction. A well-collimated laser beam bouncing off a high-quality mirror produces almost 100 percent specular reflection. Shiny metal or plastic has a high specular component, and chalk or carpet has almost none. We can think of specularity as shininess.

Although a light source delivers a single distribution of frequencies, the ambient, diffuse, and specular components might be different. For example, if we have a white light in a room with red walls, the scattered light tends to be red, although the light directly striking objects is white. OpenGL allows we to set the red, green, and blue values for each component of light independently.

3.4.3 Shading

Shading is the term used to describe the assignment of a color value to a pixel. For photorealistic applications that try to generate images that look as good as photographs of a real scene ł the goal is to choose a color value that most accurately captures the color of the light reflected from the object to the viewer. Photorealistic rendering attempts to take into account the real world interactions between objects, light sources, and the environment. It describes the interactions as a set of equations that can be evaluated at each surface point on the object.

The shading computation is by definition a per-pixel-fragment operation, but portions of the computation may not be performed per-pixel. Avoiding per-pixel computations is done to reduce the amount of processing power required to render a scene. Figure 3.3 illustrates schematically the places in the OpenGL pipeline where the color for a pixel fragment may be modified by parts of the shading computation.

There are five fundamental places where the fragment color can be affected: input color, vertex lighting, texturing, fog, and blending. OpenGL maintains the concept of a current color, so if a new color is not issued with the vertex primitive, then the current color is used. If lighting is enabled, then the vertex color is replaced with the result of the vertex lighting computation. There is some subtlety in the vertex lighting computation. While lighting uses the current material definition to provide the color attributes for the vertex, if GL_COLOR_MATERIAL is enabled, then the current color updates the current material definition before being used in the lighting computation. After vertex lighting, the primitive is rasterized. Depending on the shading model (GL_FLAT or GL_SMOOTH), the resulting pixel fragments will have the color associated with the vertex or a color interpolated from multiple vertex colors. If texturing is enabled, then the color value is further modified, or even replaced altogether by texture environment processing.

Finally, if blending is enabled, then the fragment color value is modified according to the enabled blending mode. By controlling which parts of the pipeline are enabled and disabled, some simple shading models can be implemented:

Constant Shading If the OpenGL shading model is set to GL_FLAT and all other parts of the shading pipeline disabled, then each generated pixel of a primitive has the color

of the provoking vertex of the primitive. The provoking vertex is a term that describes which vertex is used to define a primitive, or to delineate the individual triangles, quads, or lines within a compound primitive. In general it is the last vertex of a line, triangle, or quadrilateral (for strips and fans, the last vertex to define each line, triangle or quadrilateral within the primitive). For polygons it is the first vertex.

Constant shading is also called flat or faceted shading. Smooth Shading If the shading model is set to GL_SMOOTH, then the colors of each vertex are interpolated to produce the fragment color. This results in smooth transitions between polygons of different colors. If all of the vertex colors are the same, then smooth shading produces the same result as constant shading. If vertex lighting is combined with smooth shading, then the polygons are Gouraud shaded.

Phong Shading Early computer graphics papers and books have occasionally confused the definition of the lighting model (lighting) from how the lighting model is evaluated (shading). The original description of Gouraud shading applies a particular lighting model to each vertex and linearly interpolates the colors computed for each vertex to produce fragment colors. We prefer to generalize that idea to two orthogonal concepts per-vertex lighting and smooth shading. Similarly, Phong describes a more advanced lighting model that includes the effects of specular reflection. This model is evaluated at each pixel fragment to avoid artifacts that can result from evaluating the model at vertices and interpolating the colors. Again, we separate the concept of per-pixel lighting from the Phong lighting model.

OpenGL computes surface shading by evaluating lighting equations at polygon vertices. The most general form of the lighting equation uses both the vertex position and a vector that is normal to the object's surface at that position; this is called the normal vector. Ideally, these normal vectors are captured or computed with the original model data, but in practice there are many models that do not include normal vectors. Given an arbitrary polygonal model without precomputed normals, it is easy to generate polygon normals for faceted shading, but a bit more difficult to create correct vertex normals when smooth shading is desired. Computing the cross-product of two edges,

3.4.4 Material

Like lights, materials have different ambient, diffuse, and specular colors, which determine the ambient, diffuse, and specular reflectance of the material. A material's ambient reflectance is combined with the ambient component of each incoming light source, the diffuse reflectance with the light's diffuse component, and similarly for the specular reflectance and component. Ambient and diffuse reflectance define the color of the material and are typically similar if not identical. Specular reflectance is usually white or gray, so that specular highlights end up being the color of the light source's specular intensity. If we think of a white light shining on a shiny red plastic sphere, most of the sphere appears red, but the shiny highlight is white.

As mentioned previously, the apparent smoothness of a material is a function of how strongly it reflects and the size of the specular highlight. This is affected by the overall magnitude of the GL_AMBIENT, GL_DIFFUSE, and GL_SPECULAR parameters, and the value of GL_SHININESS.

Here are some heuristics that describe useful relationships between the magnitudes of these parameters:

1. The spectral color of the ambient and diffuse reflectance parameters should be the same.

2. The magnitudes of diffuse and specular reflectance should sum to a value close to 1. This helps prevent color value overflow.

3. The value of the specular exponent should increase as the magnitude of specular reflectance approaches 1. Using these relationships, or the values in Table 3.1, will not result in a perfect imitation of a given material.

The empirical model used by OpenGL emphasizes performance, not physical exactness. We set the following table for some common useful material in cable. Improving material accuracy requires going beyond the OpenGL lighting model to more sophisticated multipass techniques or use of the programmable pipeline.

Following pictures shows corresponding single helix when we use material parameters setting in OpenGL graphic library.

Table 3.1: Common Material Library

Material	GL_AMBIENT	GL_DIFFUSE	GL_SPECULAR	GL_SHININESS
Brass	0.33 0.22 0.03 1.00	0.78 0.56 0.11 1.00	0.99 0.94 0.80 1.00	27.90
Bronze	0.21 0.12 0.05 1.00	0.71 0.43 0.18 1.00	0.39 0.27 0.17 1.00	25.6
Copper	0.19 0.07 0.02 1.00	0.70 0.27 0.08 1.00	0.26 0.14 0.09 1.00	12.8
Silver	0.19 0.19 0.19 1.00	0.51 0.51 0.51 1.00	0.51 0.51 0.51 1.00	51.2
Grey Plastic	0.77 0.77 0.69 1.00	0.01 0.01 0.01 1.00	0.50 0.50 0.50 1.00	85
Black Rubber	0.02 0.02 0.02 1.00	0.01 0.01 0.01 1.00	0.4 0.4 0.4 1.00	90

Figure 3.17: Single Helix with different material

3.5 Geometric Transformation and Camera Control

Most CAD system graphics packages contain graphics concepts that produce the functions and interactivity of the system. Some of these concepts are geometrical transformations, viewing in three dimensions, modeling and object hierarchy, algorithms for removing hidden edges and surfaces, shading and coloring, and clipping.

Geometric transformations play a central role in model construction and viewing. With an input device they can be used in modeling to express locations of objects relative to others. In generating a view of an object, they are used to obtain the effect of different viewing positions and directions. Typical CAD construction commands including translate, and rotate, and zoom are all based on geometric transformations. After we complete the parametric construction of the cable model, its viewing in its modeling space can be achieved again through geometric transformation. Perspective views of a geometric model can be obtained by projecting the model onto the proper plane. Additionally, we design the model itself which can be rotated up and down to view it in its three-dimensional space.

Geometric transformations also are fitting for computer graphics applications and object modeling since the utilized geometry in OpenGL is point-based. In applications where the view point changes rapidly or where objects move fast in relation to each other, transformation of these points must be carried out rapidly and repeatedly. Therefore it is necessary to find efficient ways of performing three-dimensional transformation.

Most of the transformations implemented at the hardware level are provided commonly by CAD/CAM systems.

3.6 Transformations of Geometric Models

Geometric transformations are moving from one coordinate system to another coordinate system. In other words, the description of a geometric model of an object can change within its own Model Coordinate System. This means that the geometric model must undergo motion relative to its Model Coordinate System. Here we assume the motion is the rigid-motion, and that the object does not deform during the motion. Typical transformations include translation, rotation, reflection or scaling and any combination of them.

For our case we will apply translation and rotation to our parametric cables such as single helix, double helix, keystone, etc. The transformation matrix will provide a easy way to develop and implement geometric transformations.

From the last chapter we used the parametric equation of a helix to generate points of the helix cable, then using these points approaching the surface of the cable. So transformation of a point should be the important problem in geometric transformation. For example, a line can be represented by two points. And a general curved surface also can be generated by a series of points. Here we discuss how to transform a point. We can indicate a point **P** undergoing a motion to new position **P***

$$P^* = f(P, transformationparameters) \tag{3.6.1}$$

Geometric transformation should be unique. It means a given set of transformation parameters only get one new point from the old point. Another characteristic is that two transformations can be combinable to produce a single transformation.

In order to implement Eq(3.6.1) into OpenGL software package, it is better to express it in terms of matrix notation as

$$P^* = [T]P \tag{3.6.2}$$

here [T] is the transformation matrix. Its elements will be functions of the given transformation parameters. The matrix [T] should have some important properties. Later we will introduce homogeneous representation of Eq (3.6.2) which can explain translation.

3.6.1 Translation, Pan and Zoom

When every entity of a geometric model remains parallel to its initial position, the rigid-body transformation of the model is defined as translation. Translating a model means that every point on it moves an equal given distance in a given direction. Translation can be specified by a vector, a unit vector and distance, or two points that denote the initial and final position of the model to be translated. Figure (3.10) shows a point translated by vector **d**.

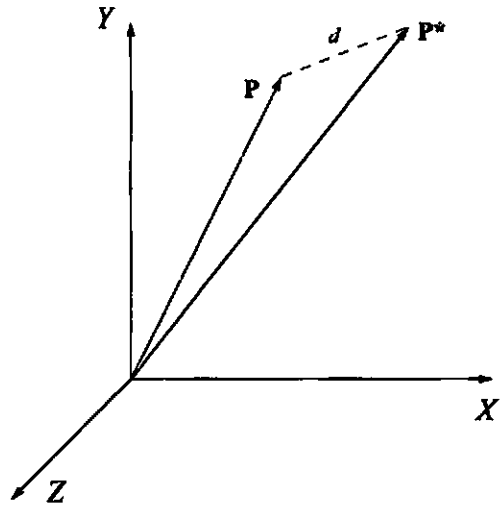


Figure 3.18: Translation of a curve

To relate the final position vector P* of a point P to its initial position vector P after being translated by a vector **d**, consider the triangle shown in Fig 3-10. In this case take the form

$$\mathbf{P}^* = \mathbf{P} + \mathbf{d} \tag{3.6.3}$$

This equation is applicable to three dimensional points and can be written in a scalar form for the three dimensional case as

$$\begin{aligned} x* &= x + x_d \\ y* &= y + y_d \\ z* &= z + z_d \end{aligned} \tag{3.6.4}$$

when Eq(3.6.3) can apply a point translated by the vector **d**. It is efficient and useful to the translation of an entity (curve, surface and solid) to its geometric representation.

3.6.2 Rotation

Rotation is an important form of geometrical transformation. Mainly it allows users to view geometric models from different angles and helps with many geometric operations. It also can he used to create entities arranged in a cirular pattern (circular arrays) by

creating the entity once and then rotating/copying to the desired positions on the circumference. For a similar condition, rotation can be used to construct axis-symmetric geometric models.

The final position and orientation of an entity are independent of the order of these operations after going through two subsequent commutative translations. Comparing with translation rotation has a unique characteristic since rotation is noncommutative. Two subsequent rotations of the entity about two different axes produce two different rotating results of the entity depending on the order of the rotation.

For example, moving two inches along the (+)Y-axis and then three inches along the (+)X-axis will put the point at the same position as moving three inches along the (+)X-axis and then two inches along the (+)Y-axis. On the contrary, rotating thirty degrees along X-axis then rotating twenty degrees along Y-axis will not get the same orientation as rotating twenty degrees along the Y-axis then rotating thirty degrees along the X-axis.

Therefore, different rotating controls have been developed. To select a viewpoint for a 3D object, we should provide some control. Rotation needs orientations. There isn't a single way of entering orientations. Various different control methods with different advantages and disadvantages have been developed. Here we compare these methods and explain how they relate to rotations and conversion with the rotation matrix. Later we will discuss how to use mouse to generate rotation control.

Euler Angles

This is by far the simplest method to implement orientation. Commonly we think yaw, pitch and roll is the best way to describe 3D rotation. But these parameters influence each other and the order of these computations create different results. For each axis, there is a value specifying the rotation around the axis. Therefore, we have three variables x, y, and z angles to rotate around the global coordinate axis that vary between 0 and 360 degrees (or 0- 2π). They are the roll, pitch, and yaw representation. Orientation is obtained by multiplying the three rotation matrices generated from the three angles together (in a specific order that is defined by user).

The rotations are specified with respect to the global coordinate axis coordinate. This means the first rotation does not change the axis of rotation for the second and third rotations.

Rotations about the coordinate axes are easy to define. Rotation about the x-axis by angle θ is

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{3.6.5}$$

where $\theta > 0$ indicates a counterclockwise rotation in the plane $x = 0$. Here we can assume the viewer positioned on the side of the plane with x ¿ 0 and looking at the origin. Rotation about the y-axis by angle θ is

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.6.6}$$

where $\theta > 0$ indicates a counterclockwise rotation in the plane $y = 0$. The observer is assumed to the positioned on the side of the plane with $z > 0$ and looking at the origin. Rotation about the z-axis by angle θ is

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.6.7}$$

where $\theta > 0$ indicates a counterclockwise rotation in the plane $z = 0$. The observer is assumed to the positioned on the side of the plane with U and looking at the origin.

A common problem is to factor a rotation matrix as a product of rotations about the coordinate axes. The form of the factorization depends on the needs of the application and what ordering is specified. For example, we might want to factor a rotation as $R = R_x(\theta_x)R_y(\theta_y)R_z(\theta_z)$ for some angles θ_x,θ_y and θ_z. The ordering is xyz. Five other possibilities are xzy, yxz, yzx, zxy and zyx. We might also extend factorizations such as xyx not discussed here. In the following discussion, we use the notation $c_a = \cos(\theta_a)$ and $s_a = \sin(\theta_a)$ for $a = x, y, z$.

Factor as $R_x R_y R_z$, setting $R = 0 \leq i \leq 2$ and $0 \leq j \leq 2$, formally multiplying $R_x(\theta_x)R_y(\theta_y)R_z(\theta_z)$, and equating yields

$$\begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} = \begin{bmatrix} c_y c_z & -c_y s_z & s_y \\ c_z s_x s_y + c_y c_z & c_x c_z - s_x s_y s_z & -c_y s_x \\ -c_x c_z s_y + s_x s_z & c_z s_x + c_x s_y s_z & c_x c_y \end{bmatrix} \tag{3.6.8}$$

We also can convert the rotation matrix to Euler angle. From Eq (3.6.8) we have $s_y = r_{02}$, so $\theta_y = \sin^{-1}(r02)$. If $\theta_y \in (-\pi/2, \pi/2)$, then $c_y \neq 0$ and $c_y(s_x, c_x) = (-r_{12}, r_{22})$, in which case $\theta_x = \tan^{-1}(-r_{12}, r_{22})$. Similarly, $c_y(s_z, c_z) = (-r_{01}, r_{00})$, in which case $\theta_z = \tan^{-1}(-r_{01}, r_{00})$.

If $\theta_y = \pi/2$, then $s_y = 1$ and $c_y = 0$. we can get

$$\begin{bmatrix} r_{10} & r_{11} \\ r_{20} & r_{21} \end{bmatrix} = \begin{bmatrix} -c_z s_x + c_x s_z & c_x c_z + c_x s_z \\ c_x c_z + s_x s_z & c_z s_x + c_x s_z \end{bmatrix} = \begin{bmatrix} \sin(\theta_z + \theta_x) & \cos(\theta_z + \theta_x) \\ -\cos(\theta_z + \theta_x) & \sin(\theta_z + \theta_x) \end{bmatrix} \tag{3.6.9}$$

Therefore, $\theta_z + \theta_x = \tan^{-1}(r_{10}, r_{11})$. There is one degree of freedom, so the factorization is not unique. One choice is $\theta_z = 0$ and $\theta_x = \tan^{-1}(r_{10}, r_{11})$. If $\theta_y = -\pi/2$, then $s_y = -1$ and $c_y = 0$. In this case

$$\begin{bmatrix} r_{10} & r_{11} \\ r_{20} & r_{21} \end{bmatrix} = \begin{bmatrix} -c_z s_x + c_x s_z & c_x c_z + s_x s_z \\ c_x c_z + s_x s_z & c_z s_x - c_x s_z \end{bmatrix} = \begin{bmatrix} \sin(\theta_z - \theta_x) & \cos(\theta_z - \theta_x) \\ \cos(\theta_z - \theta_x) & -\sin(\theta_z - \theta_x) \end{bmatrix} \tag{3.6.10}$$

Therefore, $\theta_z - \theta_x = \tan^{-1}(r_{10}, r_{11})$. There is one degree of freedom, so the factorization is not unique. One choice is $\theta_z = O$ and $\theta_x = -\tan^{-1} 2(r_{10}, r_{11})$

Rotation about an arbitrary axis

In the general condition, rotation is not constrained to the XY plane and the axis of rotation may be oriented in any direction. So the direction of the axis can involve the rotation matrix in addition to the angle of rotation. If we define the orientation by the unit, vector $\mathbf{P}^*$ can be written as

$$P^* = f(P, n, \theta) \tag{3.6.11}$$

In this equation, it is assumed that the axis of rotation passes through the origin. Here we only discussed the case where the axis passes through the origin. The axis is in an arbitrary location.

Figure (3.18) shows the three-dimensional rotation of a point P making an angle θ about an arbitrary axis that passes through the origin. The positions of the point before and after rotation are P and P^*, respectively. The orientation of the axis of rotation is defined by the unit vector v such that

$$\vec{n} = n_x\vec{i} + n_y\vec{j} + n_z\vec{k} = \cos\alpha\vec{i} + \cos\beta\vec{j} + \cos\gamma\vec{k} \tag{3.6.12}$$

where $n_x = \cos\alpha$, $n_y = \cos\beta$ and $n_z = \cos\gamma$ are the direction cosines of $\vec{n}$. If the axis of rotation is defined as a line connecting the origin O and any point, **A**, then $n_x = x_A/|A|, n_y = y_A/|A|, n_z = z_A/|A|$, where x_A,y_A and z_A are the coordinates of point A and $|A| = \sqrt{x_A^2 + y_A^2 + z_A^2}$.

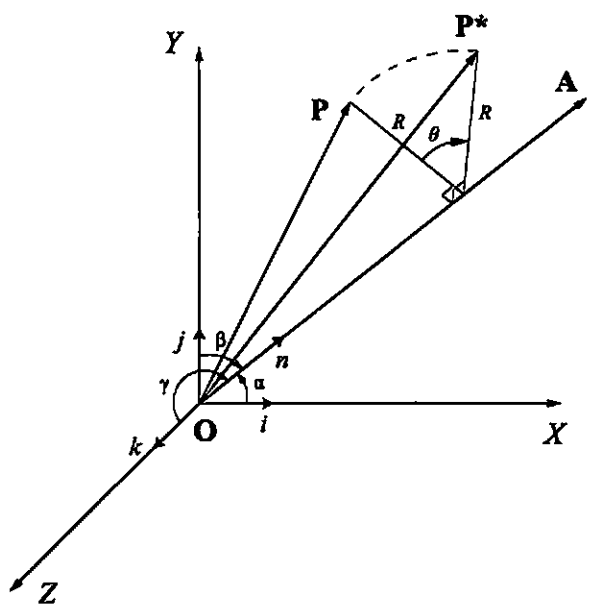


Figure 3.19: Three-dimensional rotation of a point about an arbitrary axis

The rotation of **P** about the axis **OA** defines a circle whose plane is perpendicular to **OA**. Its center is point Q which is the intersection between the axis and the plane. Its radius is R which is the perpendicular distance between **P** and **OA** in any position,

here, $R = \mathbf{PQ} = \mathbf{P^*Q}$. The angle of rotation θ in Fig,(3-18) to be positive according to the agreed-upon convention adopted for two-dimensional rotation. View A-A shows θ counterclockwise, that is, positive, if the observer is placed at A-A, that is, on the positive portion of the axis. In order to facilitate the development, let us define the directions of the lines PQ and P*Q by the unit vectors r and s respectively, as shown in Fig.(3.11) From the figure. it is obvious that the final position vector P* of point P is the resultant of three vectors, that is,

$$\mathrm{P^* = P + PQ + QP^*} \tag{3.6.13}$$

where the notation PQ indicates a vector going from point P to point Q. Using r,s, and R, Eq (3.6.13) can be written as

$$P^* = P + R\vec{r} - R\vec{s} \tag{3.6.14}$$

Considering the equation need we want to express $\vec{r}$ and $\vec{s}$ in terms of P, n, θ. From the triangle OPQ, we can write

$$PQ = Q - P \tag{3.6.15}$$

Since Q is the component of P along the axis of rotation. we can write

$$Q = (P \cdot \vec{n})\vec{n} \tag{3.6.16}$$

Substituting Eq (3.6.15) into (3.6.16) and dividing the result by R (the magnitude of PQ) gives

$$\vec{r} = \frac{(P \cdot \vec{n})\,\vec{n} - P}{R} \tag{3.6.17}$$

In order to express $\vec{r}$ in terms of P and $\vec{n}$, here we introduce the unit vector $\vec{m}$ shown in Fig. (3.11). The vector is chosen to be perpendicular to r and lies in the plane of the circle. Also it is also perpendicular to $\vec{n}$. Utilizing the cross-product definition of two vectors, we can write

$$\vec{m} = \vec{n} \otimes \vec{r} \tag{3.6.18}$$

The unit vectors $\vec{s}$ can now also be written in terms of its components in the $\vec{r}$ and $\vec{m}$. directions as

$$\vec{s} = \cos\theta\vec{r} + \sin\theta\vec{m} \tag{3.6.19}$$

Substituting Eq (3.6.18) into (3.6.19) and substituting the result together with Eq (3.6.14) into (3.6.17), we can get

$$P^* = (P\cdot\vec{n})\,\vec{n} + [P - (P\cdot\vec{n})\,\vec{n}]\cos\theta + (\vec{n}\otimes P)\sin\theta - \vec{n}\otimes(P\cdot\vec{n})\,\vec{n}\sin\theta \tag{3.6.20}$$

The last value in the above equation is equal to zero since the vector $\vec{n}$ and Q are collinear($Q = (P\cdot\vec{n})\vec{n}$) . This result in

$$P^* = (P\cdot\vec{n})\,\vec{n} + [P - (P\cdot\vec{n})\,\vec{n}]\cos\theta + (\vec{n}\otimes P)\sin\theta \tag{3.6.21}$$

writing the Eq (3.6.21) in matrix form, here

$$[P\cdot\vec{n} = xn_x + yn_y + zn_z = \begin{bmatrix} n_x & n_y & n_z \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.6.22}$$

$$\begin{aligned}\vec{n}\otimes P &= \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ n_x & n_y & n_z \\ x & y & z \end{vmatrix} = (n_y z - n_z y)\,\vec{i} + (n_z x - n_x z)\,\vec{j} + (n_x y - n_y x)\,\vec{k} \\ &= \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix}\end{aligned} \tag{3.6.23}$$

and substituting Eqs (3.3.22) and (3.6.23) into Eq (3.6.21), we can get

$$P^* = \left\{(1-\cos)\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}\begin{bmatrix} n_x & n_y & n_z \end{bmatrix} + \cos\theta\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \sin\theta\begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix}\right\}\begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.6.24}$$

When we need to represent rotations using either an angle-axis pair or a rotation matrix, it is necessary to convert from one representation to the other. The conversions are discussed here.

Rotation Matrix to Angle-Axis The inverse problem is to start with the rotation matrix and extract an angle and unit-length axis. There are multiple solutions since $-\vec{U}$ is a valid axis whenever is$\vec{U}$ and $\theta + 2\pi k$ is a valid solution whenever θ is. First, the *trace* of a matrix is defined to be the sum of the diagonal terms. Some algebra will show that $\cos\theta = (trace(R) - 1)/2$ and $R - R^T = (2\sin\theta)S$. The first formula can be solved for the angle, $\theta = \cos^{-1}((trace(R) - 1)/2) \in [0, \pi]$. If $\theta = 0$, then any axis is valid since there is no rotation. If $\theta \in (0, \pi)$, the second formula allows direct extraction of the axis, $\vec{V} = (r_{21} - r_{12}, r_{02} - r_{20}, r_{10} - r_{01})$ and $\vec{U} = \vec{V}/|\vec{V}|$. If $\theta = \pi$, the second formula does not help with the axis since $R - R^T = 0$. In this case note that

$$R = I + 2S^2 = \begin{bmatrix} 1 - 2(n_y^2 + n_z^2) & 2u_0 n_y & 2n_x n_z \\ 2n_x n_y & 1 - 2(n_x^2 + n_z^2) & 2n_y n_z \\ 2n_x n_z & 2n_y n_z & 1 - 2(n_x^2 + n_y^2) \end{bmatrix} \tag{3.6.25}$$

The idea now is to extract the maximum component of the axis from the diagonal entries of the rotation matrix. If r_{00} is maximum, then n_x must be the largest component in magnitude. Compute $4n_x^2 = r_{00} - r_{11} - r_{22} + 1$ and select $n_x = \sqrt{r_{00} - r_{11} - r_{22} + 1}/2$. Consequently, $n_y = r_{01}/(2n_x)$ and $n_z = r_{02}(2n_x)$. If r_{11} is maximum, then compute $4n_y^2 = r_{11} - r_{00} - r_{22} + 1$ and select $n_y = \sqrt{r_{11} - r_{00} - r_{22} + 1}/2$. Consequently, $n_x = r_{01}/(2n_y)$ and $n_z = r_1 2/(2n_y)$. Finally, if r_{22} is maximum, then compute $4n_z^2 = r_{22} - r_{00} - r_{11} + 1$ and select $n_z = \sqrt{r_{22} - r_{00} - r_{11} + 1}/2$. Consequently, $n_x = r_{02}/(2n_z)$ and $n_y = r_1 2/(2n_z)$.

Quaternion

A quaternion is an alternative mathematical entity that 3D graphics programmers use to represent rotations. The use of the quaternion has advantages over the use of rotation matrices in many situations because the quaternion require less storage space, concatenation of quaternion requires fewer arithmetic operations, and a quaternion are more easily interpolated for producing smooth animation.The ideas are based on Shoemake(1987).

Quaternion's allow you to describe a 3D viewpoint with 4 parameters. Like Rotation about an arbitrary axis, this technique allows you to view an object from any possible viewpoint.

A unit quaternion $q = \cos\theta + \sin\theta$ represents the rotation of the 3D vector by an angle 2θ about the 3D axis $\mathbf{u}$. The rotated vector, represented as a quaternion, is $R(\mathbf{v}) = qvq^*$.

Elevation and Azimuth

This method specifies a viewpoint based on the elevation and azimuth of the viewer. The two independent parameters of elevation and azimuth together define a dependent unit vector. This method is similar to the axis rotation. But we can think Elevation and Azimuth method is controlling the viewer's position with respect the object. Axis rotation uses the object centered frame as reference.

Rotation using mouse

A mouse only can provide two values of x and y coordinates. If we want to use a mouse to control the view, we only can use two variables to fit the rotation parameters. Here we will compare these rotation methods under mouse control.

First we discuss Euler rotation. These rotation is connected to three variable θ_x, θ_y,and θ_z. User are familiar with yaw, pitch and roll, but these three parameters interfere with each other. Different order will result in different rotation results. Each mouse movement only provide one value to one of these three variables. It means that we may need to set a variable three time to finish a complete Euler rotation. Its another disadvantage is that when the first rotation angle is equal to 90 degrees, gimbal lock occurs.

Quaternion rotation can be intuitive for the user. Basic method is click somewhere on the surface of the sphere and rotate the ball sphere around. Mouse movement point can be thought the point on sphere project on the x, y axis plane. See figure (3.19).

Elevation and Azimuth method have some advantages: intuitive for the user; Only need two parameters; the parameters do not interact with each other. But its serious problem is some viewpoints are impossible.

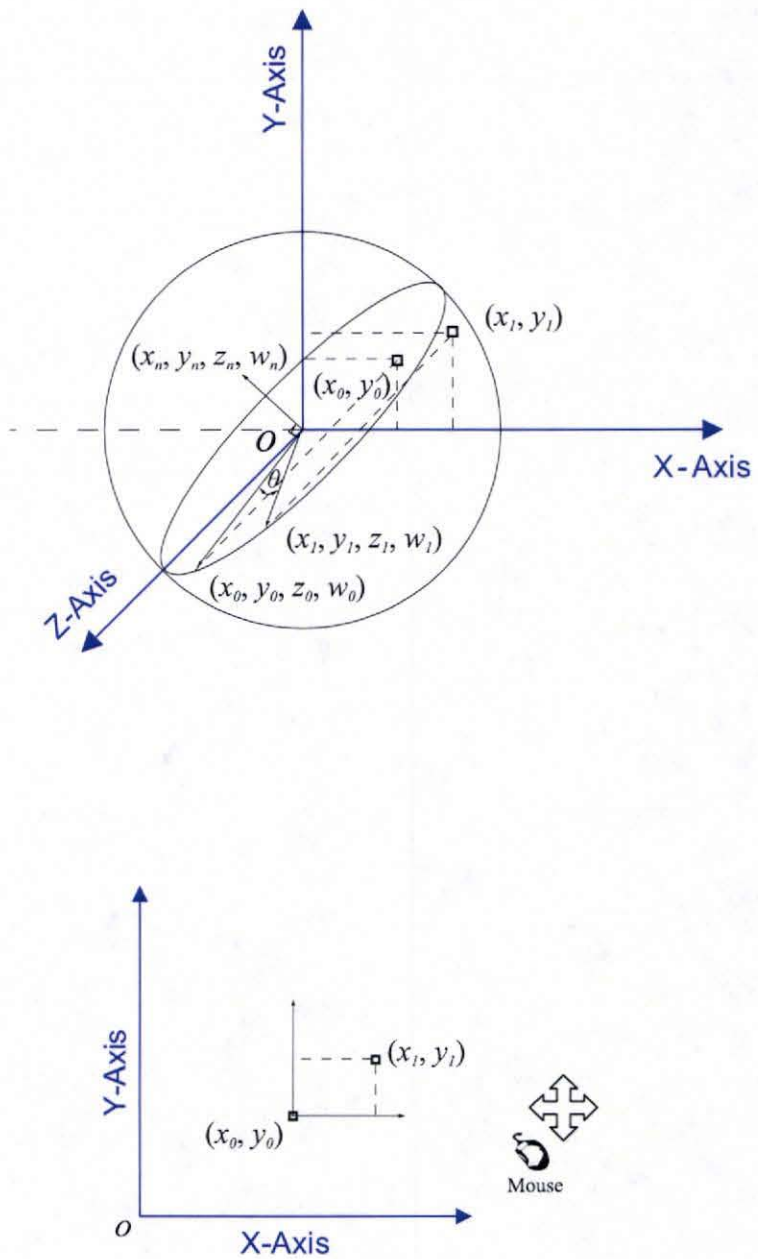


Figure 3.20: Mouse motion for quaternion rotation

Axis rotation is specified by four parameters. Three of them provide the axis vector and another one determinte the rotation degree. Although when mouse move, we only know X,Y coordinate on screen.

Here we can assume the sphere rotates in the x-y axis plane. So rotation axis always remains in the x,y plane. The tangent rotation direction is project on the x,y axis plane according be the mouse moving direction, see the figure (3.20) During the mouse movement, we can know the start point (x_0, y_0) and end point (x_1, x_1). $\vec{m} = \left(\frac{\Delta x}{\sqrt{\Delta x^2+\Delta y^2}}, \frac{\Delta y}{\sqrt{\Delta x^2+\Delta y^2}}, 0\right)$. When $\vec{m}$ rotate 90 degree counterclockwise, we can get $\vec{n}$ which is perpendicular to the $\vec{m}$.

$$\vec{n} = \left(-\frac{\Delta y}{\sqrt{\Delta x^2 + \Delta y^2}}, \frac{\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}}, 0\right) \tag{3.6.26}$$

This distance between the start point and end point can be the rotation arc length, if we know the rotation radius r. We can calculate the rotation angle, θ. In our application we choose the short side of the screen as the rotation radius.

$$\theta = \sqrt{\Delta x^2 + \Delta y^2} \times 180/2\pi r \tag{3.6.27}$$

Compared with these three rotation methods, we find the axis-rotation has more advantage than other two method. In our application we choose axis-rotation.

3.6.3 Homogeneous Representation

The various rigid-body geometric transformations have been developed in the previous section. Equations (3.6.4) and (3.6.24) represent translation and rotation respectively. While the last three equations are in the form of matrix multiplication, translation takes the form of vector addition. This makes it inconvenient to concatenate transformations involving translation. Equation s an example. It is desirable, therefore, to express all geometric translations in the form of matrix multiplications only. Representing points by their homogeneous coordinates provides an effective way to unify the description of geometric transformations as matrix multiplications.

Homogeneous coordinates have been used in computer graphics and geometry for a long time. With their aid, geometric transformations are led into graphics hardware

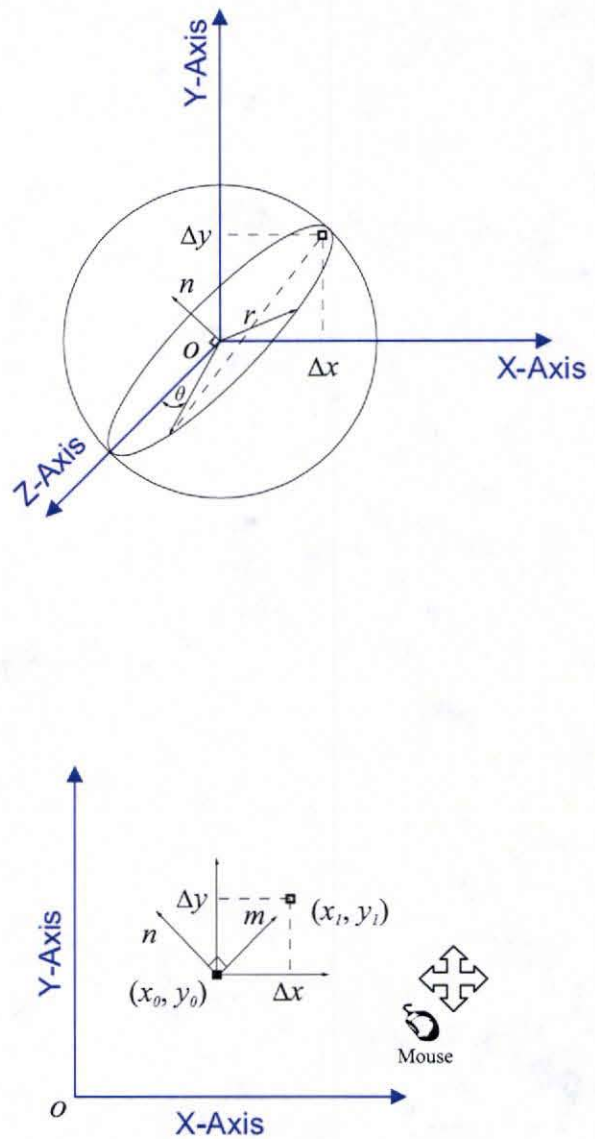


Figure 3.21: Mouse motion for axis rotation

to speed their execution. Homogeneous coordinate are useful for other applications. They are useful to obtain geometric models. The subjects of projective geometry, mechanism analysis and design, and robotics utilize them quite often in development and formulation. In addition, homogeneous coordinates remove many encountered in Cartesian geometry such as representing points at infinity and the non-intersection of parallel lines. Also, they greatly simplify defining rational parametric curves and surfaces.

For homogeneous coordinates, a three-dimensional space is mapped into four dimensional space. A point P with cartesian coordinates (x,y,z) has the homogeneous co-ordinates (x*,y*,z*,h) where h can be any factor except zero. The two types of coordinates other by the following equations:

$$x = \frac{x^*}{h}, y = \frac{y^*}{h}, z = \frac{z^*}{h} \tag{3.6.28}$$

For the purpose of geometric transformations, the scalar factor h used in Eq (??) is taken to be unity to avoid unnecessary divisions.

The translation transformation given by Eq (3.6.3) can now be written as a matrix multiplication by adding the component of 1 to each vector in the equation and using a 4 x 4 matrix as follows:

$$\begin{bmatrix} x^* & y^* & z^* & 1 \end{bmatrix}^T = \begin{bmatrix} 1 & 0 & 0 & x_d \\ 0 & 1 & 0 & y_d \\ 0 & 0 & 1 & z_c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.6.29}$$

and the rotation matrix becomes

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6.30}$$

A closer look at the transformation matrices given in Eq (3.6.29) and (3.6.30) shows that they can all be embedded into one 4 x 4 matrix. This matrix takes the form :

$$T = \left[\begin{array}{ccc|c} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ \hline t_{41} & t_{42} & t_{43} & t_{44} \end{array}\right] = \left[\begin{array}{c|c} T_1 & T_2 \\ \hline T_3 & T_4 \end{array}\right] \tag{3.6.31}$$

The 3×3 submatrix $[T_1]$ produces rotation. The 3×1 column matrix $[T_2]$ generates translation. The 1×3 row matrix $[T_3]$ produces a perspective projection. The fourth diagonal element is the homogeneous-coordinates scalar factor h used in Eq (3.6.28) and is chosen to be unity, as mentioned earlier.

Eq (3.6.28) gives the explicit form of the transformation matrix [T] used in Eq (3.6.2). It is usually written for one geometric transformation at a time by using Eqs (3.6.29) and (3.6.30). If more than one transformation is desired, the resulting matrices are multiplied to produce the total transformation, as discussed in Sec. 3.6.4 that follows.

While the homogeneous representation and the resulting transformation matrix [T] given by Eq (3.6.28) are useful and convenient to write compact equations. We can set corresponding matrix array then use OpenGL command glloadMatrix to perform the transformation.

3.6.4 Concatenated Transformations

In the last section we concentrated on one-step transformations of points such as rotating or translating a point. However, in practice a series of transformations may be applied to a geometric model. Thus, combining or concatenating transformations are quite useful. Concatenated transformations are simply obtained by multiplying the [T] matrices Eq (??) of the corresponding individual transformations. Because matrix multiplication may not be commutative for all cases, we should pay attention to the order in which transformations are applied to a given geometric model. In general, if we apply n transformations at point starting with transformation 1, with [T], and ending with transformation n, with $[T_n]$, then the concatenated transformation of the point can be given as followes:

$$P^* = [T_1][T_2]\cdots[T_{n-1}][T_n]P = [T]P \tag{3.6.32}$$

From the Eq (3.6.32), we will use one matrix to indicate the final transformation condition. When we perform the next transformation, we only need use the current matrix multiplication to obtain the new matrix.

3.6.5 View Control with Matrix Translation

In the last section we can learn that all the calculated matrices can be multiplied together to get a final transformation matrix. One can multiply each of the points (represented as a vector of three coordinates) by this matrix, and directly obtain the screen coordinate at which the point must be drawn. The vector can be extended to four dimensions using homogeneous coordinates:

From our cable surface equations, we generate the cable which is always from the original point. When we rotate, the rotation center of the cable is the bottom of cable. That is not convenient to see the cable, so we set three basic transformation procedure in order to rotate the cable along the center of itself.

The first step is to move the cable one half its height in the negative its direction. We can assume the cable is inside a cylinder. The radius of the cylinder is equal to the maximum outer radius of the cable, and the height of the cylinder is equal to the height of the cable. We can move the cable centerline cable match the coordinate original point so that any rotation transformation is always along the center of the cable.

The second thing is to rotate the cable along its center in any position. In the section 3.6.2, we only discuss rotation around the original point. So we need the set the transformation order to reach the target. An easy way is to first rotate the object firstly then make the transformation. The problem is that the user can do any transformation without considering the order of operation Eq (3.6.31).

An efficient solution is to disassemble the final transformation matrix into several order transformation matries. For our case transformation should follow the order: translate cable center to original point $[T_c]$, perform rotation along the original point $[T_r]$, execute translate cable to some position $[T_t]$.

$$[T] = [T_c][T_r][T_t] \tag{3.6.33}$$

The cable will move one half its height along in the negative minus direction. From Eq (3.6.29) we can get matrix $[T_c]$ given by.

$$T_c = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -H/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6.34}$$

From $[T]$ we can derive the rotation matrix $[T_r]$. See Eq (3.6.31), the 3×3 submatrix $[T_1]$ produces a rotation.

$$[T_r] = \begin{bmatrix} [T_1] & [0] \\ [0] & [1] \end{bmatrix} \tag{3.6.35}$$

When we perform a new transformation, we can multiply it by the corresponding transformation matrix. For rotation transformation, a new rotation matrix can multiply rotation matrix $[T_r]$; the new translation matrix can multiply the translation matrix $[T_t]$. After calculating the respective transformation matrices, we multiply these transformation matrices again.

3.6.6 Perspective Projection and Camera Model

In section 3.1.1 we mentioned the object coordinate will be converted into a camera coordinate.

To obtain a perspective view is to place the center of the projection along the Z_v axis of the viewing coordinate system and project it onto the $Z_v = 0$ or X_vY_v plane. Figure 3-10 shows this case. The center of projection C is placed at a distance d (measured along the Z_v axis), it is developed from the trigonometry shown in Fig.3-6. The viewing eye is located at the center. Here a new coordinate system is introduced called the eye coordinate system which is relative to a sight line. Its X_e and Y_e axes are parallel to the X_v and Y_v axes of the viewing coordinate system, and it is a left-handed system. The Z_e axis is taken in the direction of the line of sight. Therefore, point with a larger Z_e values are taken to be further from the viewing eye. The eye coordinate system is useful in zoom, pan and

rotation for the object. The transformation axes of coordinates of points from the viewing coordinate system to eye coordinate system or vice versa can be written as

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6.36}$$

This matrix simply inverts the sign of the z coordinate. In the orthographic views, The eye coordinate system is located at infinity. It is obvious that the eye coordinate system can be replaced by the viewing coordinate system. In this case, points with smaller z values are interpreted as being further from the viewing eye.

The figure shows the perspective projection of point P as point P_v. To find the y_v of P_v, the two similar triangles COP_2 and CP_3P_1 give

$$\frac{y_v}{y} = \frac{d}{d-z} = \frac{1}{1-z/d} \tag{3.6.37}$$

The two similar triangles COP_2 and CP_3P_1 give x_v of p_v as

$$\frac{x_v}{x} = \frac{r_2}{r_1} = \frac{d}{d-z} = \frac{1}{1-z/d} \tag{3.6.38}$$

Rearranging Eq (3.6.37) and Eq (3.6.38) to give y_v and x_v respectively and knowing $z_v = 0$, we can put the result in a homogeneous form as if this equation is expanded it gives $P_v = [\ x \quad y \quad 0 \quad (1-z/d)]^T$. This would require the division of x and$y$ by $(1-z/d)$ to obtain the corresponding cartesian coordinates of these homogeneous coordinates.

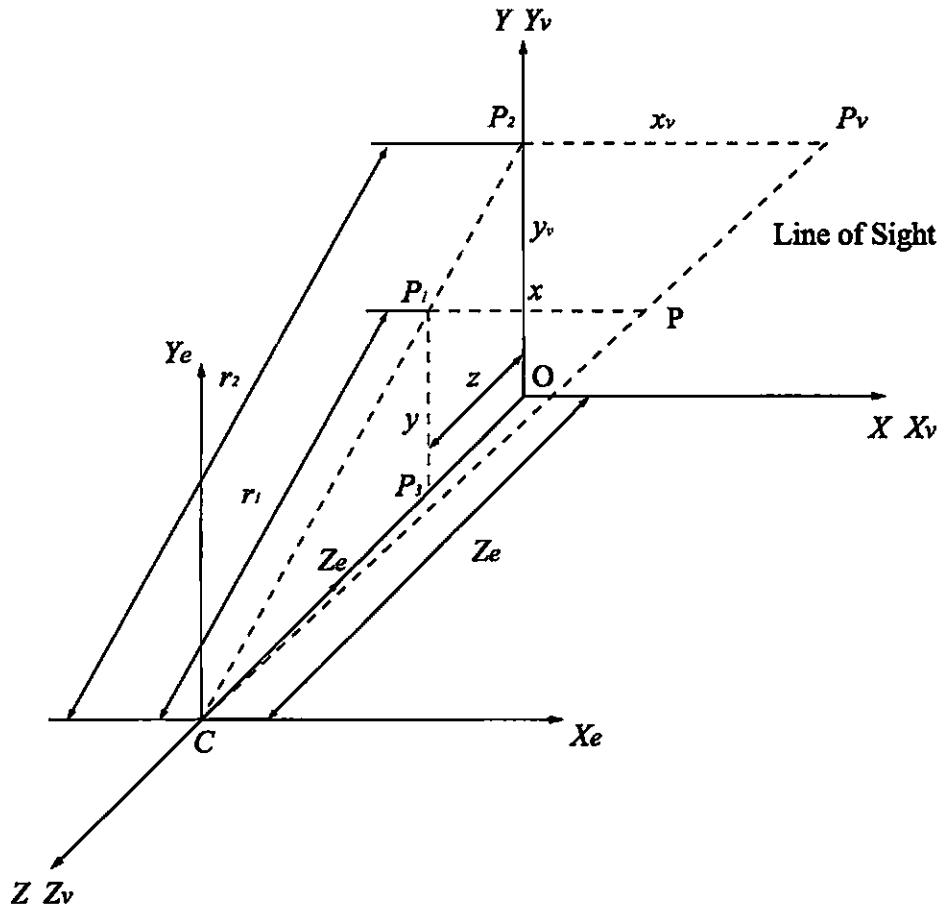


Figure 3.22: Perspective projection along Z_v axis.

Chapter 4

Examples

Simulation of Two Double-Curved Helical Surfaces

By varying the parameter, u_s, in Eq (2.4.6) for a doubly-curved helical surface, a single lay length of a wire strand is generated as depicted in Fig (4.1) compared with an actual strand. Joining six such strands produces the IWRC 6x31 rope depicted in Fig (4.2)

Fig (4.3) illustrates use of the same equation in the previous example to generate a realistic rendering of an actual undersea electrical-optical cable. In this example, use of the double helical surface equation is illustrated with the stranded copper conductors.

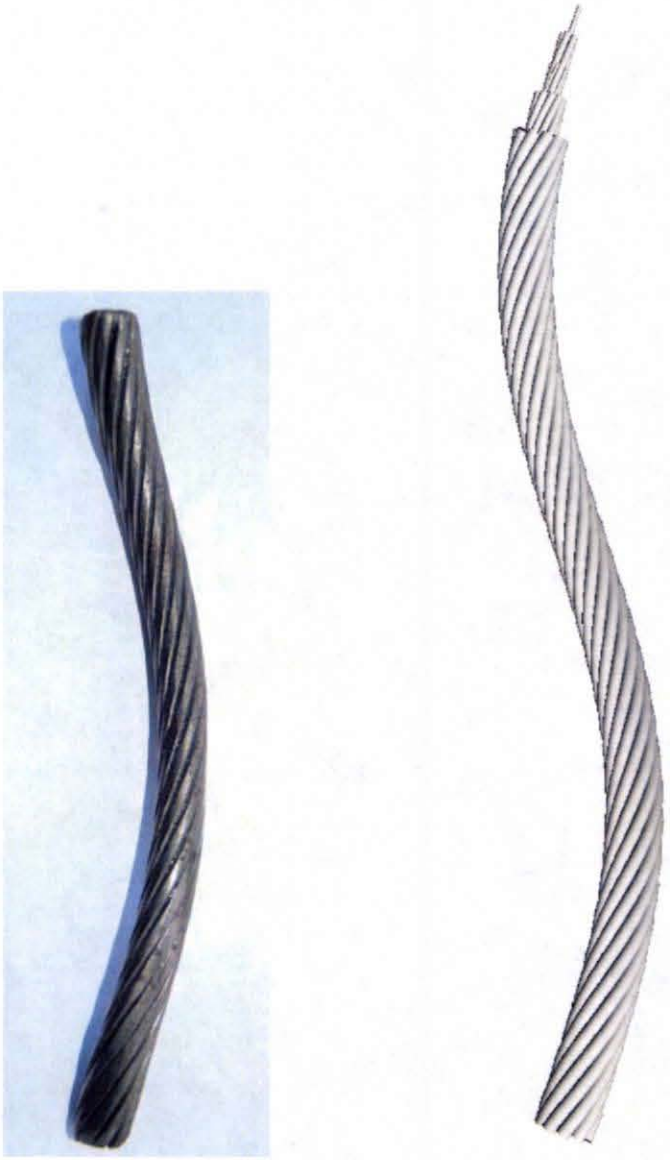


Figure 4.1: Actual steel cable (left) and a simulated doubly-curved helical surface

Figure 4.2: a doubly-curved helical surface

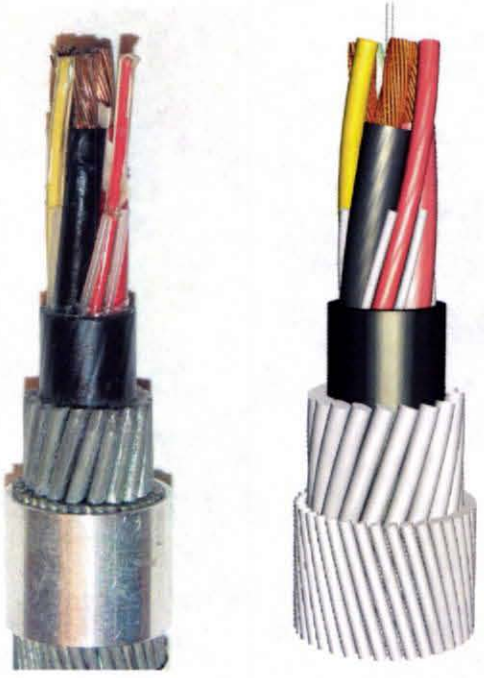


Figure 4.3: actual ROV cable (left) and simulated cable (right)

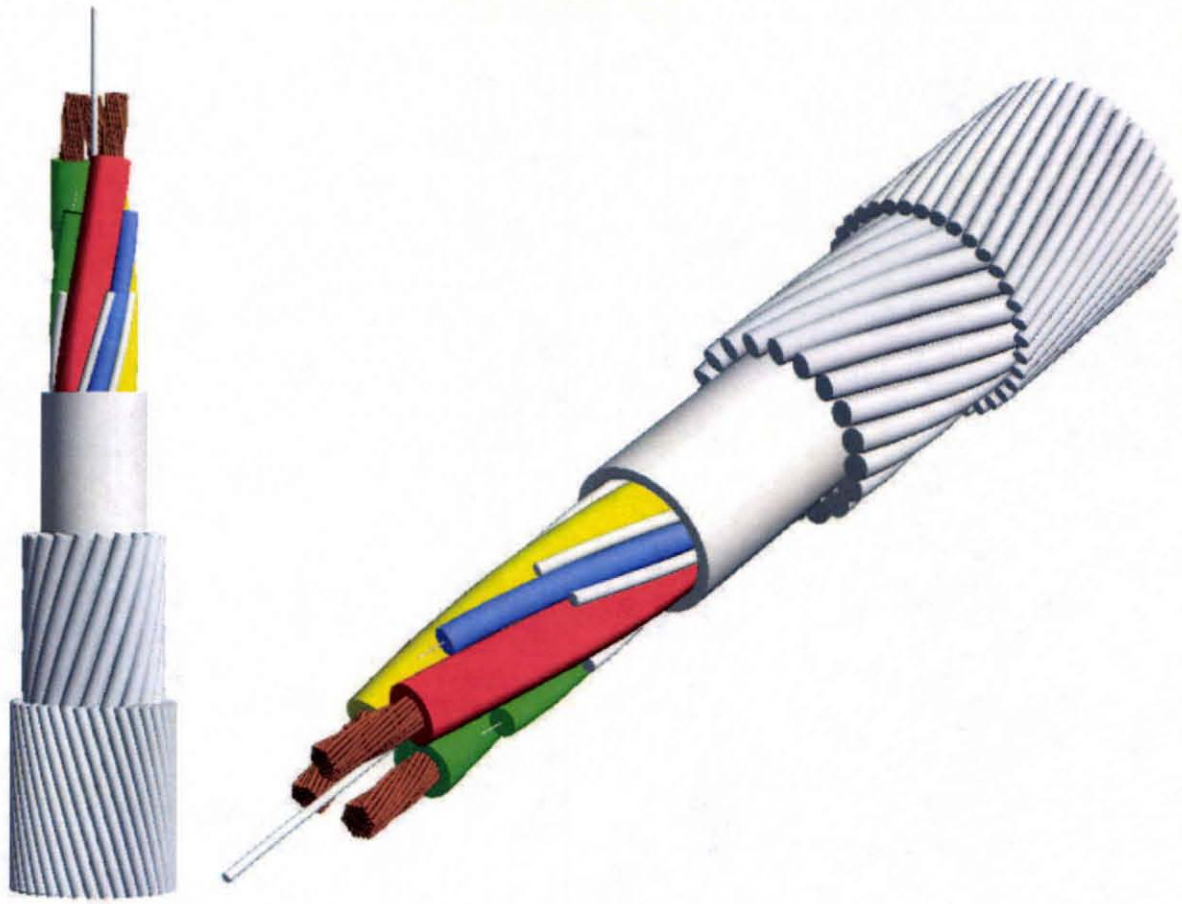


Figure 4.4: ROV cable

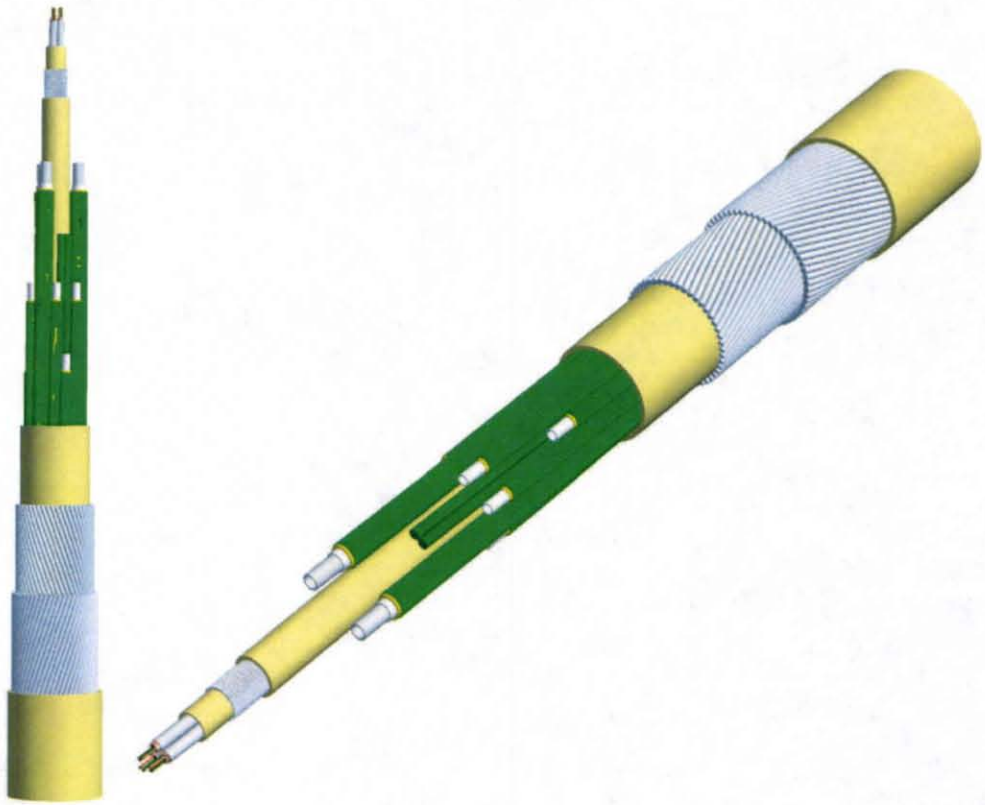


Figure 4.5: Umbilical cable

Chapter 5

Conclusion and Future work

The main challenging of the goal in this thesis was successfully achieved: Develop a mathematical algorithm able to describe the transverse section of a continuous single, double and triple helical components.

The developed model in this investigation can describe exactly the transverse section of double helical constructions, even if the double helix cable has both angles (first and second helix), in the same or opposite directions. A double helix cable may have two possible transverse cross-sections if the components have the same dimensions. These two possible forms depend on the direction of the first and second helices. When both angles have the same direction, the resulting shape is completely different when the helix angles have opposite directions. This model can be used for any case where the transverse section of the components of a double helix can be drawn with regard to the distribution of the center of the double helix cables in the transverse plane.

The model can be applied in a practical way to determine the interstitial gaps needed to be filled with water-blocking materials. Knowing in advance the transverse geometry of the double helical wire rods, production can be planned accurately, since the raw materials needed for specific productions. The key point in this investigation was the mathematical equations derived that describe the helical path of the second helix. With this path, the second important step was using the "Pencil of Spheres" equation together with the helical path. Finally, deriving this equation, representing the envelope of the "Pencil of Spheres", intercepted with the z=0 plane, the transverse section of a double helix cable for each wire that forms the double helical wire rods is obtained.

The parametric model describes the single, double and triple helical cables whose cross section is circle. The model developed in this report fully describes the geometry of the structure of wire ropes of any round-strand construction. It is expressed by vector equations in a three dimensional, right-handed, rectangular Cartesian coordinate system and is general enough that any combination of wire and strand lay directions can be handled if the stated sign conventions for the angles of strand and wire rotation and the relative rotation are followed in the component functions. The wire paths are defined for the first time by using a developed model, which reveal the shapes of the various wires. The geometric properties of each wire can be easily evaluated by using this model. Right regular lay wire rope was analyzed to illustrate the model's usefulness. A system of equations was also established for determining the structural parameters of the deformed rope at a given rope strain, with restrained ends, thus obtaining the model for the deformed rope. The geometric properties of each deformed wire can be evaluated the same way as shown in this report for the unreformed rope.

For future work it is recommended that stress analysis be conducted based on the changes of these geometric properties of the deformed wires to determine how the load is distributed among these wires. Furthermore, the model can be used to study the effect of wear and breaking of wires on strength loss for the various round-strand wire ropes used in mine hoisting so that more scientifically based retirement criteria can be established.

Future work

Appendix A

Derive Progress for Single, Double and Triple Helix

In order to simplify equation derivation, it will assume initial parameter u_s will change from 0 to 1 so corresponding initial angle should be 0 to 2π. Finally we can get completely single helical centerline parametric equation.

$$\mathrm{X}(u_s) = \begin{cases} x = R_p \cos[2\pi\lambda_s(u_s + \varphi_s)] \\ y = R_p \sin[2\pi\lambda_s(u_s + \varphi_s)] \\ z = 2\pi u_s R_p / \tan\alpha \end{cases} \tag{A-1}$$

After making sure the centerline parametric equation, we can calculate the Frenet frame system and single helical surface equation. Firstly we calculate the first derivation and second derivation equation.

$$\dot{\mathrm{X}}(u_s) = \begin{cases} \frac{dx}{du_s} = -2\pi\lambda_s R_p \sin[2\pi\lambda_s(u_s + \varphi_s)] \\ \frac{dy}{du_s} = 2\pi\lambda_s R_p \cos[2\pi\lambda_s(u_s + \varphi_s)] \\ \frac{dz}{du_s} = 2\pi R_p / \tan\alpha \end{cases} \tag{A-2}$$

$$\left\|\dot{\mathrm{X}}(u_s)\right\| = \sqrt{\left(\frac{dx}{du_s}\right)^2 + \left(\frac{dy}{du_s}\right)^2 + \left(\frac{dz}{du_s}\right)^2} = \frac{2\pi R_p}{\sin\alpha} \tag{A-3}$$

$$\vec{T} = \frac{\dot{\mathrm{X}}(u_s)}{\left\|\dot{\mathrm{X}}(u_s)\right\|} = -\sin[2\pi\lambda(u+\varphi)]\sin\alpha\vec{i} - \cos[2\pi\lambda(u+\varphi)]\sin\alpha\vec{j} + \cos\alpha\vec{k} \tag{A-4}$$

$$\ddot{\mathrm{X}}(u_s) = \begin{cases} \frac{d^2x}{du_s^2} = -4\pi^2 R_p \cos[2\pi\lambda_s(u_s+\varphi_s)] \\ \frac{d^2y}{du_s^2} = -4\pi^2 R_p \sin[2\pi\lambda_s(u_s+\varphi_s)] \\ \frac{d^2z}{du_s^2} = 0 \end{cases} \tag{A-5}$$

$$\left\|\dot{\mathrm{X}} \otimes \ddot{\mathrm{X}}\right\| = \frac{8\pi^3 R_p^2}{\sin\alpha} \tag{A-6}$$

$$\dot{\mathrm{X}} \otimes \ddot{\mathrm{X}} = \frac{8\pi^3 R_p^2 \sin[2\pi\lambda_s(u_s+\varphi_s)]}{\tan\alpha}\vec{i} - \frac{8\pi^3 R_p^2 \cos[2\pi\lambda_s(u_s+\varphi_s)]}{\tan\alpha}\vec{j} + 8\pi^3 R_p^2 \lambda_s \vec{k} \tag{A-7}$$

$$\vec{B} = \frac{\dot{\mathrm{X}} \otimes \ddot{\mathrm{X}}}{\left\|\dot{\mathrm{X}} \otimes \ddot{\mathrm{X}}\right\|} = \sin[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha\vec{i} - \cos[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha\vec{j} + \lambda_s \sin\alpha\vec{k} \tag{A-8}$$

$$\vec{N} = \vec{B} \otimes \vec{T} = -\cos[2\pi\lambda_s(u_s+\varphi_s)]\vec{i} - \sin[2\pi\lambda_s(u_s+\varphi_s)]\vec{j} + 0\vec{k} \tag{A-9}$$

A single helix surface S(u,v) based on equation 7 as followed

$$\begin{cases} x = R_p \cos[2\pi\lambda_s(u_s+\varphi_s)] - r\cos 2\pi v \cos[2\pi\lambda_s(u_s+\varphi_s)] + r\sin 2\pi v \sin[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha \\ y = R_p \sin[2\pi\lambda_s(u_s+\varphi_s)] - r\cos 2\pi v \sin[2\pi\lambda_s(u_s+\varphi_s)] - r\sin 2\pi v \cos[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha \\ z = 2\pi u_s R_p / \tan\alpha + r\lambda_s \sin(2\pi v)\sin\alpha \end{cases} \tag{A-10}$$

A double helix centerline

$$\begin{aligned} X(u_s) \;=\; & R_s \cos[2\pi\lambda_s(u_s+\varphi_s)]\vec{i} + R_s \sin[2\pi\lambda_s(u_s+\varphi_s)]\vec{j} + \tfrac{2\pi R_s u_s}{\tan\alpha}\vec{k} \\ & + R_d \cos[2\pi\lambda_d(\tfrac{\tan\beta R_s}{\sin\alpha R_d}u_s + \varphi_d)]\{-\cos[2\pi\lambda_s(u_s+\varphi_s)]\vec{i} - \sin[2\pi\lambda_s(u_s+\varphi_s)]\vec{j}\} \\ & + R_d \sin[2\pi\lambda_d(\tfrac{\tan\beta R_s}{\sin\alpha R_d}u_s + \varphi_d)]\{\sin[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha\vec{i} \\ & - \cos[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha\vec{j} + \lambda_s \sin\alpha\vec{k}\} \end{aligned} \tag{A-11}$$

These equations which are calculating the double helix centerline coordinate as followed

$$
\begin{aligned}
x_d &= R_s\cos[2\pi\lambda_s(u_s+\varphi_s)] - R_d\cos[2\pi\lambda_d(\tan\beta R_s/\sin\alpha R_d u_s+\varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)] \\
&\quad +R_d\sin[2\pi\lambda_d(\tan\beta R_s/\sin\alpha R_d u_s+\varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha
\end{aligned} \tag{A-12}
$$

$$
\begin{aligned}
y_d &= R_s\sin[2\pi\lambda_s(u_s+\varphi_s)] - R_d\cos[2\pi\lambda_d(\tan\beta R_s/\sin\alpha R_d u_s+\varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)] \\
&\quad -R_d\sin[2\pi\lambda_d(\tan\beta R_s/\sin\alpha R_d u_s+\varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha
\end{aligned} \tag{A-13}
$$

$$
z_d = 2\pi R_s u_s/\tan\alpha + R_d\sin[2\pi\lambda_d(\tan\beta R_s/\sin\alpha R_d u_s+\varphi_d)]\lambda_s\sin\alpha \tag{A-14}
$$

The first derived equation about the double helix as followed:

$$
\begin{aligned}
\frac{dx_d}{du_s} &= -2\pi\lambda_s R_s\sin[2\pi\lambda_s(u_s+\varphi_s)] \\
&\quad +\frac{2\pi\lambda_d\tan\beta R_s}{\sin\alpha}\sin[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)] \\
&\quad +2\pi\lambda_s R_d\cos[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)] \\
&\quad +\frac{2\pi\lambda_d\tan\beta R_s}{\sin\alpha}\cos[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha \\
&\quad +2\pi\lambda_s R_d\sin[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha
\end{aligned} \tag{A-15}
$$

$$
\begin{aligned}
\frac{dy_d}{du_s} &= 2\pi\lambda_s R_s\cos[2\pi\lambda_s(u_s+\varphi_s)] \\
&\quad +\frac{2\pi\lambda_d\tan\beta R_s}{\sin\alpha}\sin[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)] \\
&\quad -2\pi\lambda_s R_d\cos[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)] \\
&\quad -\frac{2\pi\lambda_d\tan\beta R_s}{\sin\alpha}\cos[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha \\
&\quad +2\pi\lambda_s R_d\sin[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha
\end{aligned} \tag{A-16}
$$

$$
\frac{dz_d}{du_s} = \frac{2\pi R_s}{\tan\alpha} + 2\pi\lambda_d\lambda_s\tan\beta R_s\cos[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)] \tag{A-17}
$$

The second derived equation about the double helix as followed:

$$
\begin{aligned}
\frac{d^2x_d}{du_s^2} = & -(2\pi)^2 R_s \cos[2\pi\lambda_s(u_s+\varphi_s)] \\
& + \left(\frac{2\pi\tan\beta R_s}{\sin\alpha}\right)^2 \frac{1}{R_d} \cos[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)] \\
& - \frac{(2\pi)^2\lambda_d\lambda_s\tan\beta R_s}{\sin\alpha}\sin[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)] \\
& - \frac{(2\pi)^2\lambda_d\lambda_s\tan\beta R_s}{\sin\alpha}\sin[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)] \\
& + (2\pi)^2 R_d \cos[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)] \\
& - \left(\frac{2\pi\tan\beta R_s}{\sin\alpha}\right)^2 \frac{1}{R_d}\sin[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha \\
& + \frac{(2\pi)^2\lambda_d\lambda_s\tan\beta R_s}{\sin\alpha}\cos[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha \\
& + \frac{(2\pi)^2\lambda_d\lambda_s\tan\beta R_s}{\sin\alpha}\cos[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha \\
& - (2\pi)^2 R_d \sin[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha
\end{aligned}
\tag{A-18}
$$

$$
\begin{aligned}
\frac{d^2y_d}{du_s^2} = & -(2\pi)^2 R_s \sin[2\pi\lambda_s(u_s+\varphi_s)] \\
& + \left(\frac{2\pi\tan\beta R_s}{\sin\alpha}\right)^2 \frac{1}{R_d} \cos[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)] \\
& + \frac{(2\pi)^2\lambda_d\lambda_s\tan\beta R_s}{\sin\alpha}\sin[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)] \\
& + \frac{(2\pi)^2\lambda_d\lambda_s\tan\beta R_s}{\sin\alpha}\sin[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)] \\
& + (2\pi)^2 R_d \cos[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)] \\
& + \left(\frac{2\pi\tan\beta R_s}{\sin\alpha}\right)^2 \frac{1}{R_d}\sin[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha \\
& + \frac{(2\pi)^2\lambda_d\lambda_s\tan\beta R_s}{\sin\alpha}\cos[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha \\
& + \frac{(2\pi)^2\lambda_d\lambda_s\tan\beta R_s}{\sin\alpha}\cos[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha \\
& + (2\pi)^2 R_d \sin[2\pi\lambda_d(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)]\cos\alpha
\end{aligned}
\tag{A-19}
$$

$$
\frac{d^2z_d}{du_s^2} = -\frac{(2\pi\tan\beta R_s)^2\lambda_s}{\sin\alpha R_d}\sin\left[2\pi\lambda_d\left(\frac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d\right)\right] \tag{A-20}
$$

$$
\vec{T}_d(u_s) = \frac{\frac{dx_d}{du_s}\vec{i}+\frac{dy_d}{du_s}\vec{j}+\frac{dz_d}{du_s}\vec{k}}{\sqrt{\left(\frac{dx_d}{du_s}\right)^2+\left(\frac{dy_d}{du_s}\right)^2+\left(\frac{dz_d}{du_s}\right)^2}} \tag{A-21}
$$

$$
\vec{B}_d(u_s) = \frac{\left(\frac{dy_d}{du_s}\frac{d^2z_d}{du_s^2}-\frac{d^2y_d}{du_s^2}\frac{dz_d}{du_s}\right)\vec{i}+\left(\frac{dz_d}{du_s}\frac{d^2x_d}{du_s^2}-\frac{d^2z_d}{du_s^2}\frac{dx_d}{du_s}\right)\vec{j}+\left(\frac{dx_d}{du_s}\frac{d^2y_d}{du_s^2}-\frac{d^2x_d}{du_s^2}\frac{dy_d}{du_s}\right)\vec{k}}{\sqrt{\left(\frac{dy_d}{du_s}\frac{d^2z_d}{du_s^2}-\frac{d^2y_d}{du_s^2}\frac{dz_d}{du_s}\right)^2+\left(\frac{dz_d}{du_s}\frac{d^2x_d}{du_s^2}-\frac{d^2z_d}{du_s^2}\frac{dx_d}{du_s}\right)^2+\left(\frac{dx_d}{du_s}\frac{d^2y_d}{du_s^2}-\frac{d^2x_d}{du_s^2}\frac{dy_d}{du_s}\right)^2}} \tag{A-22}
$$

$$\vec{N}_d(u_s) = \vec{B}_d \otimes \vec{T}_d$$

$$= \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ \frac{\frac{dy_d}{du_s}\frac{d^2x_d}{du_s^2} - \frac{d^2y_d}{du_s^2}\frac{dx_d}{du_s}}{\left\|\dot{X}_d \otimes \ddot{X}_d\right\|} & \frac{\frac{dx_d}{du_s}\frac{d^2x_d}{du_s^2} - \frac{d^2x_d}{du_s^2}\frac{dx_d}{du_s}}{\left\|\dot{X}_d \otimes \ddot{X}_d\right\|} & \frac{\frac{dx_d}{du_s}\frac{d^2y_d}{du_s^2} - \frac{d^2x_d}{du_s^2}\frac{dy_d}{du_s}}{\left\|\dot{X}_d \otimes \ddot{X}_d\right\|} \\ \frac{\frac{dx_d}{du_s}}{\sqrt{\left(\frac{dx_d}{du_s}\right)^2 + \left(\frac{dy_d}{du_s}\right)^2 + \left(\frac{dz_d}{du_s}\right)^2}} & \frac{\frac{dy_d}{du_s}}{\sqrt{\left(\frac{dx_d}{du_s}\right)^2 + \left(\frac{dy_d}{du_s}\right)^2 + \left(\frac{dz_d}{du_s}\right)^2}} & \frac{\frac{dz_d}{du_s}}{\sqrt{\left(\frac{dx_d}{du_s}\right)^2 + \left(\frac{dy_d}{du_s}\right)^2 + \left(\frac{dz_d}{du_s}\right)^2}} \end{vmatrix}$$

$$= \left(\frac{\frac{dz_d}{du_s}\frac{d^2x_d}{du_s^2} - \frac{d^2z_d}{du_s^2}\frac{dx_d}{du_s}}{\left\|\dot{X}_d \otimes \ddot{X}_d\right\|} \frac{\frac{dx_d}{du_s}}{\sqrt{\left(\frac{dx_d}{du_s}\right)^2 + \left(\frac{dy_d}{du_s}\right)^2 + \left(\frac{dz_d}{du_s}\right)^2}} - \frac{\frac{dy_d}{du_s}}{\sqrt{\left(\frac{dx_d}{du_s}\right)^2 + \left(\frac{dy_d}{du_s}\right)^2 + \left(\frac{dz_d}{du_s}\right)^2}} \frac{\frac{dx_d}{du_s}\frac{d^2y_d}{du_s^2} - \frac{d^2x_d}{du_s^2}\frac{dy_d}{du_s}}{\left\|\dot{X}_d \otimes \ddot{X}_d\right\|} \right) \vec{i}$$

$$+ \left(\frac{\frac{dx_d}{du_s}\frac{d^2y_d}{du_s^2} - \frac{d^2x_d}{du_s^2}\frac{dy_d}{du_s}}{\left\|\dot{X}_d \otimes \ddot{X}_d\right\|} \frac{\frac{dx_d}{du_s}}{\sqrt{\left(\frac{dx_d}{du_s}\right)^2 + \left(\frac{dy_d}{du_s}\right)^2 + \left(\frac{dz_d}{du_s}\right)^2}} - \frac{\frac{dz_d}{du_s}}{\sqrt{\left(\frac{dx_d}{du_s}\right)^2 + \left(\frac{dy_d}{du_s}\right)^2 + \left(\frac{dz_d}{du_s}\right)^2}} \frac{\frac{dy_d}{du_s}\frac{d^2z_d}{du_s^2} - \frac{d^2y_d}{du_s^2}\frac{dz_d}{du_s}}{\left\|\dot{X}_d \otimes \ddot{X}_d\right\|} \right) \vec{j}$$

$$+ \left(\frac{\frac{dy_d}{du_s}\frac{d^2z_d}{du_s^2} - \frac{d^2y_d}{du_s^2}\frac{dz_d}{du_s}}{\left\|\dot{X}_d \otimes \ddot{X}_d\right\|} \frac{\frac{dy_d}{du_s}}{\sqrt{\left(\frac{dx_d}{du_s}\right)^2 + \left(\frac{dy_d}{du_s}\right)^2 + \left(\frac{dz_d}{du_s}\right)^2}} - \frac{\frac{dx_d}{du_s}}{\sqrt{\left(\frac{dx_d}{du_s}\right)^2 + \left(\frac{dy_d}{du_s}\right)^2 + \left(\frac{dz_d}{du_s}\right)^2}} \frac{\frac{dz_d}{du_s}\frac{d^2x_d}{du_s^2} - \frac{d^2z_d}{du_s^2}\frac{dx_d}{du_s}}{\left\|\dot{X}_d \otimes \ddot{X}_d\right\|} \right) \vec{k}$$

(A-23)

Triple Helix Centerline

From double helix centerline equation and surface equation we get get general triple helix

$$\mathrm{X}_t(u_s) = \mathrm{X}_d(u_s) + R_t \cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]\vec{N}_d + R_t\left[\sin 2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]\vec{B}_d$$

(A-24)

Assume corresponding double helix :

$$\mathrm{X}_d(u_s) = X_d\vec{i} + Y_d\vec{j} + Z_d\vec{k} \tag{A-25}$$

Here X_d, Y_d and Z_d can be calculated from the equation (A-12 A-14).

Normal direction component and Bionormal direction component also can be assumed:

$$N_d(u_s) = X_n\vec{i} + Y_n\vec{j} + Z_n\vec{k} \tag{A-26}$$

X_n, Y_n and Z_n can be calculated from the equation (A-23).

$$B_d(u_s) = X_b\vec{i} + Y_b\vec{j} + Z_b\vec{k} \tag{A-27}$$

X_b, Y_b and Z_b can be calculated from the equation (A-22).

Now we can divide Xt into three direction component which include x, y and z coordinate

$$
\begin{aligned}
\mathrm{X}_t &= X_d\vec{i} + Y_d\vec{j} + Z_d\vec{k} + R_t\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]\left(X_n\vec{i} + Y_n\vec{j} + Z_n\vec{k}\right) \\
&+ R_t\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]\left(X_b\vec{i} + Y_b\vec{j} + Z_b\vec{k}\right) \\
&= \left\{X_d + R_t\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]X_n + R_t\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]X_b\right\}\vec{i} \\
&+ \left\{Y_d + R_t\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]Y_n + R_t\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]Y_b\right\}\vec{j} \\
&+ \left\{Z_d + R_t\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]Z_n + R_t\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]Z_b\right\}\vec{k}
\end{aligned}
\tag{A-28}
$$

$$
\begin{aligned}
\frac{d\mathrm{X}_t}{du_s} &= \frac{d\left\{X_d + R_t\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]X_n + R_t\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]X_b\right\}}{du_s}\vec{i} \\
&+ \frac{d\left\{Y_d + R_t\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]Y_n + R_t\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]Y_b\right\}}{du_s}\vec{j} \\
&+ \frac{d\left\{Z_d + R_t\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]Z_n + R_t\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]Z_b\right\}}{du_s}\vec{k}
\end{aligned}
\tag{A-29}
$$

x, y, z coordinate first deviated equation as followed

$$
\begin{aligned}
\frac{dx_t}{du_s} &= \frac{dX_d}{du_s} + \frac{d\left(R_t\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]X_n\right)}{du_s} + \frac{d\left(R_t\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]X_b\right)}{du_s} \\
&= \frac{dX_d}{du_s} - \frac{2\pi\lambda_t\tan\gamma R_s}{\cos\beta\sin\alpha}\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]X_n + R_t\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]\frac{dX_n}{du_s} \\
&+ \frac{2\pi\lambda_t\tan\gamma R_s}{\cos\beta\sin\alpha}\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]X_b + R_t\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]\frac{dX_b}{du_s}
\end{aligned}
\tag{A-30}
$$

$$
\begin{aligned}
\frac{dy_t}{du_s} &= \frac{dY_d}{du_s} - \frac{2\pi\lambda_t\tan\gamma R_s}{\cos\beta\sin\alpha}\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]Y_n + R_t\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]\frac{dY_n}{du_s} \\
&+ \frac{2\pi\lambda_t\tan\gamma R_s}{\cos\beta\sin\alpha}\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]Y_b + R_t\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t} + \varphi_t\right)\right]\frac{dY_b}{du_s}
\end{aligned}
\tag{A-31}
$$

$$\frac{dz_t}{du_s} = \frac{dZ_d}{du_s} - \frac{2\pi\lambda_t \tan\gamma R_s}{\cos\beta\sin\alpha}\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]Z_n + R_t\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{dZ_n}{du_s}$$
$$+\frac{2\pi\lambda_t \tan\gamma R_s}{\cos\beta\sin\alpha}\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]Z_b + R_t\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{dZ_b}{du_s} \quad \text{(A-32)}$$

x, y, z coordinate second deviated equation as followed

$$\begin{aligned}
\frac{d^2x_t}{du_s^2} = {} & \frac{d^2X_d}{du_s^2} - \left(\frac{2\pi\lambda_t\tan\gamma R_s}{\cos\beta\sin\alpha}\right)^2\frac{1}{R_t}\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]X_n \\
& -\frac{2\pi\lambda_t\tan\gamma R_s}{\cos\beta\sin\alpha}\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{dX_n}{du_s} \\
& -\frac{2\pi\lambda_t\tan\gamma R_s}{\cos\beta\sin\alpha}\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{dX_n}{du_s} \\
& +R_t\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{d^2X_n}{du_s^2} \\
& -\left(\frac{2\pi\lambda_t\tan\gamma R_s}{\cos\beta\sin\alpha}\right)^2\frac{1}{R_t}\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]X_b \\
& +\frac{2\pi\lambda_t\tan\gamma R_s}{\cos\beta\sin\alpha}\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{dX_b}{du_s^2} \\
& +\frac{2\pi\lambda_t\tan\gamma R_s}{\cos\beta\sin\alpha}\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{dX_b}{du_s} \\
& +R_t\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{d^2X_b}{du_s^2}
\end{aligned} \quad \text{(A-33)}$$

$$\begin{aligned}
\frac{d^2y_t}{du_s^2} = {} & \frac{d^2Y_d}{du_s^2} - \left(\frac{2\pi\lambda_t\tan\gamma R_s}{\cos\beta\sin\alpha}\right)^2\frac{1}{R_t}\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]Y_n \\
& -\frac{2\pi\lambda_t\tan\gamma R_s}{\cos\beta\sin\alpha}\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{dY_n}{du_s} \\
& -\frac{2\pi\lambda_t\tan\gamma R_s}{\cos\beta\sin\alpha}\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{dY_n}{du_s} \\
& +R_t\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{d^2Y_n}{du_s^2} \\
& -\left(\frac{2\pi\lambda_t\tan\gamma R_s}{\cos\beta\sin\alpha}\right)^2\frac{1}{R_t}\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]Y_b \\
& +\frac{2\pi\lambda_t\tan\gamma R_s}{\cos\beta\sin\alpha}\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{dY_b}{du_s^2} \\
& +\frac{2\pi\lambda_t\tan\gamma R_s}{\cos\beta\sin\alpha}\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{dY_b}{du_s} \\
& +R_t\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{d^2Y_b}{du_s^2}
\end{aligned} \quad \text{(A-34)}$$

$$
\begin{aligned}
\frac{d^2 z_t}{du_s^2} = & \frac{d^2 Z_d}{du_s^2} - \left(\frac{2\pi\lambda_t \tan\gamma R_s}{\cos\beta\sin\alpha}\right)^2 \frac{1}{R_t}\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right] Z_n \\
& -\frac{2\pi\lambda_t \tan\gamma R_s}{\cos\beta\sin\alpha}\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{dZ_n}{du_s} \\
& -\frac{2\pi\lambda_t \tan\gamma R_s}{\cos\beta\sin\alpha}\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{dZ_n}{du_s} \\
& +R_t\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{d^2 Z_n}{du_s^2} \\
& -\left(\frac{2\pi\lambda_t \tan\gamma R_s}{\cos\beta\sin\alpha}\right)^2 \frac{1}{R_t}\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right] Z_b \\
& +\frac{2\pi\lambda_t \tan\gamma R_s}{\cos\beta\sin\alpha}\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{dZ_b}{du_s^2} \\
& +\frac{2\pi\lambda_t \tan\gamma R_s}{\cos\beta\sin\alpha}\cos\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{dZ_b}{du_s} \\
& +R_t\sin\left[2\pi\lambda_t\left(\frac{\tan\gamma R_s u_s}{\cos\beta\sin\alpha R_t}+\varphi_t\right)\right]\frac{d^2 Z_b}{du_s^2}
\end{aligned}
\tag{A-35}
$$

From equation (A-30-35) if we want to calculate the values $\frac{dx_t}{du_s}, \frac{dy_t}{du_s}, \frac{dz_t}{du_s}, \frac{d^2x_t}{du_s^2}, \frac{d^2y_t}{du_s^2}, \frac{d^2z_t}{du_s^2}$, we need to know these values

$$
\begin{bmatrix} X_d & X_n & X_b \\ Y_d & Y_n & Y_b \\ Z_d & Z_n & Z_b \end{bmatrix},
\begin{bmatrix} \frac{dX_d}{du_s} & \frac{dX_n}{du_s} & \frac{dX_b}{du_s} \\ \frac{dY_d}{du_s} & \frac{dY_n}{du_s} & \frac{dY_b}{du_s} \\ \frac{dZ_d}{du_s} & \frac{dZ_n}{du_s} & \frac{dZ_b}{du_s} \end{bmatrix},
\begin{bmatrix} \frac{d^2X_d}{du_s^2} & \frac{d^2X_n}{du_s^2} & \frac{d^2X_b}{du_s^2} \\ \frac{d^2Y_d}{du_s^2} & \frac{d^2Y_n}{du_s^2} & \frac{d^2Y_b}{du_s^2} \\ \frac{d^2Z_d}{du_s^2} & \frac{d^2Z_n}{du_s^2} & \frac{d^2Z_b}{du_s^2} \end{bmatrix}.
$$

Through calculating the double helix centerline such as equation (A-12 14) we can get X_d, Y_d, Z_d. From equation (A-15 17) the value of $\frac{dX_d}{du_s}, \frac{dY_d}{du_s}, \frac{dZ_d}{du_s}$ can be calculated. We also can get the values of $\frac{d^2X_d}{du_s^2}, \frac{d^2Y_d}{du_s^2}, \frac{d^2Z_d}{du_s^2}$ using the equation(A-18 20).

To simplify the calculation of $\frac{dN_d}{du_s}, \frac{dB_d}{du_s}, \frac{d^2N_d}{du_s^2}$ and $\frac{d^2B_d}{du_s^2}$. Here we assume

$$B_d(u_s) = \frac{f(u_s)}{g(u_s)} \tag{A-36}$$

$$N_d(u_s) = \frac{m(u_s)}{n(u_s)} \tag{A-37}$$

$$f(u_s) = \left(\frac{dy_d}{du_s}\frac{d^2z_d}{du_s^2} - \frac{d^2y_d}{du_s^2}\frac{dz_d}{du_s}\right)\vec{i} + \left(\frac{dz_d}{du_s}\frac{d^2x_d}{du_s^2} - \frac{d^2z_d}{du_s^2}\frac{dx_d}{du_s}\right)\vec{j} + \left(\frac{dx_d}{du_s}\frac{d^2y_d}{du_s^2} - \frac{d^2x_d}{du_s^2}\frac{dy_d}{du_s}\right)\vec{k} \tag{A-38}$$

$$g(u_s) = \sqrt{\left(\frac{dy_d}{du_s}\frac{d^2z_d}{du_s^2} - \frac{d^2y_d}{du_s^2}\frac{dz_d}{du_s}\right)^2 + \left(\frac{dz_d}{du_s}\frac{d^2x_d}{du_s^2} - \frac{d^2z_d}{du_s^2}\frac{dx_d}{du_s}\right)^2 + \left(\frac{dx_d}{du_s}\frac{d^2y_d}{du_s^2} - \frac{d^2x_d}{du_s^2}\frac{dy_d}{du_s}\right)^2} \tag{A-39}$$

$$
\begin{aligned}
m(u_s) = & \left[\left(\frac{dz_d}{du_s}\frac{d^2x_d}{du_s^2} - \frac{d^2z_d}{du_s^2}\frac{dx_d}{du_s}\right)\frac{dz_d}{du_s} - \frac{dy_d}{du_s}\left(\frac{dx_d}{du_s}\frac{d^2y_d}{du_s^2} - \frac{d^2x_d}{du_s^2}\frac{dy_d}{du_s}\right)\right]\vec{i} \\
+ & \left[\left(\frac{dx_d}{du_s}\frac{d^2y_d}{du_s^2} - \frac{d^2x_d}{du_s^2}\frac{dy_d}{du_s}\right)\frac{dx_d}{du_s} - \frac{dz_d}{du_s}\left(\frac{dy_d}{du_s}\frac{d^2z_d}{du_s^2} - \frac{d^2y_d}{du_s^2}\frac{dz_d}{du_s}\right)\right]\vec{j} \\
+ & \left[\left(\frac{dy_d}{du_s}\frac{d^2z_d}{du_s^2} - \frac{d^2y_d}{du_s^2}\frac{dz_d}{du_s}\right)\frac{dy_d}{du_s} - \frac{dx_d}{du_s}\left(\frac{dz_d}{du_s}\frac{d^2x_d}{du_s^2} - \frac{d^2z_d}{du_s^2}\frac{dx_d}{du_s}\right)\right]\vec{k}
\end{aligned}
\tag{A-40}
$$

$$
\begin{aligned}
n(u_s) = & \sqrt{\left(\frac{dx_d}{du_s}\right)^2 + \left(\frac{dy_d}{du_s}\right)^2 + \left(\frac{dz_d}{du_s}\right)^2} \\
\times & \sqrt{\left(\frac{dy_d}{du_s}\frac{d^2z_d}{du_s^2} - \frac{d^2y_d}{du_s^2}\frac{dz_d}{du_s}\right)^2 + \left(\frac{dz_d}{du_s}\frac{d^2x_d}{du_s^2} - \frac{d^2z_d}{du_s^2}\frac{dx_d}{du_s}\right)^2 + \left(\frac{dx_d}{du_s}\frac{d^2y_d}{du_s^2} - \frac{d^2x_d}{du_s^2}\frac{dy_d}{du_s}\right)^2}
\end{aligned}
\tag{A-41}
$$

for $\frac{dB_d(u_s)}{du_s}$ we know

$$
\frac{dB_d(u_s)}{du_s} = \frac{f'(u_s)g(u_s) - f(u_s)g'(u_s)}{g^2(u_s)} \tag{A-42}
$$

$$
\frac{d^2B_d(u_s)}{du_s^2} = \frac{\left[f''(u_s)g(u_s) - f(u_s)g''(u_s)\right]g^2(u_s) - 2\left[f'(u_s)g(u_s) - f(u_s)g'(u_s)\right]g(u_s)g'(u_s)}{g^4(u_s)} \tag{A-43}
$$

Now we need to calculate $\frac{d^3x_d}{du_s^3}, \frac{d^3y_d}{du_s^3}, \frac{d^3z_d}{du_s^3}, \frac{d^4x_d}{du_s^4}, \frac{d^4y_d}{du_s^4}, \frac{d^4z_d}{du_s^4}$ to solve $\frac{df(u_s)}{du_s}, \frac{d^2f(u_s)}{du_s^2}$. From equation (A-18) to (A-19) we can assume as following:

$$
\begin{aligned}
\frac{d^2x_d}{du_s^2} = & \; a_0\cos[2\pi\lambda_s(u_s+\varphi_s)] \\
& + a_1\cos[2\pi\lambda_d(\tfrac{\tan\beta R_s}{\sin\alpha R_d}u_s + \varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)] \\
& + a_2\sin[2\pi\lambda_d(\tfrac{\tan\beta R_s}{\sin\alpha R_d}u_s + \varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)]
\end{aligned}
\tag{A-44}
$$

here

$$
a_0 = -(2\pi)^2 R_s
$$

$$
a_1 = \left(\frac{2\pi\tan\beta R_s}{\sin\alpha}\right)^2\frac{1}{R_d} + (2\pi)^2 R_d + \frac{8\pi^2\lambda_s\lambda_d\tan\beta\cos\alpha R_s}{\sin\alpha}
$$

$$
a_2 = -\left(\frac{2\pi\tan\beta R_s}{\sin\alpha}\right)^2\frac{\cos\alpha}{R_d} - (2\pi)^2 R_d\cos\alpha - \frac{8\pi^2\lambda_s\lambda_d\tan\beta R_s}{\sin\alpha}
$$

$$
\begin{aligned}
\frac{d^2y_d}{du_s^2} = & \; b_0\sin[2\pi\lambda_s(u_s+\varphi_s)] \\
& + b_1\cos[2\pi\lambda_d(\tfrac{\tan\beta R_s}{\sin\alpha R_d}u_s + \varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)] \\
& + b_2\sin[2\pi\lambda_d(\tfrac{\tan\beta R_s}{\sin\alpha R_d}u_s + \varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)]
\end{aligned}
\tag{A-45}
$$

here

$$b_0 = -(2\pi)^2 R_s$$

$$b_1 = \left(\frac{2\pi \tan\beta R_s}{\sin\alpha}\right)^2 \frac{\cos\alpha}{R_d} + (2\pi)^2 R_d \cos\alpha + \frac{8\pi^2 \lambda_s \lambda_d \tan\beta R_s}{\sin\alpha}$$

$$b_2 = \left(\frac{2\pi \tan\beta R_s}{\sin\alpha}\right)^2 \frac{1}{R_d} + (2\pi)^2 R_d + \frac{8\pi^2 \lambda_s \lambda_d \tan\beta \cos\alpha R_s}{\sin\alpha}$$

so we can write $\frac{d^3 x_d}{du_s^3}$, $\frac{d^3 y_d}{du_s^3}$ as following using equation (A-44) and (A-45):

$$\begin{aligned}\frac{d^3 x_d}{du_s^3} &= c_0 \sin[2\pi\lambda_s(u_s+\varphi_s)] \\ &\quad + c_1 \cos[2\pi\lambda_d(\tfrac{\tan\beta R_s}{\sin\alpha R_d} u_s + \varphi_d)] \sin[2\pi\lambda_s(u_s+\varphi_s)] \\ &\quad + c_2 \sin[2\pi\lambda_d(\tfrac{\tan\beta R_s}{\sin\alpha R_d} u_s + \varphi_d)] \cos[2\pi\lambda_s(u_s+\varphi_s)]\end{aligned} \tag{A-46}$$

here

$$c_0 = (2\pi)^3 R_s \lambda_s$$

$$\begin{aligned}c_1 &= 2\pi\lambda_d \tfrac{\tan\beta R_s}{\sin\alpha R_d} \left\{ -\left(\tfrac{2\pi\tan\beta R_s}{\sin\alpha}\right)^2 \tfrac{\cos\alpha}{R_d} - (2\pi)^2 R_d \cos\alpha - \tfrac{8\pi^2\lambda_s\lambda_d\tan\beta R_s}{\sin\alpha} \right\} \\ &\quad -2\pi\lambda_s \left\{ \left(\tfrac{2\pi\tan\beta R_s}{\sin\alpha}\right)^2 \tfrac{1}{R_d} + (2\pi)^2 R_d + \tfrac{8\pi^2\lambda_s\lambda_d\tan\beta\cos\alpha R_s}{\sin\alpha} \right\} \\ c_2 &= 2\pi\lambda_s \left\{ -\left(\tfrac{2\pi\tan\beta R_s}{\sin\alpha}\right)^2 \tfrac{\cos\alpha}{R_d} - (2\pi)^2 R_d \cos\alpha - \tfrac{8\pi^2\lambda_s\lambda_d\tan\beta R_s}{\sin\alpha} \right\} \\ &\quad -2\pi\lambda_d \tfrac{\tan\beta R_s}{\sin\alpha R_d} \left\{ \left(\tfrac{2\pi\tan\beta R_s}{\sin\alpha}\right)^2 \tfrac{1}{R_d} + (2\pi)^2 R_d + \tfrac{8\pi^2\lambda_s\lambda_d\tan\beta\cos\alpha R_s}{\sin\alpha} \right\}\end{aligned}$$

$$\begin{aligned}\frac{d^3 y_d}{du_s^3} &= d_0 \cos[2\pi\lambda_s(u_s+\varphi_s)] \\ &\quad + d_1 \sin[2\pi\lambda_d(\tfrac{\tan\beta R_s}{\sin\alpha R_d} u_s + \varphi_d)] \sin[2\pi\lambda_s(u_s+\varphi_s)] \\ &\quad + d_2 \cos[2\pi\lambda_d(\tfrac{\tan\beta R_s}{\sin\alpha R_d} u_s + \varphi_d)] \cos[2\pi\lambda_s(u_s+\varphi_s)]\end{aligned} \tag{A-47}$$

here

$$d_0 = -(2\pi)^3 R_s \lambda_s$$

$$\begin{aligned}d_1 &= -2\pi\lambda_d \tfrac{\tan\beta R_s}{\sin\alpha R_d} \left\{ \left(\tfrac{2\pi\tan\beta R_s}{\sin\alpha}\right)^2 \tfrac{\cos\alpha}{R_d} + (2\pi)^2 R_d \cos\alpha + \tfrac{8\pi^2\lambda_s\lambda_d\tan\beta R_s}{\sin\alpha} \right\} \\ &\quad -2\pi\lambda_s \left\{ \left(\tfrac{2\pi\tan\beta R_s}{\sin\alpha}\right)^2 \tfrac{1}{R_d} + (2\pi)^2 R_d + \tfrac{8\pi^2\lambda_s\lambda_d\tan\beta\cos\alpha R_s}{\sin\alpha} \right\} \\ d_2 &= 2\pi\lambda_s \left\{ \left(\tfrac{2\pi\tan\beta R_s}{\sin\alpha}\right)^2 \tfrac{\cos\alpha}{R_d} + (2\pi)^2 R_d \cos\alpha + \tfrac{8\pi^2\lambda_s\lambda_d\tan\beta R_s}{\sin\alpha} \right\} \\ &\quad +2\pi\lambda_d \tfrac{\tan\beta R_s}{\sin\alpha R_d} \left\{ \left(\tfrac{2\pi\tan\beta R_s}{\sin\alpha}\right)^2 \tfrac{1}{R_d} + (2\pi)^2 R_d + \tfrac{8\pi^2\lambda_s\lambda_d\tan\beta\cos\alpha R_s}{\sin\alpha} \right\}\end{aligned}$$

$\frac{d^3 z_d}{du_s^3}$ can be

$$\frac{d^3 z_d}{du_s^3} = -\frac{(2\pi\tan\beta R_s)^3 \lambda_s \lambda_d}{(\sin\alpha R_d)^2} \cos\left[2\pi\lambda_d \left(\frac{\tan\beta R_s}{\sin\alpha R_d} u_s + \varphi_d\right)\right]$$

Now we begin to calculate $\frac{d^4x_d}{du_s^4}$, $\frac{d^4y_d}{du_s^4}$ use equation (A-46) and (A-47)

$$\begin{aligned}\frac{d^4x_d}{du_s^4} &= e_0\sin[2\pi\lambda_s(u_s+\varphi_s)] \\ &+e_1\cos[2\pi\lambda_d(\tfrac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)] \\ &+e_2\sin[2\pi\lambda_d(\tfrac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)]\end{aligned} \tag{A-48}$$

here

$$e_0 = (2\pi)^4 R_s$$

$$e_1 = 2\pi\lambda_d\frac{\tan\beta R_s}{\sin\alpha R_d}c_2 + 2\pi\lambda_s c_1$$

$$e_2 = -2\pi\lambda_d\frac{\tan\beta R_s}{\sin\alpha R_d}c_1 - 2\pi\lambda_s c_2$$

$$\begin{aligned}\frac{d^4y_d}{du_s^4} &= f_0\sin[2\pi\lambda_s(u_s+\varphi_s)] \\ &+f_1\cos[2\pi\lambda_d(\tfrac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\sin[2\pi\lambda_s(u_s+\varphi_s)] \\ &+f_2\sin[2\pi\lambda_d(\tfrac{\tan\beta R_s}{\sin\alpha R_d}u_s+\varphi_d)]\cos[2\pi\lambda_s(u_s+\varphi_s)]\end{aligned} \tag{A-49}$$

here

$$f_0 = -(2\pi)^4 R_s$$

$$f_1 = 2\pi\lambda_d\frac{\tan\beta R_s}{\sin\alpha R_d}d_1 - 2\pi\lambda_s d_2$$

$$f_2 = -2\pi\lambda_d\frac{\tan\beta R_s}{\sin\alpha R_d}d_2 + 2\pi\lambda_s d_1$$

$$\frac{df(u_s)}{du_s} = \left(\frac{dy_d}{du_s}\frac{d^3z_d}{du_s^3} - \frac{d^3y_d}{du_s^3}\frac{dz_d}{du_s}\right)\vec{i} + \left(\frac{dz_d}{du_s}\frac{d^3x_d}{du_s^3} - \frac{d^3z_d}{du_s^3}\frac{dx_d}{du_s}\right)\vec{j} + \left(\frac{dx_d}{du_s}\frac{d^3y_d}{du_s^3} - \frac{d^3x_d}{du_s^3}\frac{dy_d}{du_s}\right)\vec{k} \tag{A-50}$$

$$\begin{aligned}\frac{d^2f(u_s)}{du_s^2} &= \left(\frac{d^2y_d}{du_s^2}\frac{d^3z_d}{du_s^3} + \frac{dy_d}{du_s}\frac{d^4z_d}{du_s^4} - \frac{d^4y_d}{du_s^4}\frac{dz_d}{du_s} - \frac{d^3y_d}{du_s^3}\frac{d^2z_d}{du_s^2}\right)\vec{i} \\ &+\left(\frac{d^2z_d}{du_s^2}\frac{d^3x_d}{du_s^3} + \frac{dz_d}{du_s}\frac{d^4x_d}{du_s^4} - \frac{d^4z_d}{du_s^4}\frac{dx_d}{du_s} - \frac{d^3z_d}{du_s^3}\frac{d^2x_d}{du_s^2}\right)\vec{j} \\ &+\left(\frac{d^2x_d}{du_s^2}\frac{d^3y_d}{du_s^3} + \frac{dx_d}{du_s}\frac{d^4y_d}{du_s^4} - \frac{d^4x_d}{du_s^4}\frac{dy_d}{du_s} - \frac{d^3x_d}{du_s^3}\frac{d^2y_d}{du_s^2}\right)\vec{k}\end{aligned} \tag{A-51}$$

$$\begin{aligned}\frac{dg(u_s)}{du_s} &= \left[\left(\frac{dy_d}{du_s}\frac{d^2z_d}{du_s^2} - \frac{d^2y_d}{du_s^2}\frac{dz_d}{du_s}\right)^2 + \left(\frac{dz_d}{du_s}\frac{d^2x_d}{du_s^2} - \frac{d^2z_d}{du_s^2}\frac{dx_d}{du_s}\right)^2 + \left(\frac{dx_d}{du_s}\frac{d^2y_d}{du_s^2} - \frac{d^2x_d}{du_s^2}\frac{dy_d}{du_s}\right)^2\right]^{-\frac{1}{2}} \\ &\times\left[\left(\frac{dy_d}{du_s}\frac{d^3z_d}{du_s^3} - \frac{d^3y_d}{du_s^3}\frac{dz_d}{du_s}\right) + \left(\frac{dz_d}{du_s}\frac{d^3x_d}{du_s^3} - \frac{d^3z_d}{du_s^3}\frac{dx_d}{du_s}\right) + \left(\frac{dx_d}{du_s}\frac{d^3y_d}{du_s^3} - \frac{d^3x_d}{du_s^3}\frac{dy_d}{du_s}\right)\right]\end{aligned} \tag{A-52}$$

$$
\begin{aligned}
\frac{d^2g(u_s)}{du_s^2} = & -\left[\left(\frac{dy_d}{du_s}\frac{d^2z_d}{du_s^2} - \frac{d^2y_d}{du_s^2}\frac{dz_d}{du_s}\right)^2 + \left(\frac{dz_d}{du_s}\frac{d^2x_d}{du_s^2} - \frac{d^2z_d}{du_s^2}\frac{dx_d}{du_s}\right)^2 + \left(\frac{dx_d}{du_s}\frac{d^2y_d}{du_s^2} - \frac{d^2x_d}{du_s^2}\frac{dy_d}{du_s}\right)^2\right]^{-\frac{3}{2}} \\
& \times \left[\left(\frac{dy_d}{du_s}\frac{d^3z_d}{du_s^3} - \frac{d^3y_d}{du_s^3}\frac{dz_d}{du_s}\right) + \left(\frac{dz_d}{du_s}\frac{d^3x_d}{du_s^3} - \frac{d^3z_d}{du_s^3}\frac{dx_d}{du_s}\right) + \left(\frac{dx_d}{du_s}\frac{d^3y_d}{du_s^3} - \frac{d^3x_d}{du_s^3}\frac{dy_d}{du_s}\right)\right]^2 \\
& + \left[\left(\frac{dy_d}{du_s}\frac{d^2z_d}{du_s^2} - \frac{d^2y_d}{du_s^2}\frac{dz_d}{du_s}\right)^2 + \left(\frac{dz_d}{du_s}\frac{d^2x_d}{du_s^2} - \frac{d^2z_d}{du_s^2}\frac{dx_d}{du_s}\right)^2 + \left(\frac{dx_d}{du_s}\frac{d^2y_d}{du_s^2} - \frac{d^2x_d}{du_s^2}\frac{dy_d}{du_s}\right)^2\right]^{-\frac{1}{2}} \\
& \times \left[\begin{array}{l}
\left(\frac{d^2y_d}{du_s^2}\frac{d^3z_d}{du_s^3} + \frac{dy_d}{du_s}\frac{d^4z_d}{du_s^4} - \frac{d^4y_d}{du_s^4}\frac{dz_d}{du_s} - \frac{d^3y_d}{du_s^3}\frac{d^2z_d}{du_s^2}\right) \\
+\left(\frac{d^2z_d}{du_s^2}\frac{d^3x_d}{du_s^3} + \frac{dz_d}{du_s}\frac{d^4x_d}{du_s^4} - \frac{d^4z_d}{du_s^4}\frac{dx_d}{du_s} - \frac{d^3z_d}{du_s^3}\frac{d^2x_d}{du_s^2}\right) \\
+\left(\frac{d^2x_d}{du_s^2}\frac{d^3y_d}{du_s^3} + \frac{dx_d}{du_s}\frac{d^4y_d}{du_s^4} - \frac{d^4x_d}{du_s^4}\frac{dy_d}{du_s} - \frac{d^3x_d}{du_s^3}\frac{d^2y_d}{du_s^2}\right)
\end{array}\right]
\end{aligned}
\tag{A-53}
$$

similarly we also can know

$$
\frac{dN_d(u_s)}{du_s} = \frac{m'(u_s)n(u_s) - m(u_s)n'(u_s)}{n^2(u_s)} \tag{A-54}
$$

$$
\frac{d^2N_d(u_s)}{du_s^2} = \frac{\left[m''(u_s)n(u_s) - m(u_s)n''(u_s)\right]n^2(u_s) - 2\left[m'(u_s)n(u_s) - m(u_s)n'(u_s)\right]n(u_s)n'(u_s)}{n^4(u_s)} \tag{A-55}
$$

Now we need to calculate $m'(u_s), m''(u_s), n'(u_s)$ and $n''(u_s)$.

For Equation.(A-38) we can assume it can be expressed as following:

$$
f(u_s) = f_x\vec{i} + f_y\vec{j} + f_z\vec{k} \tag{A-56}
$$

f_x,f_y,f_z can be as following:

$$
f_x = \frac{dy_d}{du_s}\frac{d^2z_d}{du_s^2} - \frac{d^2y_d}{du_s^2}\frac{dz_d}{du_s}
$$

$$
f_y = \frac{dz_d}{du_s}\frac{d^2x_d}{du_s^2} - \frac{d^2z_d}{du_s^2}\frac{dx_d}{du_s}
$$

$$
f_z = \frac{dx_d}{du_s}\frac{d^2y_d}{du_s^2} - \frac{d^2x_d}{du_s^2}\frac{dy_d}{du_s}
$$

so $m(u_s)$ can be written as following:

$$
m(u_s) = (f_y z'_d - y'_d f_z)\vec{i} + (f_z x'_d - z'_d f_x)\vec{j} + (f_x y'_d - x'_d f_y)\vec{k} \tag{A-57}
$$

For Equation.(A-44) we can assume it can be expressed as following:

$$\frac{df(u_s)}{du_s} = f'_x\vec{i} + f'_y\vec{j} + f'_z\vec{k} \tag{A-58}$$

f'_x, f'_y, f'_zcan be expressed as following:

$$f'_x = \frac{dy_d}{du_s}\frac{d^3z_d}{du_s^3} - \frac{d^3y_d}{du_s^3}\frac{dz_d}{du_s}$$

$$f'_y = \frac{dz_d}{du_s}\frac{d^3x_d}{du_s^3} - \frac{d^3z_d}{du_s^3}\frac{dx_d}{du_s}$$

$$f'_z = \frac{dx_d}{du_s}\frac{d^3y_d}{du_s^3} - \frac{d^3x_d}{du_s^3}\frac{dy_d}{du_s}$$

$$\begin{aligned} m'(u_s) &= \left(f'_y z'_d + f_y z''_d - y''_d f_z - y'_d f'_z\right)\vec{i} \\ &\quad + \left(f'_z x'_d + f_z x''_d - z''_d f_x - z'_d f'_x\right)\vec{j} \\ &\quad + \left(f'_x y'_d + f_x y''_d - x''_d f_y - x'_d f'_y\right)\vec{k} \end{aligned} \tag{A-59}$$

For Equation.(A-44) we can assume it can be expressed as following:

$$\frac{d^2f(u_s)}{du_s^2} = f''_x\vec{i} + f''_y\vec{j} + f''_z\vec{k} \tag{A-60}$$

f''_x, f''_y, f''_z can be expressed as following:

$$f''_x = \frac{d^2y_d}{du_s^2}\frac{d^3z_d}{du_s^3} + \frac{dy_d}{du_s}\frac{d^4z_d}{du_s^4} - \frac{d^4y_d}{du_s^4}\frac{dz_d}{du_s} - \frac{d^3y_d}{du_s^3}\frac{d^2z_d}{du_s^2}$$

$$f''_y = \frac{d^2z_d}{du_s^2}\frac{d^3x_d}{du_s^3} + \frac{dz_d}{du_s}\frac{d^4x_d}{du_s^4} - \frac{d^4z_d}{du_s^4}\frac{dx_d}{du_s} - \frac{d^3z_d}{du_s^3}\frac{d^2x_d}{du_s^2}$$

$$f''_z = \frac{d^2x_d}{du_s^2}\frac{d^3y_d}{du_s^3} + \frac{dx_d}{du_s}\frac{d^4y_d}{du_s^4} - \frac{d^4x_d}{du_s^4}\frac{dy_d}{du_s} - \frac{d^3x_d}{du_s^3}\frac{d^2y_d}{du_s^2}$$

$$\begin{aligned} m''(u_s) &= \left(f''_y z'_d + 2f'_y z''_d + f'_y z'''_d - y'''_d f_z - 2y''_d f'_z - y'_d f''_z\right)\vec{i} \\ &\quad + \left(f''_z x'_d + 2f'_z x''_d + f'_z x'''_d - z'''_d f_x - 2z''_d f'_x - z'_d f''_x\right)\vec{j} \\ &\quad + \left(f''_x y'_d + 2f'_x y''_d + f'_x y'''_d - x'''_d f_y - 2x''_d f'_y - x'_d f''_y\right)\vec{k} \end{aligned} \tag{A-61}$$

Here $n(u_s)$ using equation (A-41) and (A-50) can be written as following:

$$n(u_s) = \sqrt{(x'_d)^2 + (y'_d)^2 + (z'_d)^2}\sqrt{(f_x)^2 + (f_y)^2 + (f_z)^2} \tag{A-62}$$

$$
\begin{aligned}
n'(u_s) &= (x'_d x''_d + y'_d y''_d + z'_d z''_d)\sqrt{(f_x)^2+(f_y)^2+(f_z)^2}/\sqrt{(x'_d)^2+(y'_d)^2+(z'_d)^2} \\
&\quad + \left(f_x f'_x + f_y f'_y + f_z f'_z\right)\sqrt{(x'_d)^2+(y'_d)^2+(z'_d)^2}/\sqrt{(f_x)^2+(f_y)^2+(f_z)^2}
\end{aligned} \tag{A-63}
$$

$$
\begin{aligned}
n''(u_s) &= -n'(u_s)^2/n(u_s) \\
&\quad + \left(f'_x f'_x + f_x f''_x + f'_y f'_y + f_y f''_y + f'_z f'_z + f_z f''_z\right)\left((x'_d)^2+(y'_d)^2+(z'_d)^2\right)/n(u_s) \\
&\quad +4\left(f_x f'_x + f_y f'_y + f_z f'_z\right)\left(x'_d x''_d + y'_d y''_d + z'_d z''_d\right)/n(u_s) \\
&\quad + \left((f_x)^2+(f_y)^2+(f_z)^2\right)\left(x''_d x''_d + x'_d x'''_d + y''_d y'''_d + y'_d y'''_d + z''_d z''_d + z'_d z'''_d\right)/n(u_s)
\end{aligned} \tag{A-64}
$$

Now we can use equation (A-42),(A-43),(A-48) and (A-49) to calculate $\frac{dN_d}{du_s}$, $\frac{dB_d}{du_s}$, $\frac{d^2N_d}{du_s^2}$ and $\frac{d^2B_d}{du_s^2}$. Through equation (A-30 $\sim$ 35) we get $\frac{dx_t}{du_s}$, $\frac{dy_t}{du_s}$, $\frac{dz_t}{du_s}$, $\frac{d^2x_t}{du_s^2}$, $\frac{d^2y_t}{du_s^2}$, $\frac{d^2z_t}{du_s^2}$. Here we can calculate $\vec{T}_t(u_s), \vec{N}_t(u_s) and \vec{B}_t(u_s)$.

$$
\vec{T}_t(u_s) = \frac{\frac{dx_t}{du_s}\vec{i} + \frac{dy_t}{du_s}\vec{j} + \frac{dz_t}{du_s}\vec{k}}{\sqrt{\left(\frac{dx_t}{du_s}\right)^2 + \left(\frac{dy_t}{du_s}\right)^2 + \left(\frac{dz_t}{du_s}\right)^2}} \tag{A-65}
$$

$$
\vec{B}_t(u_s) = \frac{\left(\frac{dy_t}{du_s}\frac{d^2z_t}{du_s^2} - \frac{d^2y_t}{du_s^2}\frac{dz_t}{du_s}\right)\vec{i} + \left(\frac{dz_t}{du_s}\frac{d^2x_t}{du_s^2} - \frac{d^2z_t}{du_s^2}\frac{dx_t}{du_s}\right)\vec{j} + \left(\frac{dx_t}{du_s}\frac{d^2y_t}{du_s^2} - \frac{d^2x_t}{du_s^2}\frac{dy_t}{du_s}\right)\vec{k}}{\sqrt{\left(\frac{dy_t}{du_s}\frac{d^2z_t}{du_s^2} - \frac{d^2y_t}{du_s^2}\frac{dz_t}{du_s}\right)^2 + \left(\frac{dz_t}{du_s}\frac{d^2x_t}{du_s^2} - \frac{d^2z_t}{du_s^2}\frac{dx_t}{du_s}\right)^2 + \left(\frac{dx_t}{du_s}\frac{d^2y_t}{du_s^2} - \frac{d^2x_t}{du_s^2}\frac{dy_t}{du_s}\right)^2}} \tag{A-66}
$$

$$
\begin{aligned}
\vec{N}_t(u_s) &= \vec{B}_d \otimes \vec{T}_d \\
&= \left(\frac{\frac{dx_t}{du_s}\frac{d^2x_t}{du_s^2} - \frac{d^2x_t}{du_s^2}\frac{dx_t}{du_s}}{\left\|\dot{X}_t \otimes \ddot{X}_t\right\|}\frac{\frac{dx_t}{du_s}}{\sqrt{\left(\frac{dx_t}{du_s}\right)^2+\left(\frac{dy_t}{du_s}\right)^2+\left(\frac{dz_t}{du_s}\right)^2}} - \frac{\frac{dy_t}{du_s}}{\sqrt{\left(\frac{dx_t}{du_s}\right)^2+\left(\frac{dy_t}{du_s}\right)^2+\left(\frac{dz_t}{du_s}\right)^2}}\frac{\frac{dx_t}{du_s}\frac{d^2y_t}{du_s^2} - \frac{d^2x_t}{du_s^2}\frac{dy_t}{du_s}}{\left\|\dot{X}_t \otimes \ddot{X}_t\right\|}\right)\vec{i} \\
&+ \left(\frac{\frac{dx_t}{du_s}\frac{d^2y_t}{du_s^2} - \frac{d^2x_t}{du_s^2}\frac{dy_t}{du_s}}{\left\|\dot{X}_t \otimes \ddot{X}_t\right\|}\frac{\frac{dx_t}{du_s}}{\sqrt{\left(\frac{dx_t}{du_s}\right)^2+\left(\frac{dy_t}{du_s}\right)^2+\left(\frac{dz_t}{du_s}\right)^2}} - \frac{\frac{dx_t}{du_s}}{\sqrt{\left(\frac{dx_t}{du_s}\right)^2+\left(\frac{dy_t}{du_s}\right)^2+\left(\frac{dz_t}{du_s}\right)^2}}\frac{\frac{dy_t}{du_s}\frac{d^2z_t}{du_s^2} - \frac{d^2y_t}{du_s^2}\frac{dz_t}{du_s}}{\left\|\dot{X}_t \otimes \ddot{X}_t\right\|}\right)\vec{j} \\
&+ \left(\frac{\frac{dy_t}{du_s}\frac{d^2z_t}{du_s^2} - \frac{d^2y_t}{du_s^2}\frac{dz_t}{du_s}}{\left\|\dot{X}_t \otimes \ddot{X}_t\right\|}\frac{\frac{dy_t}{du_s}}{\sqrt{\left(\frac{dx_t}{du_s}\right)^2+\left(\frac{dy_t}{du_s}\right)^2+\left(\frac{dz_t}{du_s}\right)^2}} - \frac{\frac{dx_t}{du_s}}{\sqrt{\left(\frac{dx_t}{du_s}\right)^2+\left(\frac{dy_t}{du_s}\right)^2+\left(\frac{dz_t}{du_s}\right)^2}}\frac{\frac{dz_t}{du_s}\frac{d^2x_t}{du_s^2} - \frac{d^2z_t}{du_s^2}\frac{dx_t}{du_s}}{\left\|\dot{X}_t \otimes \ddot{X}_t\right\|}\right)\vec{k}
\end{aligned} \tag{A-67}
$$

$$
S_t(u_s, v) = \vec{X}_t(u_s) + r[\cos(2\pi v)\,\vec{N}_t(u_s) + \sin(2\pi v)\,\vec{B}_t(u_s)] \tag{A-68}
$$

Bibliography

[1] M. Abramowitz and I.A. Stegun. *Handbook of Mathematical Functions with Formula, Graphs, and Mathematical Tables*. Dover, 1965.

[2] David H. Eberly. *3D game engine design : a practical approach to real-time computer graphics*. 2001.

[3] Andrew S. Glassner. *Introduction to ray tracing*. 1989.

[4] Loren Heiny. *Advanced graphics programming using C/C++*. 1993.

[5] J.D.Foley and A.Van Dam. *Fundamentals of interactive computer graphics*. 1983.

[6] Jon Q.Jacobs. *Delphi developer's guide to OpenGL*. 1998.

[7] Ian Stephenson. *Production rendering : design and implementation*. 2005.

[8] Ibrahim Zeid. *CAD/CAM theory and practice*. Dover, 1991.