Implementation of NURBS Objects in a Ray Tracing Code for RCS Simulation

Zahir Al-Asad

22th Jun, 2010

Master’s Thesis in Electronics/Telecommunication
Abstract

The properties of Non-Uniform Rational B-Splines (NURBS), the IGES file format and its data structure and implementation methods are presented and discussed in this paper. The goal is to show an approach to find intersections of rays on NURBS object. This method is implemented in an already existing ray tracing code for the simulation of electromagnetic fields propagation and scattering. The ray tracing code has been implemented and verified successfully before in the department.

The discussion of the report starts in the introduction chapter with the review of previous researches done for finding intersection of ray on objects. It also contains the problem of those researches and the goal of this project to be achieved. The theoretical discussion of NURBS and the IGES file format are described in two chapters. A brief description of ray tracing method is also described in one chapter. The attention is to be focused on describing the method of reading NURBS object from IGES file and methods for finding the ray intersection points on the NURBS. The report also discusses the problems of the methods found after implementation and possible solutions. The simulation and measurement results are also shown in this report. The report concludes with gain of this research and possible future implementations.

This report shows clearly the benefits of using NURBS for finding ray intersections on arbitrary objects, which will be supportive for those engineers who work on Radar Communications, especially on object detection and ray tracing and also the CAD/CAM designer. This method can be implemented for finding ray intersection points on objects with a very reasonable CPU time.
Acknowledgments

First of all I would like to thank to my supervisor Dr.-Ing. Frank Weinmann for giving me the opportunity to do my Master’s thesis work in the department of Antenna Technology and Electromagnetic Modeling (AEM) in Fraunhofer Institute for High Frequency Physics and Radar Techniques (FHR) in Germany and for being friendly and his excellent guidance during the whole period of the thesis work. I am very grateful to my examiner Prof. Claes Beckman, who allowed me to work in this thesis work.

I also would like to thank to my mother Anwara Begum, father Prof. Dr. Md. Azhar Ali and my borther Dr. Hafiz Al-Asad for their love, financial and mental support. I would like to thank to my friends too, especially my colleagues in Fraunhofer whose are encouraging me during the hard time.
# Contents

1. Introduction .................................................................................................................. 2
2. NURBS .......................................................................................................................... 5
   2.1. B-spline Curves and Surfaces .............................................................................. 5
   2.2. B-spline Basis Functions ..................................................................................... 6
   2.3. Knot Vector .......................................................................................................... 6
   2.4. Non-Uniform Rational B-Spline (NURBS) Curves and Surfaces ....................... 7
3. IGES File Format .......................................................................................................... 10
   3.1. Classification of Entities .................................................................................... 13
   3.2. 128 Entity ........................................................................................................... 14
4. Ray Tracing .................................................................................................................. 21
5. Working procedure ...................................................................................................... 23
   5.1. NURBS Parameters .......................................................................................... 24
   5.2. Read IGES File ................................................................................................... 25
   5.3. Find Intersection Points ..................................................................................... 26
      5.3.1. Conditions for intersection point ................................................................. 26
      5.3.2. Numerical implementation ......................................................................... 28
      5.3.3. Optimized intersection algorithm ............................................................... 31
      5.3.4. Second optimization method ....................................................................... 32
   5.4. Normal to the intersection point on NURBS ..................................................... 33
6. Result and Verification ................................................................................................. 36
   6.1. Geometrical Validation of Reading IGES Files ................................................. 36
   6.2. Geometrical Validation of Intersection Points ................................................. 37
   6.3. Validation in the Ray Tracing Environment ...................................................... 38
7. Conclusion .................................................................................................................... 41
   7.1. Future Work ........................................................................................................ 41
References ......................................................................................................................... 42
Appendix I ......................................................................................................................... 45
Read_IGES() Function .................................................................................................. 45
Chapter I
1. Introduction

Radar which comes to forefront as a useful sensor in World War II is firstly used by the German Navy in 1903 for ship detection [16]. Over the last two decades radar reflectivity measurement has developed a lot [17]. Radar has been used primarily as a sensor to detect and track targets by using Radar Cross Section (RCS), which describe reflection characteristics of electromagnetic wave incident on objects [18]. However, to determine the RCS for simple shaped objects such as a sphere, cone, paraboloid, and corner reflector by solving Maxwell’s equations, but for a complex object it’s not an easy task [19].

Ray tracing is a popular and powerful technique for modeling complex flying objects [20]. A standard method of the ray tracing is Shooting and Bouncing Ray method (SRB) [21]. But for complex object SRB method is considerable challenging due to geometric and physical complexity. Basically, the overhead of computation in the ray tracing algorithm is related to the determination of the intersection of a ray and an object (a wall surface, a ground plane or a wall edge etc.) in the propagation environments [22]. This geometric testing can consume more than 90% of CPU time for a naive SBR algorithm [23]. Several techniques have been proposed to reduce the time spent of geometric testing using reprocessing of the propagation environment. Most of the modern ray tracing based applications only deals with triangles as basic primitives because of their fast intersection test with a ray [24].

The SRB algorithm is successfully applied for calculating scattered fields of arbitrary targets. This simulation program can handle some elementary objects (e.g. spheres, cylinders) analytically, but for complex objects, a triangular mesh is required. Also objects from common CAD file formats can be used, but these have to be meshed into triangles by using a commercial CAD program and saving the mesh in NASTRAN format. Although by rendering objects in triangles it is possible to use arbitrary geometric objects in the ray tracing process, but it has disadvantages. For large and complex objects large number of elements for intersection tests required, and also large memory required. To improve this, an alternative geometric representation of objects needs to be studied.

A very powerful way of representing arbitrary curved surface is the use of Non Uniform Rational B-Spline (NURBS). NURBS are the standard in computer graphics and most CAD programs for the geometrical representation and design of curved objects in computer graphics. NURBS have common mathematical formulas for both standard analytical shapes and free-form shapes. It is possible to design 3D geometry and simple 2D curves and surfaces using NURBS and provides a much higher geometrical precision. NURBS are recognized as powerful tools for geometric design in many standards like IGES, STEP, and PHIGS. In this project work IGES standard has been used to read NURBS objects. IGES form is a common language for different graphical software for interchange geometrical data.

The goal of the project work is to develop an algorithm to implement NURBS in the existing ray tracing code. To achieve that, a NURBS data structure has been created and an algorithm to read the NURBS objects parameters from IGES files has been developed. Two methods for finding ray intersection on the NURBS surface have been developed and implemented in the ray tracing code for RCS simulation. The project code has been done using C++. The immediate benefit of implementing NURBS is that IGES files can be read directly without the need of meshing the object into triangles.
Also it is expected to reduce the number of geometrical objects and thus to reduce memory usage as well as CPU time.

This report is organized as follows; chapter II provides a short overview of the fundamental mathematical formulas and properties of NURBS, chapter III explains the IGES file format, chapter IV describes about the ray tracing method shortly, chapter V provides the whole working procedure in detail, the result and verification are available chapter VI and the report ended with conclusion and future work in chapter VII.

The research work has been done as a Master’s thesis of University of Gävle in the Department of Antenna Technology and Electromagnetic Modeling of Fraunhofer Institute for High Frequency Physics and Radar Techniques FHR in Wachtberg, Germany.
Chapter II
2. NURBS

NURBS, i.e. Non-Uniform Rational B-splines [1], are the mathematical representation of 3-D geometry for describing and modeling curves and surfaces in computer aided design and computer graphics. They can describe any shape from a simple 2-D line, circle, or curve to the most complex 3-D organic free-form surface or solid.

NURBS are a special case of B-splines, whose definitions are outlined in the following subsection.

2.1. B-spline Curves and Surfaces

A B-spline is a spline, i.e., a piecewise polynomial function. The B-spline curve \( H(u) \), is defined by,

\[
H(u) = \sum_{i=0}^{n} P_i N_{i,k}(u)
\]  

(2.1)

Where,

\( \{P_i: i=0, 1, 2, \ldots, n\} \); are the \( n+1 \) control points. The polygon formed by these control points is the control polygon.

\( N_{i,k} \) are the normalized B-spline basis functions

\( k +1 \) is the order of the polynomial segments of the B-spline curve.

A B-spline surface \( Q(u, v) \) is defined by,

\[
Q(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,k}(u) M_{j,l}(v) P_{i,j}
\]  

(2.2)

Where,

\( N_{i,k}(u) \) and \( M_{j,l}(v) \) are the \( k \)th and \( l \)th-degree normalized basis functions of two knot vectors with \( k+1 \) and \( l+1 \) order, respectively.
2.2. B-spline Basis Functions

The $i$th B-spline basis function of $k$-degree (order $k+1$), is defined by,

If $k=0$,

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } (u_i \leq u < u_{i+1}) \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (2.3)

If $k>0$

$$N_{i,k}(u) = \frac{u - u_i}{u_{i+k} - u_i} N_{i,k-1}(u) + \frac{u_{i+k+1} - u}{u_{i+k+1} - u_{i+1}} N_{i+1,k}(u)$$  \hspace{1cm} (2.4)

Where,

$N_{i,d}(u)$ is a step function, equal to zero everywhere except on the half-open interval $u \in [u_i, u_{i+1})$;

$U = \{u_0, u_1, \ldots, u_m\}$ is the knot vector with $i=0, 1, \ldots, m$ number of knot values (see following paragraph for further description of knot vectors).

2.3. Knot Vector

A knot vector is an increasing series of real numbers that satisfy the relation ($u_i \leq u_{i+1}$). There are two types of knot vectors, uniform and non-uniform, also known as periodic and open [1].

A uniform knot vector has evenly spaced real numbers. For example,

[0 1 2 3 4 5 6]

Or

[−0.4 −0.2 0.0 0.2 0.4] etc.

A non-uniform knot vector has a series of real number with multiplicity of knot values at the ends equal to the order $k$ of the knot vector. Internal knot values are evenly spaced. For example,
\[ k=3 \quad [0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 3 \quad 3 \quad 3] \]

Or

\[ k=4 \quad [0 \quad 0 \quad 0 \quad 0.25 \quad 0.5 \quad 0.75 \quad 1 \quad 1 \quad 1 \quad 1] \]

2.4. Non-Uniform Rational B-Spline (NURBS) Curves and Surfaces

A \( k \)-th degree NURBS curve is defined by [2],

\[
C(u) = \frac{\sum_{i=0}^{n} P_i N_{i,k}(u) w_i}{\sum_{i=0}^{n} N_{i,k}(u) w_i} \tag{2.5}
\]

Where,

\( P_i \) are the control points.

\( N_{i,k} \) is \( k \)-th degree normalized B-spline basis function of \( k+1 \) order.

\( w_i \) are the weights. These weights are associated with the control points. Depending on the weight of a control point the curve moves closer to the control point. In other words, the weight of a control point controls how much it can pull the curve [7]. There is an example in Figure 2.1, which shows how a curve varies with different weights of control points.

![Figure 2.1: Variation of curve for different weights of control points.](image)
A NURBS surface $S(u,v)$ is the extended form of B-spline surface by introducing weighted factor which causes a four-dimensional homogeneous coordinate space is defined by,

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,k}(u)M_{j,l}(v)P_{i,j}^h w_{i,j}$$  \tag{2.6}$$

Where,

$P_{i,j}^h$ are the control points with weighting factors are represented as homogeneous coordinates in four dimensional space.

$N_{i,k}(u)$ and $M_{j,l}(v)$ are the nonrational basis functions of two knot vectors with $k+1$ and $l+1$ order respectively.

$w_{i,j}$ are the weights.

NURBS can be represented by the division of three-dimension space by the homogenous coordinate,

$$S(u,v) = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,k}(u)M_{j,l}(v)P_{i,j}^h w_{i,j}}{\sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,k}(u)M_{j,l}(v)w_{i,j}} = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{i,j} B_{i,j}(u,v)$$  \tag{2.7}$$

Where,

$P_{i,j}$ are the three-dimensional control points.

$B_{i,j}(u,v)$ are the bivariate rational B-spline surface basis functions

$$B_{i,j}(u,v) = \frac{N_{i,k}(u)M_{j,l}(v)w_{i,j}}{\sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,k}(u)M_{j,l}(v)w_{i,j}} = \frac{N_{i,k}(u)M_{j,l}(v)w_{i,j}}{sum(u,v)}$$  \tag{2.8}$$

Where,

$sum(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,k}(u)M_{j,l}(v)w_{i,j}$  \tag{2.9}$$
Chapter III
3. IGES File Format

The Initial Graphics Exchange Specification (IGES), an American National Standard (ANS) format, serves as a neutral data format to transfer a CAD design to a dissimilar system. Translators, developed to the IGES Standard, are used to export a design into an IGES file for exchange and for importing the IGES file into the destination system [8].

There are over 200 different variations of IGES files that are currently being written by different Computer-Aided Design (CAD) and 3D software packages. In general only about a quarter of the specification is used for translation of 3D polygonal and NURBS geometry [9].

An IGES file is the representation of geometric, topological, and nongeometric product information data. Each of those product definition is numbered by a positive integer, which is known as IGES entity. For example, IGES entity 102 represents Composite Curve.

Most CAD/Computer-Aided Manufacturing (CAM) systems support NURBS curves and surfaces [10][11]. IGES gives all information used to define the free-form parametric spline surfaces. Trimmed surfaces in CAD/CAM systems can be represented by the IGES entities 102, 126, 128, 142 and 144. The 126 and 128 entities support NURBS curves and surfaces respectively.

CAD systems usually maintain transcript files in a binary format. A binary file format is almost always unique to a software program. Therefore it needs to export the transcript to another software using a common language. The IGES form is that kind of format to allow the exchange of a product definition between CAD/CAM systems. It has two format types,

a) A fixed line length format
b) A compressed format

The most commonly used IGES format is the fixed line length format. In fixed line length format each line has 80 characters. These 80 characters are divided according to the table below [10],

<table>
<thead>
<tr>
<th>1-----8</th>
<th>9-----16</th>
<th>17-----24</th>
<th>25-----32</th>
<th>33-----40</th>
<th>41-----48</th>
<th>49-----56</th>
<th>57-----64</th>
<th>65-----72</th>
<th>73-----80</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Entity Type Number #</td>
<td>(2) Parameter Data )</td>
<td>(3) Structure #; )</td>
<td>(4) Line Font Pattern #; )</td>
<td>(5) Level #; )</td>
<td>(6) View 0; )</td>
<td>(7) Transformation Matrix 0; )</td>
<td>(8) Label Display Assoc. 0; )</td>
<td>(9) Status Number #</td>
<td>(10) Sequence Number D #</td>
</tr>
<tr>
<td>(11) Entity Type Number #</td>
<td>(12) Line Weight Number #</td>
<td>(13) Color Number #; )</td>
<td>(14) Parameter Line Count #</td>
<td>(15) Form Number #</td>
<td>(16) Reserved</td>
<td>(17) Reserved</td>
<td>(18) Entity Label</td>
<td>(19) Entity Subscript Number #</td>
<td>(20) Sequence Number D # + 1</td>
</tr>
</tbody>
</table>

Table 3.1. Format of the Directory Entry (DE) Section in the IGES file form
Nomenclatures used in table 3.1 are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)</td>
<td>Field number n</td>
</tr>
<tr>
<td>#</td>
<td>Integer</td>
</tr>
<tr>
<td>)</td>
<td>Pointer</td>
</tr>
<tr>
<td>#; )</td>
<td>Integer or pointer (pointer has negative sign)</td>
</tr>
<tr>
<td>0; )</td>
<td>Zero or pointer</td>
</tr>
</tbody>
</table>

Table 3.2. Nomenclatures used in the Directory Entry (DE) section

Following the format of directory entity of Table 3.1 it is sectioned in the IGES form as bellow,

<table>
<thead>
<tr>
<th>No.</th>
<th>Field Name</th>
<th>Meaning and Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Entity Type Number</td>
<td>Identifies the Entity types</td>
</tr>
<tr>
<td>2</td>
<td>Parameter Data</td>
<td>Pointer to the first line of the parameter data record for the entity. The letter P is not included.</td>
</tr>
<tr>
<td>3</td>
<td>Structure</td>
<td>Negated pointer to the directory entry of the definition entity that specifies this entity’s meaning. The letter D is not included. The integer values, 0, 1 and 2 are permissible in the field but should be disregarded.</td>
</tr>
<tr>
<td>4</td>
<td>Line Font Pattern</td>
<td>Line font pattern or negated pointer to the directory entity of a line Font Definition Entity (Type 304)</td>
</tr>
<tr>
<td>5</td>
<td>Level</td>
<td>Number of the level upon which the entity resides, or a negated pointer to the directory entry of a Definition Levels Property Entity (Type 406, Form 1) which contains a list of levels upon which the entity resides.</td>
</tr>
<tr>
<td>6</td>
<td>View</td>
<td>Pointer to the directory entry of a View Entity (Type 410), or pointer to a Views Visible Associativity Instance (Type 402, Form 3 or 4), or integer zero (default).</td>
</tr>
<tr>
<td>7</td>
<td>Transformation Matrix</td>
<td>Pointer to the directory entry of a Transformation Matrix Entity (Type 124) used in defining this entity; zero (default) implies the identity transformation matrix and zero translation vector will be used.</td>
</tr>
<tr>
<td>8</td>
<td>Label Display Associatively</td>
<td>Pointer to the directory entry of a label Display Associativity (Type 402, Form 5). The value of zero indicates no label display associativity.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Details</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
</tbody>
</table>
| 9     | Status Number | Provides four two-digit status values which are entered from left to right in the status number field in the order given below,  
1-2 Blank Status  
00 Visible  
01 Blanked  
3-4 Subordinate Entity Switch  
00 Independent  
01 Physically Dependent  
02 Logically Dependent  
03 Both (01) and (02)  
5-6 Entity Use Flag  
00 Geometry  
01 Annotation  
02 Definition  
03 Other  
04 Logical/Positional  
05 2D Parametric  
7-8 Hierarchy  
00 Global top down  
01 Global defer  
02 Use hierarchy property |
| 10    | Section Code and Sequence Number | Physical count of this line from the beginning of the Directory Entry Section, preceded by the letter D (odd number). |
| 11    | Entity Type Number | (Same as Field 1.) |
| 12    | Line Weight Number | System display thickness; given as a gradation value in the range of 0 to the maximum (Parameter 16 of the Global Section). |
| 13    | Color Number | Color number or negated pointer to the directory entry of a Color Definition Entity (Type 314). |
| 14    | Parameter Line Count Number | Number of lines in the parameter data record for this entity. |
| 15    | Form Number | Certain entities have different interpretations. These interpretations are uniquely identified by a form number. Possible form numbers are listed within each entity description. |
| 16    | Reserved for future use | |
| 17    | Reserved for future use | |
| 18    | Entity Label | Up to eight alphanumeric characters (right justified). |
| 19    | Entity Subscript Number | 1 to 8 digit unsigned number associated with the label. |
| 20    | Section Code and Sequence Number | Same meaning as Field 10 (even number). |

Table 3.3. Directory Entry (DE) Section.
The sequence number is preceded in the line by a single letter code in column 73 given in the table below,

<table>
<thead>
<tr>
<th>Section</th>
<th>Letter Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag (not always</td>
<td>B or C (Use to</td>
</tr>
<tr>
<td>present)</td>
<td>signify binary)</td>
</tr>
<tr>
<td>Start</td>
<td>S</td>
</tr>
<tr>
<td>Global</td>
<td>G</td>
</tr>
<tr>
<td>Directory Entry</td>
<td>D</td>
</tr>
<tr>
<td>Parameter Data</td>
<td>P</td>
</tr>
<tr>
<td>Terminate</td>
<td>T</td>
</tr>
</tbody>
</table>

Table 3.4. Letter Codes Specification.

3.1. Classification of Entities

Generally, the IGES entities are classified into four classes,

- a) Curve and surface geometry entities
- b) Constructive solid entities
- c) Annotation entities
- d) Structure entities

For geometric representation of CAD data curve and surface geometry entities are needed. Entity type numbers from 100 through 199 are generally reserved for geometry entities. The following curve and surface geometry entities are defined in this specification:

<table>
<thead>
<tr>
<th>Entity number</th>
<th>Entity type</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Circular Arc</td>
</tr>
<tr>
<td>102</td>
<td>Composite Curve</td>
</tr>
<tr>
<td>104</td>
<td>Conic Arc</td>
</tr>
<tr>
<td>106</td>
<td>Copious Data</td>
</tr>
<tr>
<td></td>
<td>Linear Path</td>
</tr>
<tr>
<td></td>
<td>Simple Closed Planar Curve</td>
</tr>
<tr>
<td>108</td>
<td>Plane</td>
</tr>
<tr>
<td>110</td>
<td>Line</td>
</tr>
</tbody>
</table>
The project work is to implement the NURBS objects into the ray tracing code. There are several IGES entities that represent 3D curved surfaces. Each entity has different way of defining them. For example, entity 114 represents parametric Spline surface, entity 140 represents offset surface, etc. But entity 128 is the general form on NURBS. It is possible to almost all type of NURBS objects using entity 128.

IGES entity type 128 represents a Rational B-Spline Surface and its directory entry is defined as follows,

<table>
<thead>
<tr>
<th>1——8</th>
<th>9——16</th>
<th>17——24</th>
<th>25——32</th>
<th>33——40</th>
<th>41——48</th>
<th>49——56</th>
<th>57——64</th>
<th>65——72</th>
<th>73——80</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>(2)</td>
<td>(3)</td>
<td>(4)</td>
<td>(5)</td>
<td>(6)</td>
<td>(7)</td>
<td>(8)</td>
<td>(9)</td>
<td>(10)</td>
</tr>
<tr>
<td>Entity Type Number 128</td>
<td>Parameter Data</td>
<td>Structure</td>
<td>Line Font Pattern</td>
<td>Level</td>
<td>View</td>
<td>Transformation Matrix</td>
<td>Label Display Assoc.</td>
<td>Status Number</td>
<td>Sequence Number</td>
</tr>
<tr>
<td>#;</td>
<td>#;</td>
<td>#;</td>
<td>#;</td>
<td>#;</td>
<td>0;</td>
<td>0;</td>
<td>0;</td>
<td>#</td>
<td>D #</td>
</tr>
<tr>
<td>Entity Type Number 128</td>
<td>Line Weight Number</td>
<td>Color Number</td>
<td>Parameter Line Count</td>
<td>Form Number</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Entity Label</td>
<td>Entity Subscript Number</td>
<td>Sequence Number</td>
</tr>
<tr>
<td>#;</td>
<td>#;</td>
<td>#;</td>
<td>#;</td>
<td>#;</td>
<td></td>
<td></td>
<td></td>
<td>#</td>
<td>D # + 1</td>
</tr>
</tbody>
</table>

Table 3.6. Format of the directory entry for entity 128
Parameter data (P-Section) are sequenced as in table 3.7.

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>K1</td>
<td>Integer</td>
<td>Upper index of first sum (Equation 2.6)</td>
</tr>
<tr>
<td>2</td>
<td>K2</td>
<td>Integer</td>
<td>Upper index of second sum (Equation 2.6)</td>
</tr>
<tr>
<td>3</td>
<td>M1</td>
<td>Integer</td>
<td>Degree of first set of basis function</td>
</tr>
<tr>
<td>4</td>
<td>M2</td>
<td>Integer</td>
<td>Degree of second set of basis function</td>
</tr>
</tbody>
</table>
| 5   | PROP1 | Integer | 1 = Closed in first parametric variable direction  
|     |      |       | 0 = Not close |
| 6   | PROP2 | Integer | 1 = Closed in second parametric variable direction  
|     |      |       | 0 = Not Closed |
| 7   | PROP3 | Integer | 1 = Polynomial  
|     |      |       | 0 = Rational |
| 8   | PROP4 | Integer | 1 = Non-periodic in first parametric variable direction  
|     |      |       | 0 = Periodic in first parametric variable direction |
| 9   | PROP5 | Integer | 1 = Non-periodic in second parametric variable direction  
|     |      |       | 0 = Periodic in second parametric variable direction |

Let,  
\[ N1 = 1 + K1 - M1, \]  
\[ N2 = 1 + K2 - M2, \]  
\[ A = N1 + 2 * M1, \]  
\[ B = N2 + 2 * M2, \]  
\[ C = (1 + K1) * (1 + K2) \]

<table>
<thead>
<tr>
<th>10</th>
<th>S(-M1)</th>
<th>Real</th>
<th>First values of first knot sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10+A</td>
<td>S(N1+M1)</td>
<td>Real</td>
<td>Last value of first knot sequence</td>
</tr>
<tr>
<td>11+A</td>
<td>T(-M2)</td>
<td>Real</td>
<td>First value of second knot sequence</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11+A+B</td>
<td>T(N2+M2)</td>
<td>Real</td>
<td>Last value of second knot sequence</td>
</tr>
<tr>
<td>12+A+B</td>
<td>W(0,0)</td>
<td>Real</td>
<td>First weights</td>
</tr>
</tbody>
</table>

15
<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>13+A+B</td>
<td>(W(1,0))</td>
<td>Real</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\ldots)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11+A+B+C</td>
<td>(W(K1,K2))</td>
<td>Real</td>
<td>Last weights</td>
</tr>
<tr>
<td>12+A+B+C</td>
<td>(X(0,0))</td>
<td>Real</td>
<td>First control point</td>
</tr>
<tr>
<td>13+A+B+C</td>
<td>(Y(0,0))</td>
<td>Real</td>
<td></td>
</tr>
<tr>
<td>14+A+B+C</td>
<td>(Z(0,0))</td>
<td>Real</td>
<td></td>
</tr>
<tr>
<td>15+A+B+C</td>
<td>(X(1,0))</td>
<td>Real</td>
<td></td>
</tr>
<tr>
<td>16+A+B+C</td>
<td>(Y(1,0))</td>
<td>Real</td>
<td></td>
</tr>
<tr>
<td>17+A+B+C</td>
<td>(Z(1,0))</td>
<td>Real</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\ldots)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9+A+4*C</td>
<td>(X(K1,K2))</td>
<td>Real</td>
<td>Last control point</td>
</tr>
<tr>
<td>10+A+4*C</td>
<td>(Y(K1,K2))</td>
<td>Real</td>
<td></td>
</tr>
<tr>
<td>11+A+4*C</td>
<td>(Z(K1,K2))</td>
<td>Real</td>
<td></td>
</tr>
<tr>
<td>12+A+4*C</td>
<td>(U(0))</td>
<td>Real</td>
<td>Starting value of first parametric direction</td>
</tr>
<tr>
<td>13+A+4*C</td>
<td>(U(1))</td>
<td>Real</td>
<td>Ending values of first Parametric direction</td>
</tr>
<tr>
<td>14+A+4*C</td>
<td>(V(0))</td>
<td>Real</td>
<td>Starting value of second parametric direction</td>
</tr>
<tr>
<td>15+A+4*C</td>
<td>(V(1))</td>
<td>Real</td>
<td>Ending value of second parametric direction</td>
</tr>
</tbody>
</table>

| **Table 3.7. Parametric data sequence of entity 128.** |

An IGES file can contain any number of entities and can have different entities in the same file. An example of a simple IGES file given is below in figure 3.1. It contains different entities including entity 128. It is possible to read the entity form the IGES file following the table 2.6 and 3.7. Column 73 of each line contains the letter code as in table 3.4. The IGES file starts with the letter code “S” and terminates with “T”. All lines with letter code “D” contain the directory entities and lines with letter code “P” contain the parameters of entities.
Figure 3.1. Simple IGES file containing different IGES entities

| 1H, 1H: |
| 78BC: \Dokumente und Einstellungen\frank\AUS\Eigene Dateien\M\Frank\opheG |
| Rel2.1SG, |
| 26Rhinoceros ( Apr 24 2002 ), 31HTrout Lake IGES 012 Apr 24 2002, |
| 32, 33, 6, 305, 15, |
| 1.0DD, 2.2MM, 1.0, 0.254DC, 13H090923.153641, |
| 0.01DD, |
| 3, 9, |
| 10R, |
| 10Rhinoceros, |
| 1, 0, 13H090923.153641; |
| 314 1 0 0 0 0 0 0 000000200D 1 |
| 314 0 1 1 0 0 0 0 0 COLOR DD 2 |
| ... |
| 314 6 0 0 0 0 0 0 0 000000200D 11 |
| 314 0 8 1 0 0 0 0 0 COLOR DD 12 |
| 406 7 0 0 1 0 0 0 000000300D 13 |
| 406 0 -1 1 3 0 0 LEVELDEF DD 14 |
| ... |
| 406 12 0 0 0 0 0 0 0 000000300D 23 |
| 406 0 -11 1 3 0 0 0 LEVELDEF DD 24 |
| 128 13 0 0 0 0 0 0 0 000000000D 25 |
| 128 0 -1 42 0 0 0 0 0 TrimSrf DD 26 |

314, 0.0, 0.0, 0.0, 0.0, 20HRGB (0, 0, 0); 00000001P 1
314, 100.0, 0.0, 0.0, 0.0, 20HRGB (255, 0, 0); 00000002P 2
314, 49, C1961, 14.90156, 80.39216, 20HRGB (125, 38, 205); 00000003P 3
314, 0.0, 3.0, 100.0, 0.0, 20HRGB (0, 0, 255); 00000004P 4
314, 0.0, 49, 50392, 0.0, 20HRGB (0, 127, 0); 00000005P 5
314, 100, 0.0, 100.0, 0.0, 20HRGB (255, 255, 255); 00000006P 6
406, 2.1, 7HDefault; 00000013P 7
406, 2.2, 8Layer 01; 00000015P 8
406, 2.3, 8Layer 02; 00000017P 9
406, 2.4, 8Layer 03; 00000019P 10
406, 2.5, 8Layer 04; 00000021P 11
406, 2.6, 8Layer 05; 00000023P 12
128, 8, 4, 2, 1, 0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.50.30733729463599D, 00000025P 13
1.530733729463599D, 3.06146745920718DD, 0.06146724526718D, 0.06146724526718D, 14
4.59202118381077DD, 4.59202118381077DD, 4.59202118381077DD, 6.122934917844136DD, 00000015P 15
6.122934917844136DD, 6.122934917844136DD, 6.122934917844136DD, 0.0DD, 0.0DD, 0.0DD, 0.0DD, 0.0DD, 15
1.530733729463599D, 1.50733729463599D, 3.06146745920718DD, 0.06146724526718D, 0.06146724526718D, 16
3.06146745920718DD, 3.06146745920718DD, 3.06146745920718DD, 3.06146745920718DD, 17
0.7071067811865475DD, 1.0DD, 0.7071067811865475DD, 1.0DD, 00000025P 18
0.7071067811865475DD, 1.0DD, 0.7071067811865475DD, 1.0DD, 00000025P 19
0.7071067811865475DD, 1.0DD, 0.7071067811865475DD, 1.0DD, 00000025P 20
0.7071067811865475DD, 0.500000000000001DD, 0.7071067811865475DD, 1.0DD, 00000025P 21
0.7071067811865475DD, 0.500000000000001DD, 0.7071067811865475DD, 1.0DD, 00000025P 22
0.7071067811865475DD, 0.500000000000001DD, 0.7071067811865475DD, 1.0DD, 00000025P 23
0.7071067811865475DD, 0.500000000000001DD, 0.7071067811865475DD, 1.0DD, 00000025P 24
0.7071067811865475DD, 0.500000000000001DD, 0.7071067811865475DD, 1.0DD, 00000025P 25
... |

1.1102232024682515BD-016.0, 0.9999999999999999D, -1.0DD, 0.00000025P 49
0.9999999999999999D, 0.9999999999999999D, -1.0DD, 0.00000025P 50
-1.0DD, 0.0DD, -1.0DD, 0.0DD, 0.0DD, -1.0DD, 0.0DD, 0.0DD, -1.0DD, 0.00000025P 51
0.0DD, 0.0DD, -1.0DD, 0.0DD, -1.0DD, 0.0DD, -1.0DD, 0.0DD, -1.0DD, 0.00000025P 52
-1.0DD, 0.0DD, -1.0DD, 0.0DD, -1.0DD, 0.0DD, -1.0DD, 0.00000025P 53
6.122934917844136DD, 0.0DD, 3.06146745920718DD, 0.00000025P 54
00000001G000000026P00000054 |

17
The IGES file in Figure 3.1 is divided into two parts shown in Figure 3.2 and 3.3. The first half of the IGES file contains the directory entities. The directory entity is in the D section (lines with letter code “D”) in the IGES file. For example, the directory entity of the 128 entity is bordered in Figure 3.2. The directory entity is built up according to Table 3.6. For reading the parametric data of 128 entity, only two entries are taken from the directory entity, the starting line and the number of lines in the P section (lines with letter code “P”). In Figure 3.2 the second field of the directory entity is 13, it means the 13th line in the P section is the starting line of the parametric data. And the 14th field is 42, it means there are 42 lines from 13th line in the P section containing parametric data of this 128 entity.

![Directory of 128 entity](image)

**Figure 3.2. Directory of 128 entity is bordered.**

The parametric data of all entities are sequenced in Figure 3.3. The red border highlights the parametric data of the 128 entity. The two knot vectors, weights, control points and the ranges of two parametric directions are sequenced inside this border respectively. In this project work those data are used to calculate points on the NURBS surface.
Figure 3.3. Parametric data sequence of 128 is bordered.

Although the IGES file is a complicated format, this is a common language for the CAD/CAM, that makes easy to share and use object definition into different software. Following the directory entities and arrangement of parametric data as described in the above sections objects can be identified and used in different software packages.
Chapter IV
4. Ray Tracing

For the electromagnetic modeling of scattered fields from large and complex objects, a very efficient simulation tool based on the well-known Shooting-and-Bouncing-Rays (SBR) technique has been developed [13]. This algorithm uses field-based high-frequency approaches for finding the relevant propagation paths, i.e. rays are traced through the scenario according to the laws of Geometrical Optics (GO) and diffracted rays are generated according to the laws of the Uniform Theory of Diffraction (UTD). This GO/UTD part is combined with a source-based approach, where each ray generates an equivalent surface current at each interaction with the object. These currents are calculated using Physical Optics (PO) and Physical Theory of Diffraction (PTD), and the superposition of all these contributions leads to the total scattered field strengths. Further technical details about the ray tracing approach can be found in [13, 14, 15]. With this simulation tool, large and complex objects can be modeled, which are still too costly for numerically exact methods. Generally, both metallic and dielectric objects [14], and also combined metallic-dielectric objects, can be modeled.

Consider a ray hitting a smooth surface which is part of the object. The ray is described according to GO and is reflected at the surface. However, there is also a direct field contribution to the receiver, which is calculated by applying PO. After calculating the PO contribution, the ray is reflected according to the laws of GO. The next interaction in the example of Figure 4.1 is at an edge of the object. At an edge the scattered fields are calculated by PTD and diffracted ray according to the law of UTD incident on a smooth surface reflected according to GO and another field contribution from there is calculated by PO. Finally total scattered field strengths are calculated by combining the calculated fields of PO and PTD.

Figure 4.3. Example of implemented calculation methods.
Chapter V
5. Working procedure

The goal of the project work is to read NURBS objects from IGES files, find the ray intersection points on those objects and at last calculate the normal vector on those intersection points which are needed for calculating reflected ray. To achieve this goal the work has been divided into two parts,

a) Create NURBS data structures and read the data from IGES files.

b) Find the intersection points during the ray tracing processes.

The whole working procedure is shown in a flow chart in Figure 5.1.
5.1. NURBS Parameters

The first part of this project work is to read all 128 entities (NURBS objects) from the IGES file. Chapter II provides more details about NURBS. The NURBS parameters are number of segments, knot vectors, order of knot vectors, control points, weight of the control points and the parametric directions. Considering those parameters a NURBS class has been created in the project code, which is shown below,

class NURBS
{
public:
    VECTOR ** control_net;
    int n, m;
    int k, l;
    double * X;
    double * Y;
    double **N;
    double **M;
    double **H;
    double u0,u1,v0,v1;
    double delu,delv;

    friend void basis_f(NURBS *, double, double);
    VECTOR normal;
    NURBS * p_next;
    NURBS();
};

Where,

control_net is a VECTOR type two-dimensional control net. VECTOR is a self-defined class which is created for simplifying vector calculation.

n and m are the number of segments in u and v direction.

k and l are the spline orders in u and v direction.

X and Y are the knot vectors.

H is the weights of control points.

u0, u1, v0 and v1 are the parametric directions.

basis_f() is a friend function which returns calculated results from spline basis function for each knot vector.

M and N are the returned values from basis f() for knot vector X and Y respectively.
normal is a VECTOR type variable for temporarily storing the local normal vector at the intersection point.

p_next is pointer to the next NURBS in the object list.

5.2. Read IGES File

In order to define NURBS data structures the 128 entity is chosen from the IGES file. Considering the IGES fixed line format as described in section 3 and following the Tables 3.6 and 3.7 for directory and parameter sequence of 128 entity respectively, the file reading process follows the algorithm below,

Algorithm to read 128 entity from IGES file:

Step 1: Check each line if the 72th character is “D”.

For each 128 entity, the directory entry consists of 2 lines, each line having 10 fields. Each field is represented by 8 characters.

Step 2: Look for “128”, in the first 8 characters of each line which contain “D” in the 72th position.

Step 3: Save the 2nd field of the first line, which represents the line number of the starting of parameter data sequence.

Step 4: Read the second line and save the 4th field, which represents the number of lines for the parameter data.

Step 5: Continue the steps 2, 3, 4 and store the data in a temporary array of parameter sequence list of all NURBS. Stop this procedure at the end of the directory section.

The parameter data is placed in those lines where the 72th character of a line is “P”. Each two parameters are separated by “,” and the parameter data sequence is ended with “;”.

Step 6: Start checking each line for “P” character in 72th position.

Step 7: If “P” is found in 72th position start counting the number of lines until the line number matches with the starting line number in the NURBS class which has been stored in step 3. (The 1st match is for the 1st NURBS, 2nd match is for 2nd NURBS and so on).

Step 8: If steps 6 and 7 satisfy then start reading the parameters sequentially as in table 3.7 and save the data in the NURBS class until the line number reaches the limit that is stored in NURBS class in step 4.

Step 9: Continue steps 7 and 8 until the parameters for all NURBS are read from the file.
The algorithm has been implemented in the function Read_IGES() in the project work, which is given in Appendix I.

5.3. Find Intersection Points

After reading all the NURBS the next step is to find the intersection of rays on the NURBS surface. This procedure has to be performed for each ray on each NURBS object, and if necessary even multiple times, e.g. after reflection of the ray. Thus, this part is very critical with respect to the run time of a simulation. The main problem here is the fact that there is no analytical solution to finding ray intersections on a NURBS surface. That is the reason why either the surface representation has to be converted to plane facets or an alternative algorithm, e.g. an iterative algorithm, has to be used. In this work, a straightforward approach has been developed, which is illustrated in the following subsection 5.2.1 and 5.2.2. This algorithm is iterative in the sense that it uses two stages of refinement in order to improve the accuracy of the computed intersection points. However, it turned out that this iteration is a major drawback because it results in a large amount of calculations and therefore increased CPU time. For this reason, an optimization of the approach is described in subsection 5.2.3, which uses an interpolation algorithm instead of an iterative approach. It turned out that by using this improved method of finding intersection points, computation times were drastically reduced as well as accuracy was definitely improved. Further possible optimizations of the intersection algorithm are briefly described at the end of this chapter.

5.3.1. Conditions for intersection point

To find an intersection point of a ray with a NURBS object, the initial approach is to follow the method developed by Kajiya [12]. Following this method, a grid of points \( S(u,v) \) on the surface are calculated using equation 2.2.

Consider a complex surface \( S(u,v) \) and a ray \( R(t) \), whose intersection point with the surface is to be calculated. The ray can be defined by its parametric equation,

\[
R(t) = o + td
\]

(5.1)

Where,

- \( o \) is the ray origin and \( d \) is the ray direction. \( t \) is the distance parameter of ray.

The methods to find the intersection point of the ray on the NURBS, following the development of Kajiya, are as follows.

The ray can be represented as the intersection of two orthogonal planes, for example, \( P_1 = (N_1,d_1) \) and \( P_2 = (N_2,d_2) \) (see Figure 5.1).
Figure 5.1: Representation of a ray by the intersection of two orthogonal planes.

The normal of the first plane is defined as,

\[
N_1 = \begin{cases} 
    \frac{(\hat{d}_x, -\hat{d}_y, 0)}{|(\hat{d}_x, -\hat{d}_y, 0)|} & \text{if } |\hat{d}_x| > |\hat{d}_y| \text{ and } |\hat{d}_x| > |\hat{d}_z| \\
    (0, \hat{d}_x, -\hat{d}_y) & \text{otherwise}
\end{cases}
\]  

(5.2)

Thus, \( N_1 \) is always perpendicular to the ray direction \( \hat{d} \). The normal of the second plane, \( N_2 \), is defined as,

\[ N_2 = N_1 \times \hat{d} \]  

(5.3)

The distances \( d_1 \) and \( d_2 \) of the planes from the global origin \( O=(0,0,0) \) with respect to the normal vectors \( N_1 \) and \( N_2 \) respectively are defined as,

\[
d_1 = -N_1 \cdot O \\
d_2 = -N_2 \cdot O
\]  

(5.4) 
(5.5)
Here, the ray origin \( \mathbf{o} \) is the vector from the origin of the global coordinate system \( \mathbf{O} = (0,0,0) \) to the starting point of the ray, it lies on the ray and on both planes.

The addition of distance with the scalar product of normal and point on the plane must be zero if there is an intersection point on the surface. The conditions are as bellow,

\[
N_1 \cdot S(u,v) + d_1 = 0 \\
N_2 \cdot S(u,v) + d_2 = 0
\]  \hspace{1cm} (5.6) \hspace{1cm} (5.7)

However, due to the complex definition of a NURBS surface, equations (5.6) and (5.7) can generally not be solved analytically. That is why an iterative algorithm has been developed in order to find a point on the surface which satisfies equations (5.6) and (5.7) as close as possible.

### 5.3.2. Numerical implementation

As stated above, there is no general analytical solution to equation (5.6) and (5.7), which must be satisfied for an intersection of a ray with a surface \( S(u,v) \). Thus, a straightforward algorithm is developed here, using a grid of points on the surface \( S(u,v) \) and finding the point which matches (5.5) most closely.

In IGES files, the ranges of \( u \) and \( v \) parameters are given, i.e. start and ending of \( u \) and \( v \) parameters \((u_0, u_i, v_0, v_j)\), which provides a clear definition of the surface (Figure 5.2). Theoretically, all the points on the surface would have to be checked with equations (5.6) and (5.7) to find the correct intersection point. To solve this problem, \( u \) and \( v \) parameters are first divided into very small intervals \( u_0, u_1, ..., u_i, v_0, v_1, ..., v_j \). This grid of parameters corresponds to a grid of points on the surface (calculation of points according to equation 2.7), and thus \( S(u,v) \) is represented by a meshed surface as in Figure 5.3.

![Figure 5.2: Four points calculated using the end and starting points of \( u \) and \( v \).](image1)

![Figure 5.3: Each corner of the rectangle represents points on the surface.](image2)
After calculating those points, the distances $F_1$, $F_2$ of each grid point $S(u,v_j)$ to the two planes are calculated using the equation below,

$$ N_1S(u,v_j) + d_1 = F_1(u,v_j) \quad (5.8) $$

$$ N_2S(u,v_j) + d_2 = F_2(u,v_j) \quad (5.9) $$

If the ray hits the patch, it will pass “in between” the four points $S(u,v_j)$, $S(u,v_{j+1})$, $S(u_{i+1},v_j)$, and $S(u_{i+1},v_{j+1})$, which is equivalent to the following condition,

$$ \begin{align*}
F_1(u,v_j) &> 0, F_1(u,v_{j+1}) > 0, F_1(u_{i+1},v_j) > 0, F_1(u_{i+1},v_{j+1}) > 0 \\
\text{or} & \\
F_1(u,v_j) &< 0, F_1(u,v_{j+1}) < 0, F_1(u_{i+1},v_j) < 0, F_1(u_{i+1},v_{j+1}) < 0 \\
\text{or} & \\
F_2(u,v_j) &> 0, F_2(u,v_{j+1}) > 0, F_2(u_{i+1},v_j) > 0, F_2(u_{i+1},v_{j+1}) > 0 \\
\text{or} & \\
F_2(u,v_j) &< 0, F_2(u,v_{j+1}) < 0, F_2(u_{i+1},v_j) < 0, F_2(u_{i+1},v_{j+1}) < 0
\end{align*} \quad (5.10) $$

As an example, in Figure 5.4 the ray crosses the surface through the black point and the resulting intersection will be the red point. Obviously, this is a very coarse approximation of intersection points, which is the reason for implementing an iteration of the described approach.

To get a better approximation of the real intersection point, the surface area between the points $S(u,v_j)$, $S(u,v_{j+1})$, $S(u_{i+1},v_j)$, and $S(u_{i+1},v_{j+1})$ is refined according to Figures 5.2 and 5.3 if these points fulfill condition 5.10 (see Figure 5.5). For this fine grid of points, the procedure described above is applied again, in order to find a better approximation of the intersection point. Theoretically, the refinement can be performed in many levels, which would lead to a very accurate calculation of intersection points. However, each iteration requires the calculation of a new grid of points, which takes more CPU resources. Thus, only two stages of refinement are used in the scope of this work. Typical numbers of grid points are 11x11, 21x21, and 51x51, resulting in meshes of 10x10, 20x20, and 50x50 elements.
However, it was found that, in some cases this algorithm cannot find the approximate intersection points. In some cases it can find intersections only for the first refinement and during the second refinement it cannot find any intersection. If only the first refinement is considered, there are some grid points closer to the intersection point are also treated as intersection points.

As it is shown in Figures 5.6 and 5.7, sometimes it is possible to find more than one polygons which satisfy the condition of intersection. In this case the algorithm takes the nearest polygon for second refinement, which doesn’t contain the intersection point.

Beside that it takes large amount of calculations and therefore increased CPU time.
5.3.3. Optimized intersection algorithm

To solve the problem of the approach described in section 5.2.1 and 5.2.2, an optimization method has been developed. For simplification consider the nearest four points of an intersection point on the surface, which for better readability are written here as, \( S_{11} \), \( S_{12} \), \( S_{21} \) and \( S_{22} \) instead of \( S(u_i,v_j) \), \( S(u_i,v_{j+1}) \), \( S(u_{i+1},v_j) \) and \( S(u_{i+1},v_{j+1}) \). The steps of the new approach is given below,

Step 1: Project \( S_{11} \), \( S_{12} \), \( S_{21} \) and \( S_{22} \) on ray as shown in figure 5.8, which results in \( P_1 \), \( P_2 \), \( P_3 \) and \( P_4 \). So there is an approximate intersection point,

\[
P = \frac{1}{4}(P_1 + P_2 + P_3 + P_4)
\] (5.11)

Step 2: Create a plane through \( P \) and perpendicular to the ray. Project \( S_{11} \), \( S_{12} \), \( S_{21} \) and \( S_{22} \) on that plane and the projected points are \( S_{11P} \), \( S_{12P} \), \( S_{21P} \) and \( S_{22P} \). Calculate the angles, \( \alpha_1 \), \( \alpha_2 \), \( \alpha_3 \) and \( \alpha_4 \) as shown in Figure 5.9. Now if \( \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 360^\circ \), the approximate intersection point on the ray \( P \) is inside the projected area. In that case, \( P \) is considered to be the approximate intersection point.

Step 3: Calculate the relative distances \( \text{dis}_1 \) and \( \text{dis}_2 \) (as shown in Figure 5.10). Both distances must between 0 and 1. Calculate interpolated \( u \) and \( v \) values using the following equations,
\[ u_{\text{int}} = u[i] + \text{dis}_1 (u[i+1] - u[i]) \]  
\[ v_{\text{int}} = v[i] + \text{dis}_2 (v[j+1] - v[j]) \]

The equation for \( \text{dis}_1 \) and \( \text{dis}_2 \) are below,

\[
\text{dis}_1 = (P - S_{11} - P) \cdot (S_{21} - P - S_{11}) / (\text{norm}(S_{21} - P - S_{11}) \cdot \text{norm}(S_{21} - P - S_{11}))
\]

\[
\text{dis}_2 = (P - S_{11} - P) \cdot (S_{12} - P - S_{11}) / (\text{norm}(S_{12} - P - S_{11}) \cdot \text{norm}(S_{12} - P - S_{11}))
\]

The resultant approximate intersection point is \( P_{\text{int}} = S(u_{\text{int}}, v_{\text{int}}) \). This is a good approximation for the intersection point exactly on the surface.

The first algorithm described in section 5.3.2 could yield grid points as approximate intersection points, but this second optimized method does an interpolation between the grid points. Thus the second refinement is not needed in this method, which reduces calculation time and thus, CPU time decreases significantly.

5.3.4. Second optimization method

The result of the optimized intersection algorithm described above in section 5.3.3 is sufficient to find approximate intersection points on the surface, but still CPU time is not low enough. The reason
for that is that for each ray, each NURBS is checked for intersection, even if it is far away from the ray trajectory.

To implement the efficiency of the intersection algorithm, a so-called space partition can be used. For example, 3D spatial “grids” are formed, and objects are linked with the grids they are part of. Also there is a link between the ray and the current spatial grid the ray is in. An example is illustrated in Figure 5.11. In this figure triangular objects are placed in different partitions. Thus, his method performs intersection tests only in the current spatial grid, and if there is no intersection the ray moves on to the next grid. The calculation of intersection points is done only when an object is found in the current grid. This approach saves a lot of unnecessary intersection testing.

The described method has already been implemented earlier for triangles and could be done in a similar way for NURBS or patches of NURBS.

Since the calculation is done only if an object is found in a space partition, calculation efforts and CPU time are significantly decreased.

5.4. Normal to the intersection point on NURBS

In this project work approximate normal to the intersection point has been calculated, which is needed to find the reflection of the incident ray. For that purpose, four neighboring points close to the intersection points are calculated by adding a very small value with u and v parameters in both directions.

\[ Q(u_i \pm d, v_j \pm d) \quad u_i < d < u_{i+1} \quad \text{and} \quad v_j < d < v_{j+1}; \]

These four points can be considered as four corners of a rectangle. Taking the cross product of the difference vectors between points the normal is calculated.
Figure 5.12: Suppose a ray intersects the surface in the black point.

Figure 5.13: The normal is the cross product of the difference vectors of neighboring four points.
Chapter VI
6. Result and Verification

The verification of the thesis work has been done in 3 steps.

6.1. Geometrical Validation of reading IGES files
6.2. Geometrical validation of intersection points
6.3. Validation in the ray tracing environment by calculating radar cross section (RCS) data of NURBS objects

6.1. Geometrical Validation of Reading IGES Files

Reading an IGES files is a very important part of this project work. It needs to be accurate in order to represent the object’s geometrical properties correctly. To verify that, several IGES files have been experimented.

Figure 6.1 and Figure 6.2 show the results of reading known IGES files, a sphere consisting of a single NURBS object and a more complex object (generic missile) consisting of 29 NURBS objects, respectively.

After reading the geometric data from the IGES file, each NURBS object is divided into 20×20 patches by splitting the u and v domains into 20 subintervals. Then each patch is divided into two triangles and those triangles’ data is written in an output file. Thus, a triangular mesh of the whole object is created. Using Matlab the triangles are read from the file and displayed graphically. For better visualization, a blue and a read triangle are used for each patch of four points \( S(u_i,v_j), S(u_i,v_{j+1}), S(u_{i+1},v_j), \) and \( S(u_{i+1},v_{j+1}) \)

Figure 6.1: Reading result of a sphere from IGES file.
For both IGES files, the triangular meshes created in Matlab show a good representation of the objects, so that the IGES reading algorithm is validated accurately.

### 6.2. Geometrical Validation of Intersection Points

To verify the calculated intersection points, Figure 6.3 shows the graphical representation of intersection points for the optimized algorithm. The figures show the ray intersection points at a sphere. Each cross represents an intersection point in these figures, since there is a grid of rays was shot at the sphere, lot of crosses shows in the figure.
6.3. Validation in the Ray Tracing Environment

The IGES file of a sphere described by a single NURBS object has been used for validation in the ray tracing environment. For frequencies with wavelengths much smaller than the sphere with radius \( r \), theoretical RCS equal to the geometrical cross section \( (\pi r^2) \). In \( \text{dBm}^2 \): \( 10 \times \log_{10}(\pi r^2)\).

The result of the project work in the ray tracing environment depends mainly on the accuracy of the calculated intersection points. Thus, it depends also on the number of refinements. The result of implementation of the method described in section 5.3.2 is better if the ranges of \( u \) and \( v \) parameters are divided into smaller intervals, because the resultant approximate intersection points become closer to the real intersection points. Figures 6.5, 6.6 and 6.7 show the different RCS results for different intervals in the \( u \) and \( v \) domain. Where, hh-pol and vv-pl represent horizontal and vertical polarization of the RCS signal respectively. In 10x10 intervals, the variation of RCS is from -5 to 20 \( \text{dBm}^2 \). The variation of the RCS decreases in 20x20 intervals shown in Figure 6.4 to -5 to 14 \( \text{dBm}^2 \). As expected, the best result is obtained in Figure 6.7 where 50x50 intervals have been used. RCS values are in the range from 4 to 7 \( \text{dBm}^2 \). This is rather close to the expected RCS of 5 \( \text{dBm}^2 \). The simulation results in Figures 6.5 to 6.7 show that a good refinement is required in order to get accurate RCS results.

However, the CPU time rises exponentially when improving the number of interval. For this reason, an optimized algorithm has been developed, which is described in Section 5.3.3. The RCS results of the improved algorithm are shown in Figures 6.8 to 6.11. Figure 6.10 and 6.11 both are for 50x50 intervals; however Figure 6.11 is for a large number of rays.

![Figure 6.5: Result of RCS simulation for 10x10 intervals.](image)

![Figure 6.6: Result of RCS simulation for 20x20 intervals.](image)
Figure 6.7: Result of RCS simulation for 50x50 intervals.

Figure 6.8: Result of RCS simulation for 10x10 intervals (improved algorithm).

Figure 6.9: Result of RCS simulation for 20x20 intervals (improved algorithm).

Figure 6.10: Result of RCS simulation for 50x50 intervals (improved algorithm).

Figure 6.11: Result of RCS simulation for 50x50 intervals and for large amount of rays (improved algorithm).
Chapter VII
7. Conclusion

The goal of the project work is to reduce the CPU time for geometric testing (i.e. to determine ray intersections) of objects in the ray tracing method for RCS simulation by the implementation of NURBS objects in the already developed ray tracing code. To achieve that, a code was written, which is able to read all 128 entities from IGES file, which represent NURBS objects. Methods to find ray intersection points were developed and coded. Normal vector to the intersection points are also calculated at the end of the coding.

There are two methods implemented for finding intersection points on the NURBS surface. The first method takes a lot of CPU time and under certain conditions it is not possible to find any intersection point. Using the second method it is possible to find all the intersection points and also decrease the CPU time.

After implementing the code implemented in a ray tracing environment it was observed that, if the number of refinement increases the RCS results become more accurate.

7.1. Future Work

In future work space partition can be implemented for finding intersection on NURBS surfaces. A possible implementation of a space partition algorithm is described briefly in section 5.3. Since this method tests only the grids which contain objects, it avoids unnecessary testing for the grids where no object is available. This way CPU time can decrease impressively.

It has been found that objects from IGES files with several NURBS objects (e.g. the generic missile) show problems in the simulation. Between NURBS objects it may possible to have gaps. So rays can travel inside the object through the gaps, which leads to errors in the calculations. To solve that an algorithm can be developed to close gaps between NURBS objects.
References


Appendix I

Read_IGES() Function

void Read_IGES(GLOBAL *p_global, char *filename)
{
    char temp[64];
    int i,j,t,tt,l,count,c,d,e,f,g,h,stop,txp,tpy;
    long int k;
    string s;
    double start,num,form_n,entity,u,w;
    int prop[6],nx,ny;
    NURBS *p_nu = new NURBS;
    stop=tpy=0;
    ifstream file_op0(filename,ios::in);
    if(!file_op0)
    {
        cout<<"Cant't open file"<<endl;
        exit(0);
    }
    //Counting the number of NURBS
    while(!file_op0.eof())
    {
        getline(file_op0,s);
        while(s[72]!="D")
        {
            if ((s[5]!="1") && (s[6]!="2") && (s[7]!="8"))
                tpy++;
            getline(file_op0,s);
            if(s[72]!="D")
                stop=1;
        }
        if (stop==1)
            break;
    }
    file_op0.close;
    tpy=tpy/2; //number of nurbs
    //Define array for storing all the NURBS directory
    long int **t_no;
    t_no= new long int *[tpy];
    for(i=0;i<tpy;i++)
        t_no[i] = new long int[2];
    ifstream file_op(filename,ios::in);
    if(!file_op)
    {
cout << "Can't open file" << endl;
exit(0);

k = num = c = d = tt = t = count = e = f = g = h = tpx = 0;
while (!file_op.eof())
{
    // t_no[tpx][tpy]=0;
    getline(file_op, s);
    // test = s.length();
    // cout << s;
    while (s[72] == 'D')
    {
        {
            l = 0;
            for (j = 0; j < 64; j++)
                temp[j] = '\0';
            for (i = 0; i < 80; i++)
            {
                if ((i > 7) && (i < 16))
                {
                    temp[l] = s[i];
                    // temporary store the starting line number
                    l = l + 1;
                    if (l > 7)
                    {
                        start = atof(temp);
                        t_no[tpx][0] = start;
                        l = 0;
                        for (j = 0; j < 64; j++)
                            temp[j] = '\0';
                    }
                }
            }
        }
    }
    getline(file_op, s);
    l = 0;
    for (i = 0; i < 72; i++)
        if ((i > 23) && (i < 32))
        {
            temp[l] = s[i];
            l = l + 1;
            if (l > 7)
            {
                num = atof(temp);
                t_no[tpx][1] = num;
                l = 0;
            }
        }
}
tpx++; }

tpx=0;
while(s[72]=='P')
{
    k=k++;
    if (((k>=int(t_no[tpx][0]))&&(k<(int(start+t_no[tpx][1])))))
    {
        for(i=0;i<64;i++)
        {
            if (s[i]!=' ')
            {
                for(j=0;j<64;j++)
                { temp[j] = '\0';
                    l=0
                    while(s[i]!=',')
                    {
                        temp[l]=s[i];
                        l++;
                        if ((s[i]=='','')||(s[i]==';'))
                        {
                            count++;  
                            if(s[i]==';')
                            break;
                        }
                    }
            }
            else break;
        }
        if (count==1)
        entity=atof(temp);
        if (count==2)
        {
            p_nu->n=atoi(temp);
            p_nu->H = new double *[p_nu->n+1];
            p_nu->control_net = new VECTOR *[p_nu->n+1];
        }
        if (count==3)
        {
            p_nu->m=atoi(temp);
            for(j=0;j<=p_nu->n;j++)
            {
                p_nu->H[j] = new double [p_nu->m+1];
                p_nu->control_net[j] = new VECTOR [p_nu->m+1];
            }
        }
}
}
if (count==4)
{
    p_nu->k=atoi(temp)+1;
    p_nu->X = new double [(p_nu->n+p_nu->k)*2+3];
    for (j=0; j<(p_nu->n+p_nu->k)*2+3; j++)
        p_nu->X[j] = 0;
}

if (count==5)
{
    p_nu->l=atoi(temp)+1;
    p_nu->Y = new double [(p_nu->m+p_nu->l)*2+3];
    for (j=0; j<(p_nu->m+p_nu->l)*2+3; j++)
        p_nu->Y[j] = 0;
}
if (count==6)
    prop[1]=atoi(temp);
if (count==7)
    prop[2]=atoi(temp);
if (count==8)
    prop[3]=atoi(temp);
if (count==9)
    prop[4]=atoi(temp);
if (count==10)
    prop[5]=atoi(temp);
if ((count>10)&&(count<(10+p_nu->n+p_nu->k+1))&&(i<65))
{
    p_nu->X[c]=atof(temp);
    c++;
}
if ((count>(10+p_nu->n+p_nu->k+1))&&(count<=(10+p_nu->n+p_nu->k+p_nu->m+p_nu->l+2))&&(i<65))
{
    p_nu->Y[d]=atof(temp);
    d++;
}

//////// Read Wei ghts from IGES file //////////

if ((count>(10+p_nu->n+p_nu->k+p_nu->m+p_nu->l+2))&&(count<=((10+p_nu->n+p_nu->k+p_nu->m+p_nu->l+2)+(p_nu->n+1)*(p_nu->m+1)))&&(i<65))
if (tt <= p_nu - m)
{
    p_nu->H[t][tt] = atof(temp);
    t++;
    if(t > p_nu->n)
    {
        t = 0;
        tt++;
    }
}

.Normalize control points /////////////////

if((count>((10+p_nu->n+p_nu->k+p_nu->m+p_nu->l+2)+((p_nu->n+1)*(p_nu->m+1))))&&
   (count<=((10+p_nu->n+p_nu->k+p_nu->m+p_nu->l+2)+4*(p_nu->n+1)*(p_nu->m+1)))&&
   (i<65))
{
    if(f <= p_nu->m)
    {
        if (g == 0)
        {
            p_nu->control_net[e][f].x = atof(temp);
            g++;
        }
        else if (g == 1)
        {
            p_nu->control_net[e][f].y = atof(temp);
            g++;
        }
        else if (g == 2)
        {
            p_nu->control_net[e][f].z = atof(temp);
            g = 0;
            e++;
        }
        if(e > p_nu->n)
        {
            e = 0;
            f++;
        }
    }
}

.Normalize U,V from IGES file ///////////////
if((count>((10+p_nu->n+p_nu->k+p_nu->m+p_nu->l+2)+4*(p_nu->n+1))* (p_nu->m+1)))&(count<=((14+p_nu->n+p_nu->k+p_nu->m+p_nu->l+2)+4*(p_nu->n+1)*(p_nu->m+1)))&(i<65))
{
if(h==0)
    p_nu->u0 = atof(temp); // this is an example
if(h==1)
    p_nu->u1 = atof(temp); // this is an example
if(h==2)
    p_nu->v0 = atof(temp); // this is an example
if(h==3)
    p_nu->v1 = atof(temp); // this is an example,
VECTOR *Q= new VECTOR;
nx = p_nu->n + p_nu->k;
p_nu->N = new double *[nx+1];
for(j=0; j<=nx; j++) // 9 is the length of the knot vector
    p_nu->N[j] = new double [nx+1];
ny= p_nu->m + p_nu->l;
p_nu->M = new double *[ny+1];
for(j=0; j<=ny; j++) // 9 is the length of the knot vector
    p_nu->M[j] = new double [ny+1];
CreatNewNurbs(p_global, p_nu);
}

h++;