Database Storage of 3D Freeform Curves and Surfaces

Shi Pu

Dec 24, 2004

Contents

Chapter 1

Introduction

After rapid adoption of the OpenGIS specification, current GIS is becoming an integration of strong database management and a powerful editing and visualization environment. A lot of research has been done since the increasing requirement for representing, accessing and disseminating spatial data, and more and more DBMSs (Oracle, Informix, Ingres, Postgres), and CAD/GIS applications (MicroStation, MapInfo) give support to spatial operations.

However, current spatial DBMSs support only 2D representations and operations. For example: **Informix** supports three basic spatial data types: point, line and polygon; **Ingres** supports one more type: circle, besides the three basic types; **Oracle Spatial** not only has points, lines, polygons and circles, but gives further support to arc strings and compound polygons. In these three DBMSs, all spatial data types and functions are 2-dimensional except that points may have a z-coordinate. No real 3D spatial primitive can be stored.

An effective solution to this problem has been developed at the Section GIS Technology, Technical University of Delft [1]: A 3D primitive (polyhedron) can be defined as a bounded subset of 3D space enclosed by a finite set of flat polygons, such that every edge of a polygon is shared by exactly one other polygon. The polygons are in 3D space because they are represented by vertices, which can be 3D points in Geo-DBMSs. Based on this idea, a true 3D primitive in the Geo-DBMS, including validation, conversion, indexing and spatial functions in 3D, has been implemented, and topology is partially maintained in 3D instead of 2D.

Since this was the first attempt of 3D spatial operations in Geo-DBMSs, more complex primitives can be researched to implement. Many shapes in real world are freeform, i.e. not only polyhedrons, but also curves and curved surfaces. Although freeform shapes can be simulated by tiny polygons, it is quite unrealistic and inconvenient to store every polygon into DBMS, especially when shapes are very huge or complex. A more efficient solution is to use a powerful mathematical model which is able to represent any desired shape, and store this mathematical representation directly into DBMS. So here comes the objective of this research assignment: **exploring database storage of 3D freeform curves and surfaces**.

In order to store freeform curves and surfaces in Geo-DBMS, a powerful and convenient mathematical representation is required to represent any shapes we desire, so the report starts with Chapter 2 which introduces the three most popular mathematical models for representing curves and curved surfaces, and makes a choice. A detailed description of the NURBS model and its fundamental algorithms will be given first in Chapter 3; then how NURBS can be used to construct freeform solids will also be explained in this chapter. Chapter 4 gives a snapshot of two CAD applications: AutoCAD and MicroStation, to show how freeform curves and surfaces are supported in practice. Current database support of spatial data is illustrated in Chapter 5, and Chapter 6 continues with the spatial data storage in a particular spatial DBMS: Oracle Spatial. The report is closed with conclusions and recommendations in Chapter 7.

Chapter 2

Mathematical Representations of 3D Freeform Curves and Surfaces

Various different representations of 3D freeform curves and surfaces are in use. Curves may be represented by using different kinds of equations. In particular, we distinguish the following representations [6]:

1. explicit or Cartesian, where the curve is given as the graph of a function;
2. implicit, as the zero set of one or more global algebraic equations;
3. parametric, associated with a vector function of one parameter;
4. intrinsic, where points locally satisfy differential equations.

Among the representations above, the parametric representation is most widely used by CAD applications. In this chapter we will make a comparison of the three most popular parametric representations for 3D freeform curves and surfaces, and choose one as our mathematical representation for database storage.

2.1 Bézier

The following describes the mathematics for the so-called Bézier curves (Figure 2.1)/surfaces (Figure 2.2). It is attributed and named after a French engineer, Pierre Bézier, who used them for the body design of the Renault car in the 1970's. Consider n+1 control points P_k (k=0..n) in 3D space. The Bézier parametric curve function is of the form

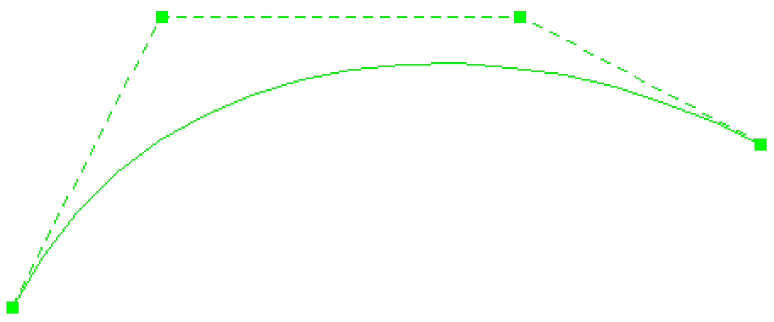


Figure 2.1: A cubic (degree 3) Bézier curve

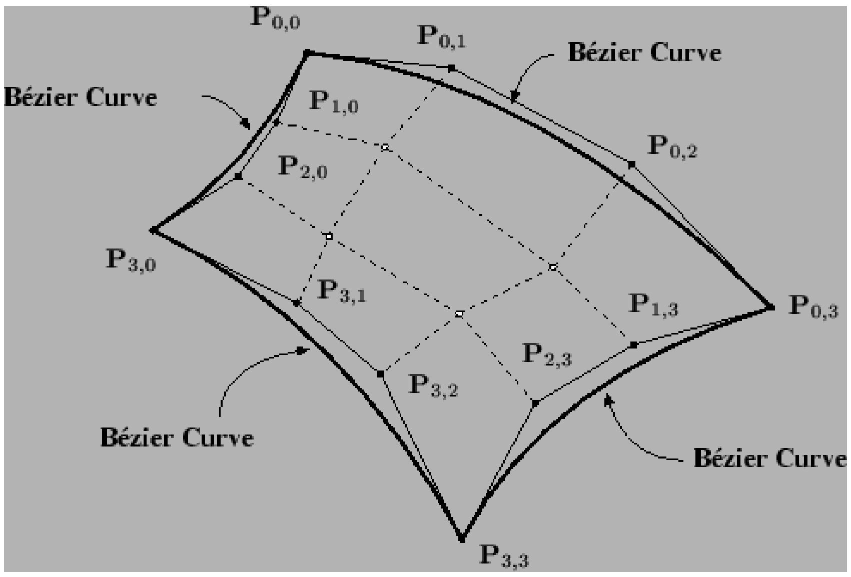


Figure 2.2: Bézier patch

$$C(u) = \sum_{k=0}^{n} P_k B_k^n(u) \qquad (0 \leq u \leq 1) \tag{2.1}$$

B(u) is a Bernstein polynomial[1] and it is defined by:

$$B_k^n(u) = \frac{n!}{k!(n-k!)} u^k (1-u)^{n-k} \tag{2.2}$$

The extension of Bézier curves to surfaces is called the Bézier patch. The patch is constructed from an (n+1)×(m+1) array of control points $\{P_{i,j} : 0 \leq i \leq n, 0 \leq j \leq m.\}$ And the resulting surface, which is now parameterized by two variables, is given by the equation

$$P(u,v) = \sum_{j=0}^{m} \sum_{i=0}^{n} P_{i,j} B_{i,n}(u) B_{j,m}(v) \tag{2.3}$$

The Bézier patch is the surface extension of the Bézier curve (see Figure 2.2). Whereas a curve is a function of one variable and takes a sequence of control points, the patch is a function of two variables with an array of control points. Most of the methods for the patch are direct extensions of those for the curves.

Some important properties of Bézier curves/surfaces are listed below:

- **Endpoint interpolation**: Bézier curves interpolate the first and the last control points: $P_0 = C(0)$ and $P_n = C(1)$; Bézier surfaces interpolate the four corner control points.
- **Smoothness**: With a Bezier curve, we can only get a smooth curve of the given degree, with continuous derivative everywhere, by defining a sequence of curves, each curve defined by (degree+1) points. The directions of $P_1 - P_0$ and $P_n - P_{n-1}$ are always tangent to the Bézier curves at the two endpoints P_0 and P_1 respectively. Hence, in order to make two Bźier curves join smoothly, we need to put the last two control points of the first curve and the first two of the second curve in a line.
- **Convex hull**: the curves are contained in the convex hulls of their defining control points;

[1]See Chapter 2.2 for explanation of polynomial

- **Variation diminishing**: no straight line/plane intersects a curve more times than it intersects the curve's control polygon;
- **Affine invariance**: one can apply to Bézier curves/surfaces the usual transformations, such as rotations, translations, and scalings, by just applying them to the control polygon/net;
- **Partition of unity**: For curve, $\sum_{i=0}^{n} B_k^n(u) = 1$ for all $0 \leq u \leq 1$; For surface, $\sum_{j=0}^{m} \sum_{i=0}^{n} B_{i,n}(u)B_{j,m}(v) = 1$ for all $0 \leq u \leq 1$ and $0 \leq v \leq 1$.

2.2 B-Spline

A B-spline (Figure 2.3) is a generalization of the Bézier curve. The theory for B-splines was first suggested by a Romanian mathematician Schoenberg in 1946. A B-spline curve of degree p is defined by n+1 control

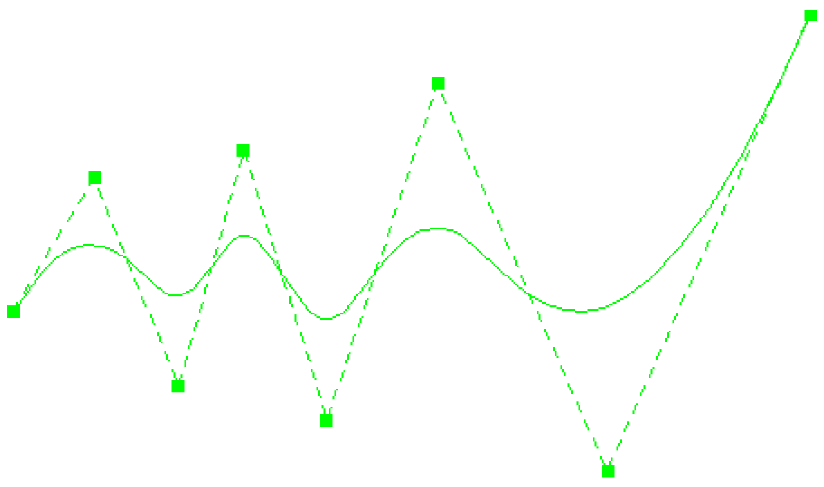


Figure 2.3: A B-spline curve with 8 control points

points $P_0, ..., P_n$ and a knot vector of m+1 knots:

$$T = \{t_0, t_1, ..., t_m\}$$

where T is a nondecreasing sequence with $t_i \in [0, 1]$, and n, m and p must satisfy:

$$p \equiv m - n - 1 \tag{2.4}$$

The B-spline parametric curve function is of the form:

$$C(t) = \sum_{i=0}^{n} P_i N_{i,p}(t) \tag{2.5}$$

$N_{i,p}(t)$ is the basis functions of B-splines, defined by:

$$N_{i,0}(t) = \begin{cases} 1 & \text{if } t_i \leq t \leq t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t) \tag{2.6}$$

Equation 2.6 is also called the Cox-de Boor recursion formulas [3].

A B-Spline surface is an expansion of B-spline curves in two directions, with corresponding control points, knot vectors, and univariate B-spline functions. The surface is defined by

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

where the k,l are the orders (degree+1) of the B-spline surface in both directions. The $N_{i,k}(u)$ are the polynomial B-spline basis functions of degree k-1 in u parameter direction, and $N_{j,l}(v)$ are the basis functions of degree l-1 in v direction.

Here are some important properties of B-spline curves/surfaces:

- **Piecewise polynomial curve**: A B-spline curve is a piecewise polynomial curve.

  A **polynomial** is a mathematical expression involving a sum of powers in one or more variables multiplied by coefficients. For example, an univariate polynomial[1] is defined by

  $$f(x) = \sum_{i=0}^{n} a_i x^i \tag{2.8}$$

  A bivariate polynomial[2] is defined by

  $$f(x,y) = \sum_{i=0}^{n} \sum_{j=0}^{m} a_{i,j} x^i y^j \tag{2.9}$$

  The highest power in a polynomial is called its **degree**. A piecewise polynomial curve is obtained by fitting polynomials to each coordinate of an ordered sequence of points, which split the curve into several segments. Each segment of the piecewise polynomial curve is constructed so that they join with some level of **continuity**. Continuity is a general mathematical property obeyed by mathematical objects in which all elements are within a neighborhood of nearby points. [25].

  As B-spline can be viewed as the union of curve segments defined on each knot interval, its degree is **independent** of the number of control points.

- **Smoothness**: A p-degree B-spline curve/surface is p-k times continuously differentiable at the knots of multiplicity k (k knots clamped together). It is very difficult to visualize the difference of smoothness for the parts with continuity bigger than 2. For continuity 1, the corresponding point lies on the control polygon, while the continuity 0 forces the curve to pass through a control point, which results a visual discontinuity. Figure 2.4 shows a 4-degree B-spline curve. It is easy to calculate that the curve at the three multiple knots has continuities of 2, 1, 0 respectively. Note that there's a cusp at the multiple knots with continuity 0.

- **Strong convex hull**: The curves/surfaces are contained in the convex hulls of their defining control points. Especially, for B-spline curve, if $u \in [t_i, t_{i+1}), p \le i \le m-p-1$, then the curve C(t) is in the convex hull of the control points $P_{i-p}, ..., P_i$. For a B-spline surface with a knot vector $\{u_0,\ u_1, ...,\ u_m\}$ in u direction and a knot vector $\{v_0,\ v_1, ...,\ v_n\}$ in v direction, if $(u,v) \in [u_{i_0}, u_{i_0+1}) \times [v_{j_0}, v_{j_0+1})$, then the surface S(u,v) is in the convex hull of the control points $P_{i,j}$, $i_0 - p \le i \le i_0$ and $j_0 - q \le j \le j_0$;

- **Variation diminishing**: No straight line/plane intersects a curve more times than it intersects the curve's control polygon;

- **Affine invariance**: An affine transformation is any transformation that preserves collinearity (i.e., all points lying on a line initially still lie on a line after transformation) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after transformation). In general, an affine transformation is a composition of rotations, translations, dilations, and shears, and they are all subclasses of projective transformations [25]. For a B-spline curve/surface, one can apply the affine transformations by just applying them to the control polygon/net.

[1]A polynomial in one variable.

[2]A polynomial in two variables.

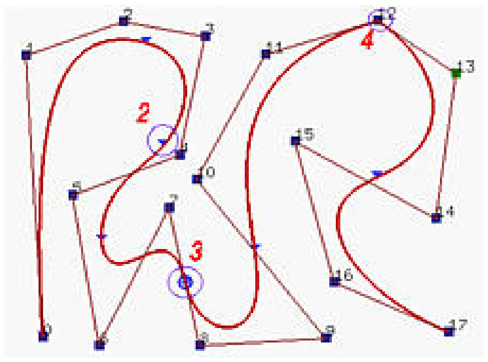


Figure 2.4: A B-spline curve with different continuity. The multiple knots are marked with circles; the numbers beside the circles are the multiplicities of the knots.

- **Partition of unity**: For curve, $\sum_{i=0}^{n} N_{i,p}(t) = 1$ for all $0 \le t \le 1$; for surface, $\sum_{i=0}^{m} \sum_{j=0}^{n} N_{i,k}(u)N_{j,l}(v) = 1$ for all $0 \le u \le 1$ and $0 \le v \le 1$;
- **Local modification scheme**: Moving P_i changes a B-spline curve only in the interval $[t_i, t_{i+p+1}]$, because $N_{i,p}(t) = 0$ for $u \notin [t_i, t_{i+p+1}]$; similarly, moving $P_{i,j}$ only affects a B-spline surface in the rectangle $[u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$, because $N_{i,p}(u)N_{j,q}(v)$ is zero if (u,v) is outside of the this rectangle.

2.3 NURBS

A NURBS (Nonuniform Rational B-Splines) curve C(u) (Figure 2.5), which is a vector-valued piecewise rational polynomial function, is defined as:

$$C(u) = \frac{\sum_{i=0}^{n} w_i P_i N_{i,k}(u)}{\sum_{i=0}^{n} w_i N_{i,k}(u)} \tag{2.10}$$

where
w_i: weights
P_i: control points (vector)
$N_{i,k}$: normalized B-spline basis functions of degree k, defined by Equation 2.6.
A NURBS surface is defined in a similar way:

$$S(u,v) = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} w_{i,j} N_{i,k}(u) N_{j,l}(v) P_{ij}}{\sum_{i=0}^{n} \sum_{j=0}^{m} w_{i,j} N_{i,k}(u) N_{j,l}(v)} \tag{2.11}$$

NURBS differ from B-splines mainly in two aspects: **nonuniform**, which means the knots of NURBS are unequally placed; **rational**, which means the control points of NURBS have weight values. Below are some important properties of NURBS curves/surfaces.

- **Piecewise rational polynomial curve**: A NURBS curve is a piecewise rational polynomial curve. A rational curve is represented with rational functions of the form

$$x(u) = \frac{X(u)}{W(u)}$$

$$y(u) = \frac{Y(u)}{W(u)}$$

  where X(u), Y(u),and W(u) are polynomials, that is, each of the coordinate functions has the same denominator [2].

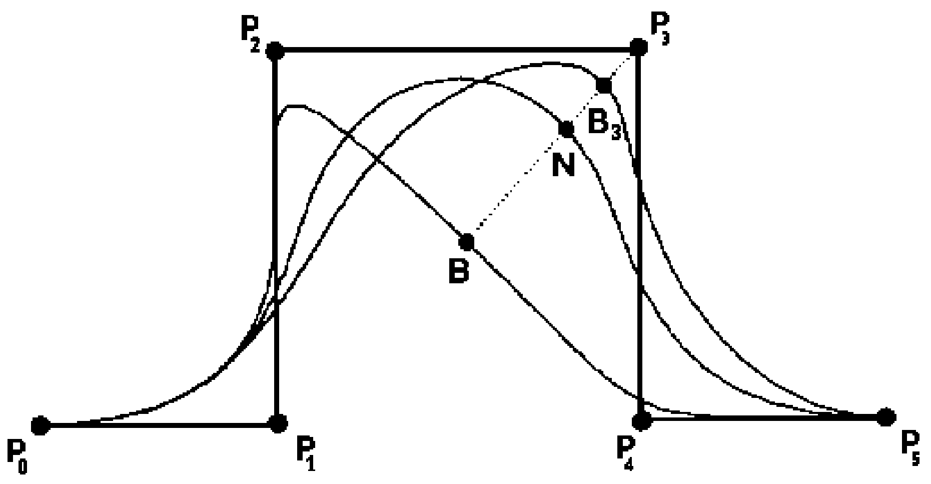


Figure 2.5: A NURBS curve with 6 control points. B, N, and B_3 illustrate the change of a point's position (same u) when the weight of P_3 is 0/1/(bigger than 1) respectively. [9]

- **Endpoint interpolation**: NURBS curves interpolate the first and the last control points: $P_0 = C(0)$ and $P_n = C(1)$; NURBS surfaces interpolate the four corner control points.
- **Smoothness**: A p-degree NURBS curve/surface is p-k times continuously differentiable at the knot of multiplicity k. The NURBS curves/surfaces are smooth with continuity larger than 1, and are less smooth with continuity 1 and 0. This property is similar to the same property of B-splines.
- **Strong convex hull**: The curves/surfaces are contained in the convex hulls of their defining control points. Especially, for NURBS curve, if $u \in [u_i, u_i+1), p \leq i \leq m-p-1$, then C(u) is in the convex hull of the control points $P_{i-p}, ..., P_i$. For NURBS surface, if $(u, v) \in [u_{i_0}, u_{i_0+1}) \times [v_{i_0}, v_{i_0+1})$, then S(u,v) is in the convex hull of the control points $P_{i,j}$, $i_0 - p \leq i \leq i_0$ and $j_0 - q \leq j \leq j_0$;
- **Variation diminishing**: No straight line/plane intersects a curve more times than it intersects the curve's control polygon;
- **Projective invariance**: One can apply to curves/surfaces not only the affine transformations, but also the projective transformations, by just applying them to the control polygon/net;
- **Partition of unity**: For curve, $\sum_{i=0}^{n} N_{i,p}(t) = 1$ for all $0 \leq t \leq 1$; for surface, $\sum_{i=0}^{m} \sum_{j=0}^{n} N_{i,k}(u)N_{j,l}(v) = 1$ for all $0 \leq u \leq 1$ and $0 \leq v \leq 1$;
- **Local modification scheme**: Moving P_i or changing its weight change a NURBS curve only in the interval $[t_i, t_{i+p+1}]$; and moving $P_{i,j}$ or changing its weight only affect a NURBS surface in the rectangle $[u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$.

2.4 Choosing a representation

Various properties hold for Bézier, B-spline and NURBS. A comparison of these representations would be quite helpful in order to choose the most appropriate representation for 3D freeform curves and surfaces. The properties mentioned in previous subsections are compared and listed in Figure 2.6.

From the comparison it is clear that NURBS and B-spline hold more helpful properties than Bézier because:

- The NURBS and B-spline curves/surfaces are **smoother** than Bézier curves/surfaces.
- The **local modification scheme** property of NURBS and B-spline is very important to curve design, because we can modify a curve locally without changing the shape in a global way.

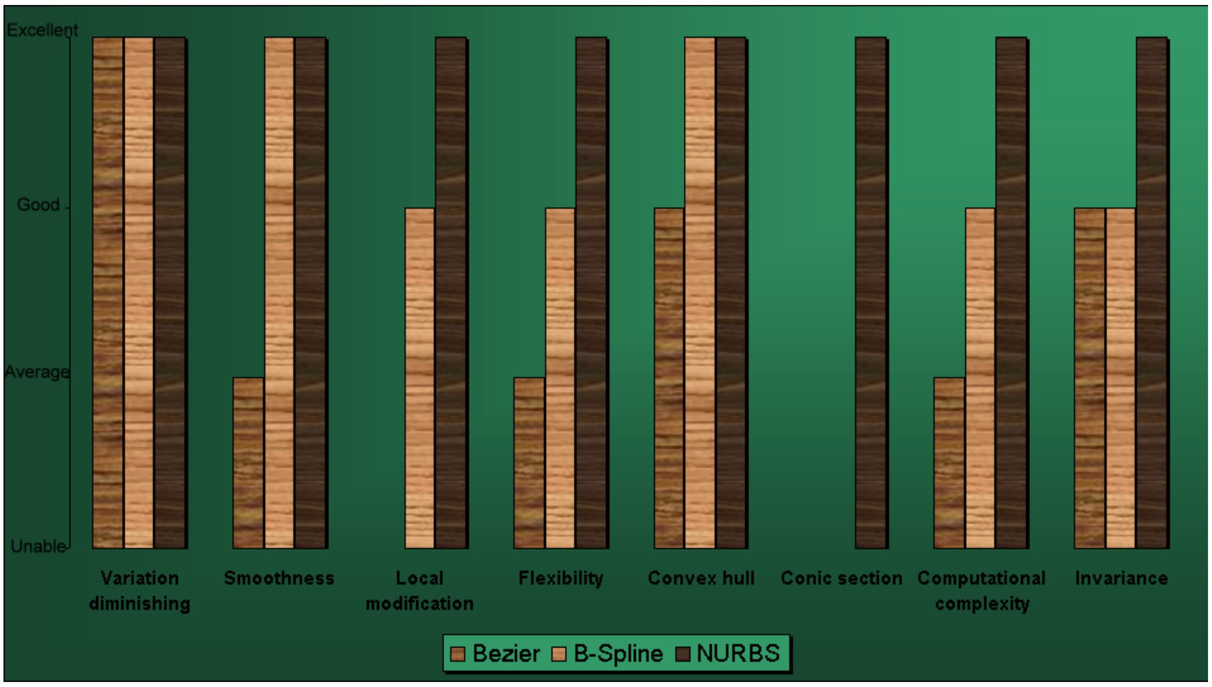


Figure 2.6: Comparion between Bézier, B-spline and NURBS

- The NURBS and B-spline curves/surfaces can be modelled with more parameters than Bézier curves/surfaces, which give more **flexibility** to shape design.

And NURBS are better than B-spline because:

- For the **local modification scheme** property, NURBS can change the curves/surfaces not only by moving the control points, but also by changing their weights[1].
- NURBS can represent exactly the **conic sections**, i.e. circles, ellipses, cones, cylinders. Such curves occur very frequent in CAD/GIS applications, where several shapes and geometric constructions are based on such geometric primitives [2].
- When we want to apply projective transformations to a NURBS curve/surface, **projective invariance** property guarantees that we can apply the transformations to control points and the transformed NURBS curves/surfaces are defined by the transformed control points, leaving to the visualization application the task of rendering the transformed curves/surfaces;

Although NURBS requires more complex algorithms to compute, it can be acceptable considering the extraordinary properties by NURBS. From the above analysis, NURBS is chosen as the mathematical representation for 3D freeform curves and surfaces in this report.

[1]This is very important for interactive shape design. Read next chapter for more explanation of NURBS's local modification scheme.

Chapter 3

NURBS

Non-uniform rational B-splines are normally denoted as NURB splines or simply as NURBS. These splines are very important for both graphics and CAD applications, and have been recognized by many international standards, e.g., IGES, STEP, and PHIGS, as powerful tools for geometric design [2]. Based on the introduction of NURBS in Chapter 2, this chapter continues with more advanced explanations of NURBS, including fundamental algorithms, techniques for constructing freeform solids, and some NURBS-based examples.

3.1 Representation

The definition of NURBS is already given in 2.10 and 2.11. Different parts of this definition are explained below for better understanding of the NURBS representation.

3.1.1 Knot vector

From Chapter 2.2 we know that knot vectors are nondecreasing sequences in [0,1]. If a knot t_i appears k times (i.e., $t_i = t_{i+1} = ... = t_{i+k-1}$), where $k > 1$, t_i is a multiple knot of multiplicity k. Otherwise, if t_i appears only once, it is a simple knot.

If the knots are equally placed (i.e., t_{i+1} - t_i is a constant for all knots), the knot vector or the knot sequence is called uniform, for example:

[0 0.2 0.4 0.6 0.8 1]

Otherwise, if the knots are unequally placed, it is called nonuniform. For example:

[0 0.3 0.75 0.95 1]

3.1.2 Basis functions

With its basis functions, a NURBS curve is contained in the **convex hull** of its control points. Moreover, if u is in knot span $[u_i, u_{i+1}]$, then the curve is in the convex hull of the control points $P_{i-p}, P_{i-p+1}, ..., P_i$.

NURBS basis functions also enable **local modification scheme**: changing the position/weight of a control point P_i only affects the curve C(u) on the interval $[u_i, u_{i+p+1}]$. Recall that $N_{i,p}(u)$ is non-zero on interval $[u_i, u_{i+p+1}]$. If u is not in this interval, $N_{i,p}(u)$ is zero and $N_{i,p}(u)w_iP_i$ has no effect in computing C(u). On the other hand, if u is in the indicated interval, $N_{i,p}(u)$ is non-zero, and if $N_{i,p}(u)P_i$ is changed, so is C(u). This local modification scheme is very important to curve design, because we can modify a curve locally without changing the shape in a global way. Moreover, if fine-tuning of a curve shape is required, one can insert more knots (and therefore more control points) so that the affected region could be restricted to a more narrow one.

3.1.3 Weights

The weights of NURBS w_i affect only the range $[t_i, t_{i+p+1}]$, and they should always be **non-negative**.

If $w_i = 0$ (point B in Figure 2.5) then P_i has no effect on the shape.

If w_i increases, the curve/surface is pulled toward P_i, and the other way around. Furthermore, for a fixed knot vector, the movement by changing a control point's weight is along a straight line through this control point.

If all weights are equal, a NURBS curve/surface becomes a B-spline curve/surface. If furthermore the number of control points n is equal to the degree p, and there are 2(p + 1) = 2(n + 1) knots with p + 1 of them clamped at each end, or in other words, the knot vector is just k zeros followed by k ones, then the NURBS curve/surface reduces to a Bézier curve/surface. So actually B-spline and Bézier are special cases of NURBS.

3.1.4 Rational curves and surfaces

Curves/surfaces that are defined with a weight w_i for each control point, are called rational curves. Mathematically, such curves/surfaces are defined in four-dimensional space (since there're four components in a control point, three for coordinate and one for weight) and are projected to three-dimensional space.

If a projective transformation is applied to a NURBS curve/surface, the result can be constructed from the projective images of its control points. This is a nice property. When we want to apply affine transformations or even projective transformations to a NURBS curve/surface, this property guarantees that we can apply the transformation to the control points, and the transformed NURBS curve/surface is defined by the transformed control points. Therefore, we do not have to transform the curve/surface. Bézier curves and B-spline curves only satisfy the affine invariance property rather than this **projective invariance** property. This is because only NURBS curves involve projective transformations.

3.2 Fundamental algorithms

Three fundamental algorithms for implementing NURBS are given in this part: evaluation algorithm, knot insertion and degree elevation. There are more advanced algorithms for NURBS, such as point inversion and projection, surface tangent vector inversion, transformations and projections, etc., Detailed descriptions of these algorithms can be found in [2], and they are not going to be explained here for content limitation.

3.2.1 Evaluation algorithm

An effective evaluation algorithm is given in [2] by using homogeneous coordinates.

1. Add one dimension to the control points (e.g. $P(x, y, z) \rightarrow P'(x, y, z, 1)$) and multiply them by their corresponding weights (i.e. in 3D: $P_i(x_i, y_i, z_i) \rightarrow P'_i(w_i x_i, w_i y_i, w_i z_i, w_i)$).

2. Calculate NURBS in homogeneous coordinates:

$$C'(u) = \sum_{i=0}^{n} P'_i N_{i,k}(u)$$

3. Map the points on "homogeneous" NURBS back to the original coordinate system with:

$$map(x, y, z, W) = \begin{cases} (x/W, y/W, z/W), & \text{if W} \neq 0 \\ (x, y, z), & \text{if W=0} \end{cases}$$

$$C(u) = map(C'(u))$$

Nearly every algorithm on NURBS starts with these "projection" steps.

3.2.2 Knot insertion

Knot insertion is an important algorithm for NURBS because it can be used for:

1. evaluating points and derivatives on curves and surfaces.
2. subdividing curves and surfaces.
3. adding control points in order to increase flexibility in shape control (interactive design).

Given a set of n+1 control points $P_0, P_1, ..., P_n$, a knot vector of m+1 knots $U = \{u_0, u_1, ..., u_m\}$ and a degree p, we want to insert a new knot t into the knot vector without changing the shape of the B-spline curve C(u). Suppose the new knot t lies in the knot span $[u_k, u_{k+1}]$. From the strong convex hull property, C(t) lies in the convex hull defined by control points $P_k, P_{k-1}, ..., P_{k-p}$ and the basis functions of all other control points are zero. Thus, the knot insertion computation can be restricted to control points $P_k, P_{k-1}, ..., P_{k-p}$. The way of inserting t is to find p new control points Q_k on segment $P_{k-1}P_k$, Q_{k-1} on segment $P_{k-2}P_{k-1}$, ..., and Q_{k-p+1} on segment $P_{k-p}P_{k-p+1}$ such that the old polyline between P_{k-p} and P_k is replaced by $P_{k-p}Q_{k-p+1}Q_{k-p+2}...Q_{k-1}Q_kP_k$ by cutting the corners at $P_{k-p+1}, ..., P_{k-1}$. All other control points are unchanged. Note that p-1 control points of the original control polyline are removed and replaced with p new control points, as in Figure 3.1.

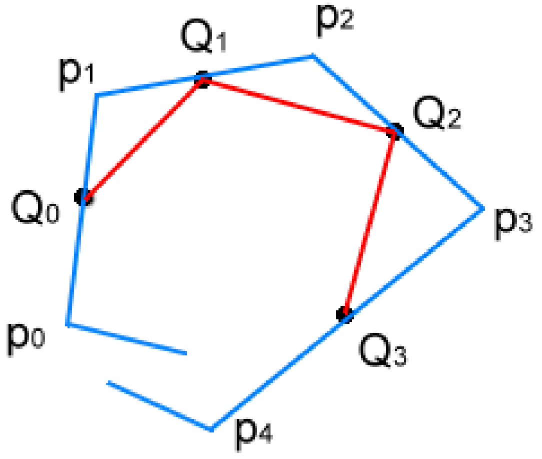


Figure 3.1: Knot insertion

De Boor's Algorithm [3] gives the formula for computing the new control point Q_i on segment $P_{i-1}P_i$ as the following:

$$Q_i = (1 - a_i)P_{i-1} + a_iP_i \tag{3.1}$$

where ratio a_i can be computed by:

$$a_i = \frac{t - u_i}{u_{i+p} - u_i} \tag{3.2}$$

In summary, to insert a new knot t, we first find the knot span $[u_k, u_{k+1}]$ that contains t. With k available, p new control points $Q_{k-p+1}, ..., Q_k$ are computed with the above formula. Finally, the original control polyline between P_{k-p} and P_k is replaced with the new polyline defined by P_{k-p}, Q_{k-p+1}, Q_{k-p+2}, ... , Q_{k-1}, Q_k, P_k. Note that after inserting a new knot, the knot vector becomes u_0, u_1, ..., u_k, $\mathbf{t}$, u_{k+1}, ..., u_m. If the new knot t is equal to u_k, the multiplicity of u_k is increased by one.

Knots are inserted into surfaces by simply applying the previous formulas and algorithms to the rows and/or columns of control points.

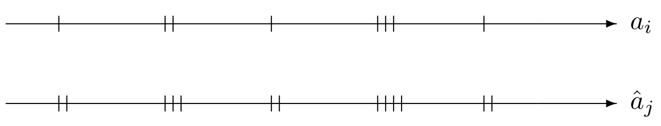


Figure 3.2: Degree elevation

3.2.3 Degree elevation

Since a B-spline curve is a piecewise polynomial curve, it is possible to raise its degree from k to k+m, where m is an integer greater than or equal to 1. Thus, there must exist control points Q_i and a new knot vector $T = [t_0, ..., t_{n+k+m}]$ such that

$$P(t) = Q(t) = \sum_{i=0}^{n} N_{i,k+m}(t) Q_i \tag{3.3}$$

where n+1 is the number of control points of Q(t), and $N_{i,k+m}(t), i = 0, ..., n$ are the B-spline basis functions of degree k+m+1 defined on the knot vector T. The curves P(t) and Q(t) have the same geometry and parameterization. The computation of n, Q_i, and T is referred to as **elevating the degree** of the curve.

Degree elevation is important because quite often a designer wants to manipulate a curve/surface into the shape he desires, but the degree limit of the parametric representation makes it impossible. Degree elevation is the right operation to elevate the flexibility of the representation.

Due to the basis properties of B-spline, we can write a spline of degree n:

$$s(u) = \sum_{i=0}^{n} c_i N_i^n(u)$$

with some knot sequence (a_i) also as a linear combination of B-splines of degree n+1:

$$s(u) = \sum_{j=0}^{n} d_j \hat{N}_j^{n+1}(u)$$

with the knot sequence$(\hat{a}_j)$ obtained from (a_i) by raising the multiplicity of each knot a_i by one, as illustrated in Figure 3.3.

Various papers (Piegl [2], Prautzsch and Piper [4] and Cohen [10]) give fast and efficient methods for degree elevation. Both [10] and [4] elevate the degree by 1, whereas the method by Cohen [10] gives stable and efficient algorithms for low-degree special cases, such as linear to quadratic, quadratic to cubic, and Prautzsch and Piper's method [4] is for general case. The method by Piegl [2], which achieve the same result by extracting the Bézier segments from the curve, is more competitive in the case where the degree is to be raised by more than 1. Here we present the algorithm by Piegl [2].

Because the algorithm of degree elevation for Bézier curve will be used in Piegl's algorithm, we first give a brief introduction of how to elevate Bézier curve's degree by one:

Bézier degree elevation The degree of a Bézier curve can be increased by increasing the number of control points. For a Bézier curve with n+1 control points $p_0 ... p_n$, the new representation with n+2 control points $\hat{p}_0 ... \hat{p}_{n+1}$ is given by:

$$C(u) = \sum_{k=0}^{n} p_k B_k^n(u) = \sum_{k=0}^{n+1} \hat{p}_i B_k^{n+1}(u)$$

where

$$\hat{p}_0 = p_0$$

$$\hat{p}_i = a_i p_{i-1} + (1 - a_i) p_i \qquad a_i = \frac{i}{n+1}$$

$$\hat{p}_{n+1} = p_n$$

Piegl's NURBS degree elevation Piegl's NURBS degree elevation can be summarized to three steps:

1. The NURBS curve is decomposed into piecewise Bézier curve segments by inserting k-1 multiple knot values at each internal knot.
2. These Bézier curve segments are degree elevated.
3. Remove unnecessary knot values.

The first step can be fulfilled using an algorithm developed in [2] (p173-176) which decomposes a NURBS curve and returns Bézier segments.

The second step is just the Bézier degree elevation algorithm.

The third step is related to another fundamental algorithm for NURBS: knot removal. Knot removal is the reverse operation of knot insertion. Here we only give the formulas for knot removal. Detailed explanation and proof can be found in many papers [6][10][2][4].

Knot removal Let $u = u_r \neq u_{r+1}$ be a knot of multiplicity s, where $1 \leq s \leq p$ (degree). The equations for computing the new control points for one removal of u are

$$P_i^1 = \frac{P_i^0 - (1 - a_i) P_{i-1}^1}{a_i} \qquad r - p \leq i \leq \frac{1}{2}(2r - p - s - 1)$$

$$P_j^1 = \frac{P_j^0 - a_j P_{j+1}^1}{1 - a_j} \qquad \frac{1}{2}(2r - p - s + 2) \leq j \leq r - s \tag{3.4}$$

with

$$a_k = \frac{u - u_k}{u_{k+p+1} - u_k} \qquad k = i, j$$

For a surface, one can elevate the degree in either the u- or the v- direction. In this case, every row or column of control points will be modified by the curve elevation algorithm for curves.

3.3 Free-form solids

Boundary representation There're many representations for solids, among which boundary representation is the most suitable one for freeform solids. Boundary representations represent a solid indirectly by its boundary faces, which can be represented by their edges, which can again be represented by their vertices.

The boundary representation satisfies **Euler's formula**, which expresses an invariant relationship among the number of faces f, edges e, and vertices v:

$$v - e + f = 2 \tag{3.5}$$

This is only valid for simple objects without internal holes. For general cases, boundary representation should satisfy:

$$v - e + f = 2(s - h) + r \tag{3.6}$$

where r is the number of rings (loops) in faces, h is the number of holes through an object, and s is the number of disjoint parts (shells) [11].

Trimmed NURBS surfaces When NURBS surface patches are joined together to make a boundary representation of solid models, only portions of a tensor product surface are often used. These may be specified by giving a set of trimming curves in parameter (u,v) space of the patch. Such curves, usually given as 2D NURBS splines, are converted into a set of closed loops by a suitable combination with the boundary of parameter space. The used portions of the surface are those internal to the loops, according to some loop orientation. Notice that in order to have a meaningful description, the loops must not intersect themselves or each other. The geometry representation of the boundary using **trimmed NURBS surfaces** can represent freeform solids of nearly every shape, as well as their Boolean combinations. Trimmed NURBS are currently the industry standard representation of surfaces within the kernel geometric libraries adopted by most CAD systems [7]. A typical trimmed NURBS surface with

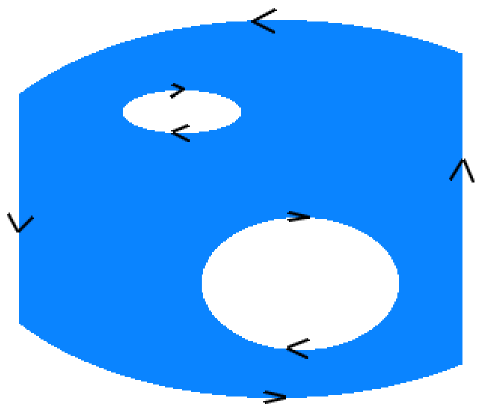


Figure 3.3: Trimmed NURBS surface

three trimming loops is shown in Figure 3.3. It has one outer trimming loop and two inner trimming loops. The trimming loops for a trimmed surface are arranged such that the outer trimming loop is counter-clockwise and all the inner trimming loops are clockwise with respect to the outward normal in the parametric domain.

Offset NURBS NURBS curves can also be constructed by moving an existing NURBS curve $C_{base}(u)$ along a certain direction N(u) over an offset d in 3D space. This is called offset NURBS. The parametric form of the offset NURBS curves is given by:

$$C(u) = C_{base}(u) + dN(u)$$

where d is the offset distance, and N(u)/N(u,v) are the unit normals computed at u/(u,v) [8].

Swung NURBS Swung by NURBS is also a solid construction method for freeform solids. Let a NURBS curve P(u) be a profile curve defined in x-z plane, and let another NURBS curve T(v) be trajectory curve defined in x-y plane, we can obtain a NURBS surface by swinging P(u) about z-axis and simultaneously scaling it according to T(v). So this swung surface is:

$$S(u,v) = (aP_x(u)T_x(v), aP_x(u)T_y(v), P_z(u))$$

where a is an arbitrary scaling factor.

Swept NURBS NURBS surfaces can also be constructed by traversing NURBS curve C(u) along a NURBS curve path T(v) in 3D space. A general form of swept surface S(u, v) is given by:

$$S(u, v) = T(v) + M(v)C(u)$$

where M(v) is a 3×3 matrix incorporating rotation and nonuniform scaling of C(u) as a function of v. If M(v) is the identity matrix for all v, then Equation 3.3 becomes

$$S(u, v) = T(v) + C(u)$$

We call this surface a **generalized translational sweep** [2].

An example of swept NURBS is given in Chapter 3.4.3.

Skinned NURBS Skinning NURBS is a process of blending K NURBS curves (u direction) together in v direction to form a NURBS surface. These curves are commonly called section curves. Take Figure 3.4 as example: assume the series of curves are defined on the same knot vector, and have common degree (if not, we can make them uniform by knot insertion/removal and degree elevation). The process

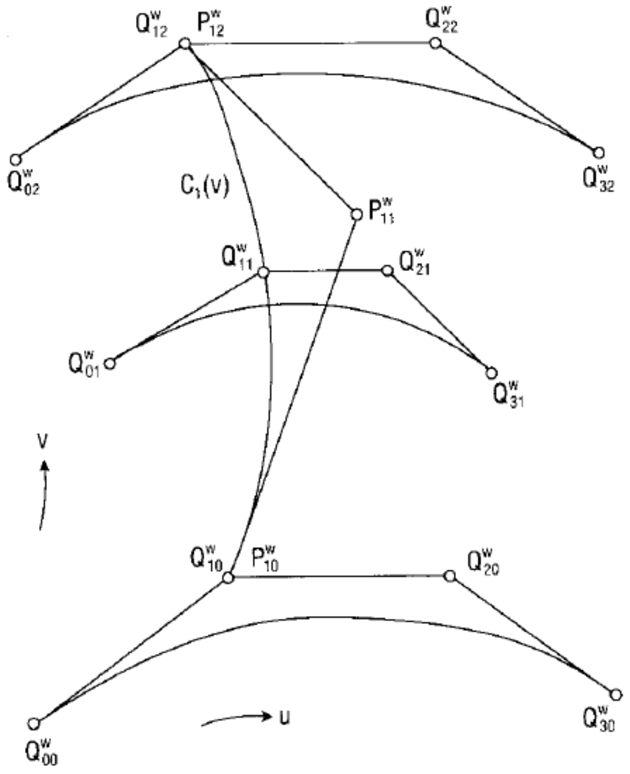


Figure 3.4: NURBS skinning: surface interpolation through cross-sectional curves [9].

is given as:

1. Choose degree q for direction v. The value of q is arbitrary, the only restriction is q $\leq K + 1$
2. Form a knot vector in v direction. This is done by first calculating the knot value in v direction for each NURBS curve, then determining the knot vector in v direction. Formulas for these calculations can be found in [2] (p462, p365).
3. Compute control points for skinned surface by interpolating the nth control points of separate curves with corresponding knot values. Note that the number of control points for all section

curves is the same because the knot vector and degree are both same. These three values satisfy Equation 2.4.

3.4 Examples

Three examples: a circle, a cylinder and a freeform surface, are given below to show NURBS' convenience in representing conics and freeform curves/surfaces:

3.4.1 Circle

A circle in x-y plane with its center at (0,0,0) and radius=1 (Figure 3.5).

Knots:

$$U = [0, 0, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, \frac{3}{4}, \frac{3}{4}, 1, 1, 1]$$

Weights:

$$w_i = [1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1]$$

Control points:

$$P_i = [(1,0,0), (1,1,0), (0,1,0), (-1,1,0), (-1,0,0), (-1,-1,0), (0,-1,0), (1,-1,0), (1,0,0)]$$

It is obvious that the degree of the circle is 2, so the NURBS formula for this circle is given as:

$$C(u) = \sum_{i=0}^{8} N_{i,2}(u) P_i^w$$

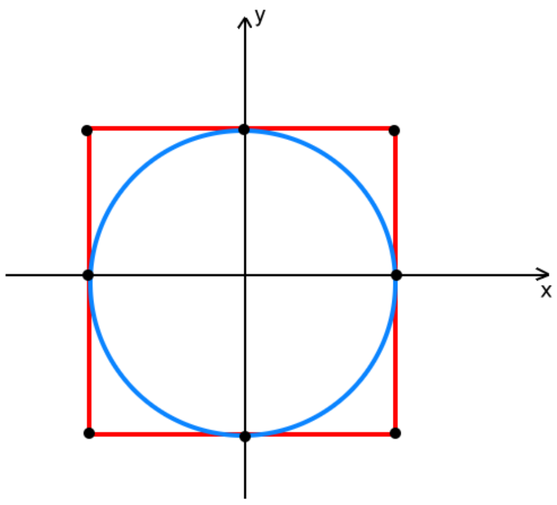


Figure 3.5: A NURBS Circle

3.4.2 Cylinder

A cylinder (Figure 3.6) can be constructed with the idea of offset NURBS. The moving of a NURBS curve C(u) along direction W for distance d can be mathematically defined as:

$$S(u,v) = C(u) + dW = \sum_{i=0}^{n} R_{i,p}(P_i + dW)$$

$R_{i,p}$ is another form of NURBS' basis function in Equation 2.10. It is defined by:

$$R_{i,p,j,q}(u) = \frac{N_{i,p}(u)N_{j,q}w_{i,j}}{\sum_{k=0}^{n}\sum_{l=0}^{m} N_{k,p}(u)N_{l,q}(v)w_{k,l}}$$

Because d is fixed in the v direction, the final representation is:

$$S(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{1} R_{i,p,j,q}(u,v)P_{i,j}$$

We can use the circle's NURBS representation in Chapter 3.4.1 here to represent the bottom circle of cylinders. Hence, a cylinder's knot vector, control points and weights in u direction are the same with the example in Chapter 3.4.1: U, P_i and w_i.

The knot vector in v direction is:

$$[0,0,1,1]$$

because for fixed u, S(u,v) is a straight line segment from C(u) to C(u)+dW.

The control points are:

$$P_{i,0} = P_i(i = 0,1,...,8)$$

$$P_{i,1} = P_{i,0} + dW$$

Weights:

$$w_{i,1} = w_i$$

$$w_{i,0} = w_{i,1}$$

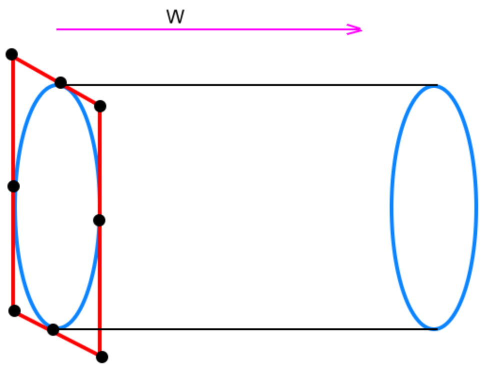


Figure 3.6: A NURBS cylinder

3.4.3 A freeform surface

The surface in Figure 3.7 is a NURBS surface constructed by sweeping curve C1(u) along curve C2(v) without any rotation and nonuniform scaling. C1(u) is a NURBS curve with control points P_i^1 and weights w_i^1 (i=0,...,3) in y-z plane, and C2(v) is a NURBS curve with control points P_j^2 and weights w_j^2 (j=0,...,3) in x-z plane.

According to explanation of swept NURBS in Chapter 3.3, we know that M(v) for this surface is the identity matrix, because no rotation and nonuniform scaling is applied. Hence the surface can be described by the equation below:

$$S(u,v) = C1(u) + C2(v)$$

Knot vectors in u and v direction are the knot vectors of C1 and C2 respectively.

The control points are:

$$P_{i,j} = P_i^1 + P_j^2 \quad \text{i=0,...,3} \quad \text{j=0,...,3}$$

Weights:

$$w_{i,j} = w_i^1 w_j^2 \quad \text{i=0,...,3} \quad \text{j=0,...,3}$$

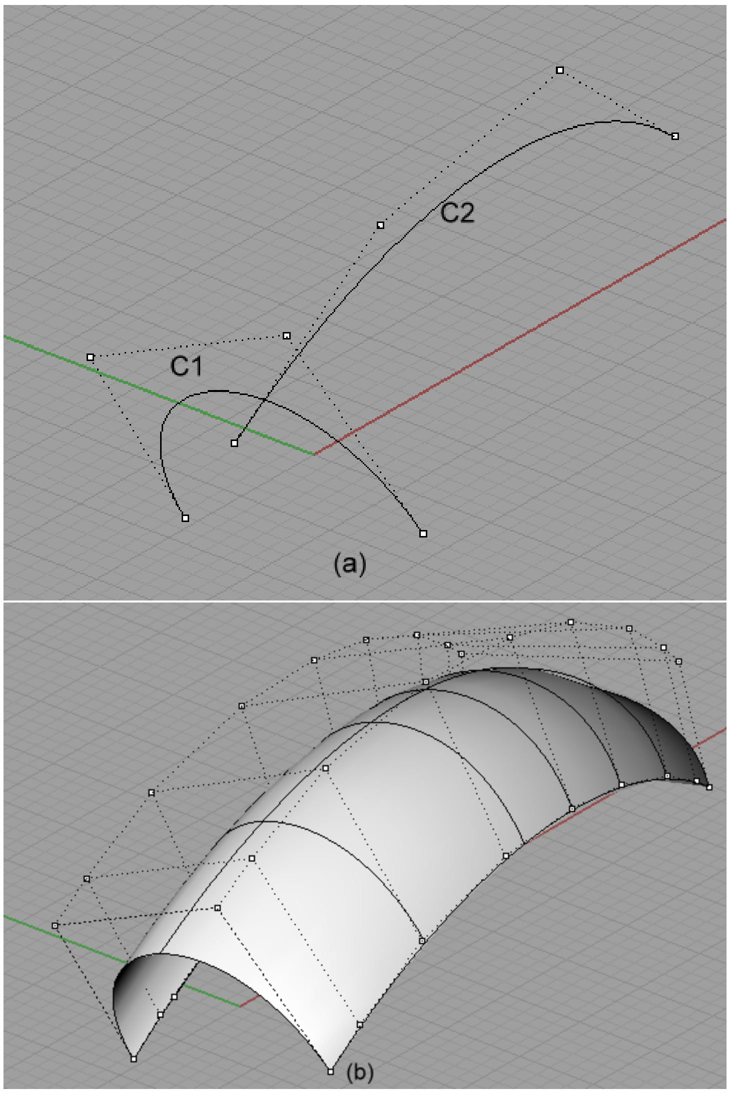


Figure 3.7: A freeform surface in perspective view (drawn with Rhinoceros 3.0)

Chapter 4

Freeform Curves/Surfaces in CAD Applications

Bézier, B-spline and NURBS play an important role in CAD world, and they have been widely used in current CAD applications. After the theoretical research in the previous chapters, now let's take a look at two popular CAD applications: AutoCAD 2004 and MicroStation V8, to learn how freeform curves/surfaces can be designed with the three representations in practice.

4.1 AutoCAD

AutoCAD is one of the most famous CAD applications nowadays. It is able to create and modify freeform curves/surfaces represented with Bézier, B-spline and NURBS [27].

The tool 'Spline' in AutoCAD 2004 is used to create NURBS curves. To define a NURBS curve, we need to give several **fit points**, which differ from control points in that NURBS curves pass (not exactly) through fit points. All the fit points have a **tolerance** value, which describes how closely the spline fits the set of fit points we specify. The lower the tolerance, the more closely the spline fits the points. At zero tolerance, the spline passes through the points. After defining all the fit points, **control points** for this curve will be calculated by AutoCAD 2004. These control points have equal **weights** 1 by default, which actually represent a B-spline. As a special case for B-Spline, we can get Bézier curves in AutoCAD 2004 when the order is one less than the number of control points.

Figure 4.1 illustrates the fit points, control points, and the tolerance's influence on curves.

Figure 4.2 (a) shows a NURBS curve in AutoCAD 2004. N

An existing NURBS curve can be edit by the tool 'SPLINEDIT'. Options for the edit can be

- Fit Data. Edits the fit point data that define the spline, including changing the tolerance.

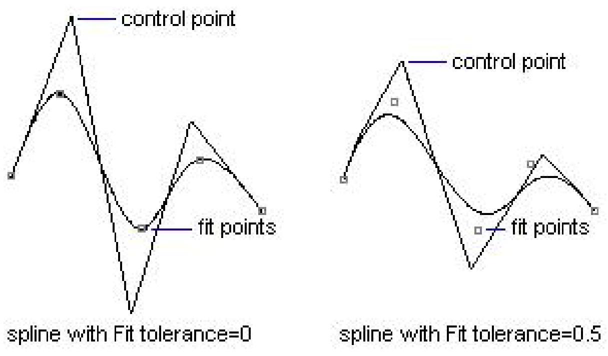
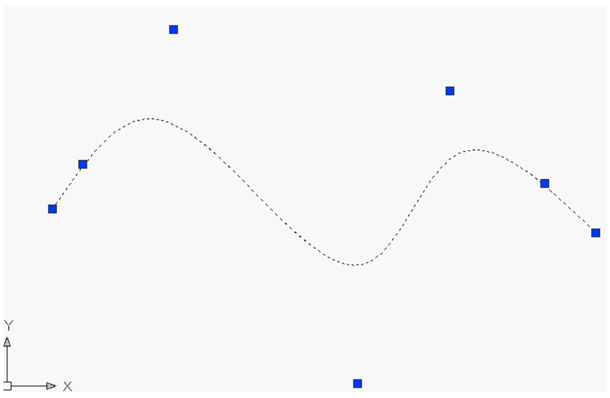
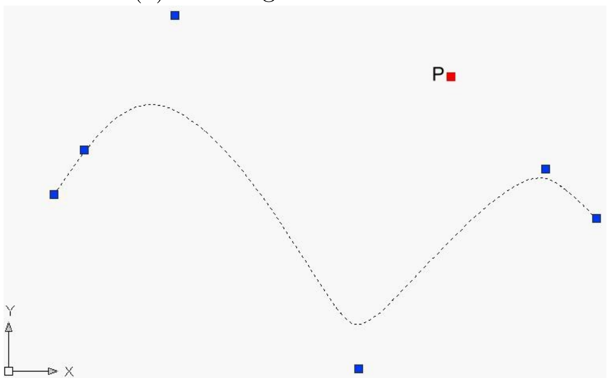


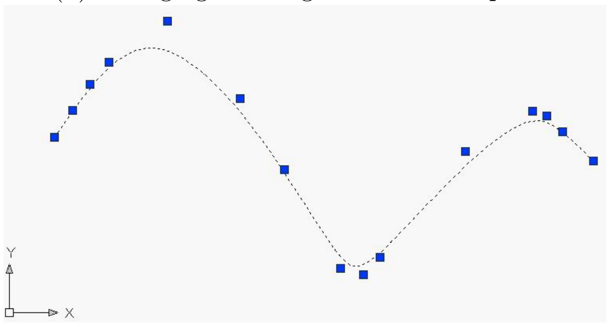
Figure 4.1: Control points, Fit points and Tolerance in AutoCAD 2004



(a) Creating a NURBS curve.



(b) Changing the weight of a control point.



(c) Elevating the order from 4 to 6.

Figure 4.2: Creating and editing a NURBS curve in AutoCAD 2004

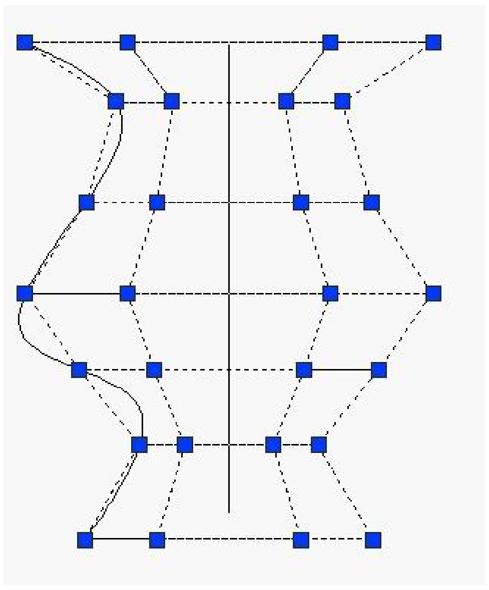


Figure 4.3: Surface of Revolution in AutoCAD 2004

- Close. Changes an open spline into a continuous, closed loop.
- Move Vertex. Moves a fit point to a new location.
- Refine. Modifies a spline definition by adding and weighting control points and elevating the order (maximum 26) of the spline.
- Reverse. Changes the direction of the spline.

Figure 4.2 (b) shows the change of the shape in Figure 4.2 (a) by changing a control point P's weight from 1.0 to 0.1 .

Figure 4.2 (c) shows the change of the shape in Figure 4.2 (b) by elevating the order from 4 to 6.

Freeform surfaces cannot be defined directly by giving fit points in AutoCAD 2004. They can only be constructed by a number of geometric construction methods, such as sweep, extrude and revolution. Figure 4.3 shows a surface constructed by revolving a NURBS curve about an axis (the vertical line in the middle).

4.2 MicroStation

MicroStation is another popular CAD application which is able to design freeform curves/surfaces. B-spline and Bézier[1] are supported in MicroStation, while NURBS is absent. In other words, the control points in MicroStation have no weights [24].

To define a B-spline, we can choose to define between **input data points** or **pick line string**. The **input data points** don't have to be control points, they can also be the points on the curve (like the fit points in AutoCAD), curve approximation points and Catmull-Rom points. The Catmull-Rom curve is popular with aircraft and ship hull designers. It passes directly through the data points or vertices on which it is based. If we want curves to be build by **pick line string**, the curve is constructed based on the vertices of an identified line string or complex chain (results in open B-spline) or shape or complex shape (results in closed B-spline).

Figure 2.3 shows a B-spline defined by 8 control points.

MicroStation enable us to do the following modifications to an existing B-spline:

- Move data point. Moves data points (control points, etc.) to new locations.

[1]Bézier is also treated as a special case of B-spline in MicroStation.

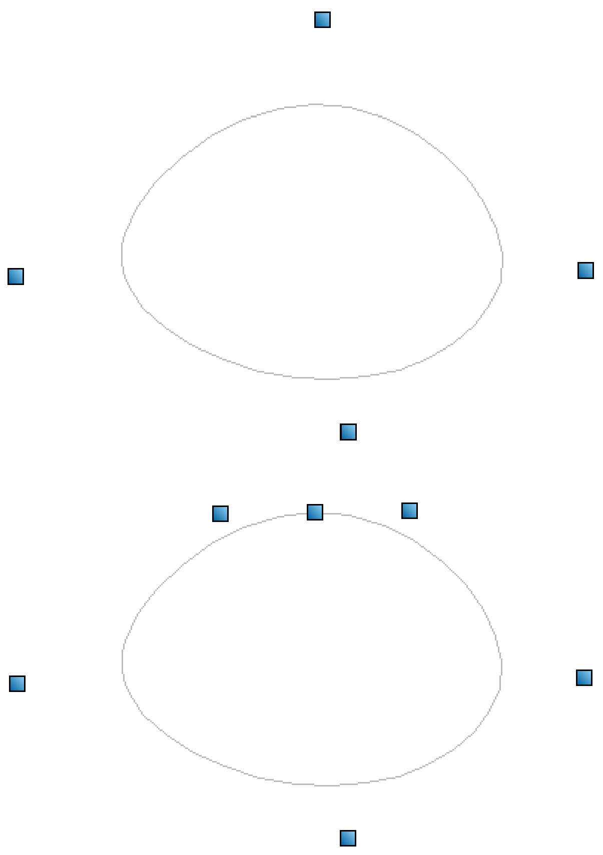


Figure 4.4: A closed B-spline curve converted to an open curve in MicroStation V8

- Closure. Convert a closed/open B-spline to a open/closed curve.
- Refine. Set the curve's order; deform the curve; flatten the curve; change the tolerance value.
- Reverse. Changes the direction of the spline.

Figure 4.4 shows the conversion of a closed B-spline to a open curve. Notice the change to its control points. Although this B-spline's shape remains the same, it is not closed any more.

To define B-spline surfaces, we can either input data points in two directions (u,v), which is similar with defining B-spline curves, or using geometry construction methods, which can be extruding, sectioning, sweeping, skinning, etc.

Figure 4.5 shows a B-spline surface constructed with data points in both u- and v-direction. Data points must be input row by row, and the number of data points for each row should be no less than the order in u-direction, the number of rows should be no less than the order in v-direction.

Figure 4.6 shows a B-spline surface constructed by three B-spline curves (can also use lines, line strings, arcs, ellipses, complex chains, or complex shapes). From Figure 4.6 it can be inferred that MicroStation probably uses skinning method to construct B-spline surfaces from B-spline curves.

4.3 Summary

AutoCAD and MicroStation both give nice support to freeform curves and surfaces. AutoCAD uses NURBS to represent freeform curves and surfaces, whereas MicroStation uses B-spline. Both of them can easily define freeform curves by specifying data points (control points/fit points), and define freeform surfaces by geometry construction methods (sweeping, skinning, revolution, etc.). Freeform surfaces can also be defined by specifying data points in u and v directions in MicroStation. After the freeform

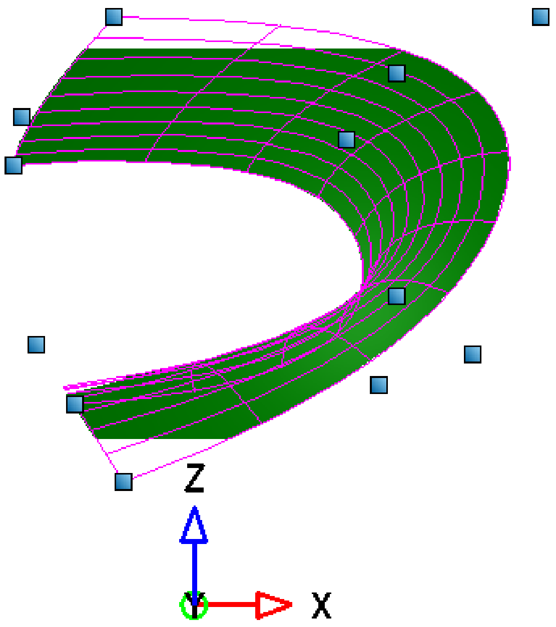


Figure 4.5: A B-spline surface constructed with data points in MicroStation V8

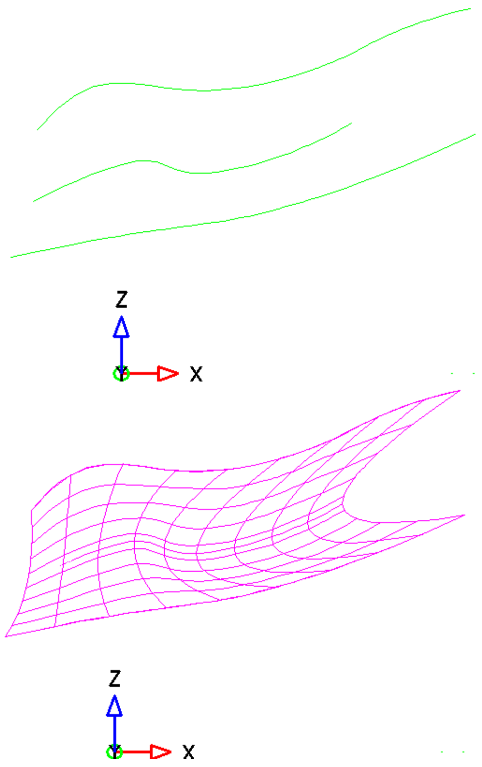


Figure 4.6: A B-spline surface constructed by cross-sections in MicroStation V8

curves/surfaces are created, all their properties (degrees, control points' positions, closure, etc.) can be edited in AutoCAD and MicroStation. AutoCAD is also able to modify control points' weights.

Chapter 5

Database Storage of Spatial Data

Database management systems (DBMSs) have extraordinary ability in managing huge amounts of data. Because of the increasing need from GIS/CAD for spatial data, more and more DBMSs have implemented modules for spatial data managing, including spatial data types, functions to store, query and analyze spatial data, etc. Such DBMSs are called **spatial DBMSs**. Several years' research on spatial data has achieved some standards, the Open GeoSpatial Consortium specifications, and some of these specifications give standards for spatial DBMSs. A general introduction of spatial DBMSs will be given in this chapter.

5.1 OGC specifications

The OGC (Open GeoSpatial Consortium) is a non-profit organization dedicated to open systems geoprocessing. OGC gives standards in the GIS field and recommends the directions for GIS researchers. One of the main tasks of OGC is to develop specifications, which follows strict procedures and policies. First, various topics are mentioned by OGC's members and discussed in the TC (Technical Committee) and/or PC (Planning Committee). Then, recommendation papers are generated from the discussion topics, and maybe adopted by TC/PC. Finally, some of the adopted recommendation paper may become specifications under **The OGC Technical Committee Policies & Procedures** [14].

The OGC creates two kinds of specifications: Abstract specifications and Implementation specifications.

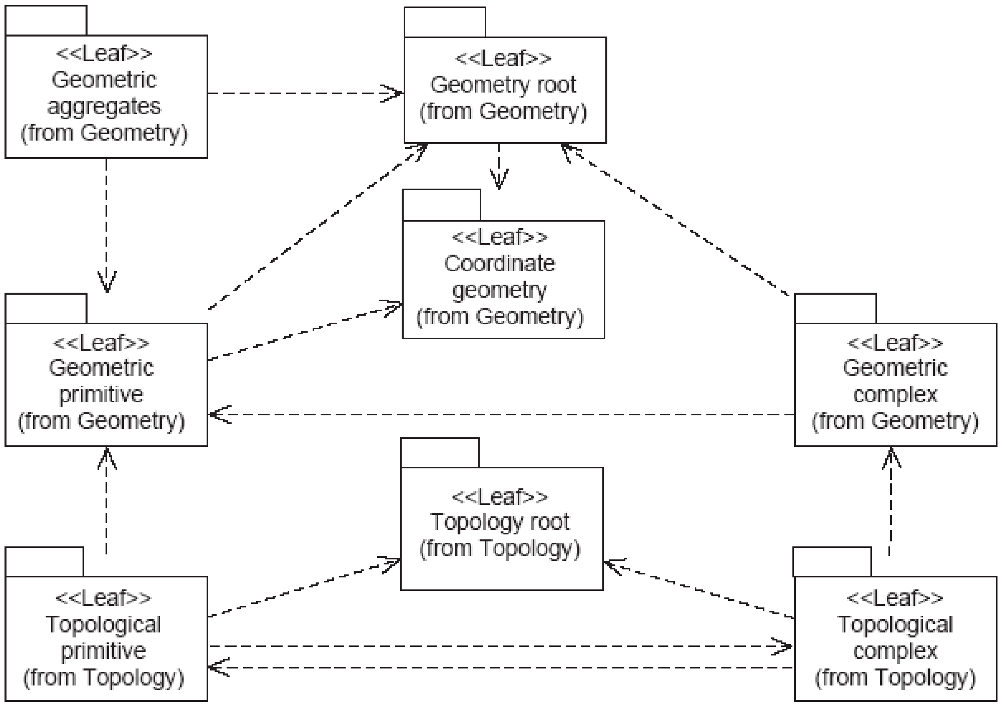
5.1.1 Abstract specifications

The purpose of **Abstract specifications**[1] is to create and document a conceptual model sufficient enough to allow for the creation of Implementation Specifications. One OGC abstract specification, the **Spatial schema**, gives a standard for geographic information by specifying geometry and topology separately as a UML (Unified Modeling Language) package dependency tree in Figure 5.1. The geometry package (Figure 5.2) has several internal packages that separate primitive geometric objects, aggregates and complexes, which have a more elaborate internal structure than simple aggregates [7]. Some representations of freeform curves and surfaces, such as Bézier curve/surface and B-spline curve/surface, have been standardized in the geometric package.

Take the '**GM_BSplineCurve**' package in Figure 5.3 as example. There are four attributes: 'degree', 'curveForm', 'knotSpec' and 'isPolynomial', and one constructor in this package.

- The '**degree**' attribute specify the algebraic degree of the basis functions;
- The '**curveForm**' attribute is used to identify particular types of curve which this spline is being used to approximate, or set to 'NULL' if no such approximation is intended;
- The '**knotSpec**' attribute gives the type of knot distribution used in defining this spline;
- The '**isPolynomial**' attribute is set to 'True' if this is a polynomial spline.

[1]This abstract specification has also been accepted by ISO and is referred as **ISO 19107**.



<<Leaf>> Geometric aggregates (from Geometry)
<<Leaf>> Geometry root (from Geometry)
<<Leaf>> Coordinate geometry (from Geometry)
<<Leaf>> Geometric primitive (from Geometry)
<<Leaf>> Geometric complex (from Geometry)
<<Leaf>> Topology root (from Topology)
<<Leaf>> Topological primitive (from Topology)
<<Leaf>> Topological complex (from Topology)

Figure 5.1: UML package dependency tree [7]

- The class constructor '**GM_BSplineCurve**' takes four parameters (degree, control points, knots, knot types) to construct a B-spline curve. [7].

Note that the weight values are stored along with the coordinates of control points as an extra dimension.

Unfortunately, NURBS has not been standardized in the latest **Spatial schema**, although it has already been included in many geometric standards, such as OpenGL, IGES, STEP and PHIGS [2]. For example in OpenGL, function '*gluNurbsCurve()*' is used to describe a NURBS curve with the following parameters [26]:

nknots Specifies the number of knots.

knot Specifies an array of knot values.

stride Specifies the offset (as a number of single-precision floating-point values) between successive curve control points.

ctlarray Specifies a pointer to an array of control points.

order Specifies the order of the NURBS curve.

5.1.2 Implementation specifications

The **Implementation specifications** are unambiguous technology platform specifications for implementation of industry-standard, software application programming interfaces [14]. One OGC implementation specification that we should pay attention is the **OpenGIS Simple Features Specification for SQL (SFS)**[16]. This specification defines standard SQL schema for simple geospatial features (geometric primitives) which are based on OGC **Abstract specifications**. Although OGC **Abstract specifications** also defines standards for complex features (topological primitives), they are still missing

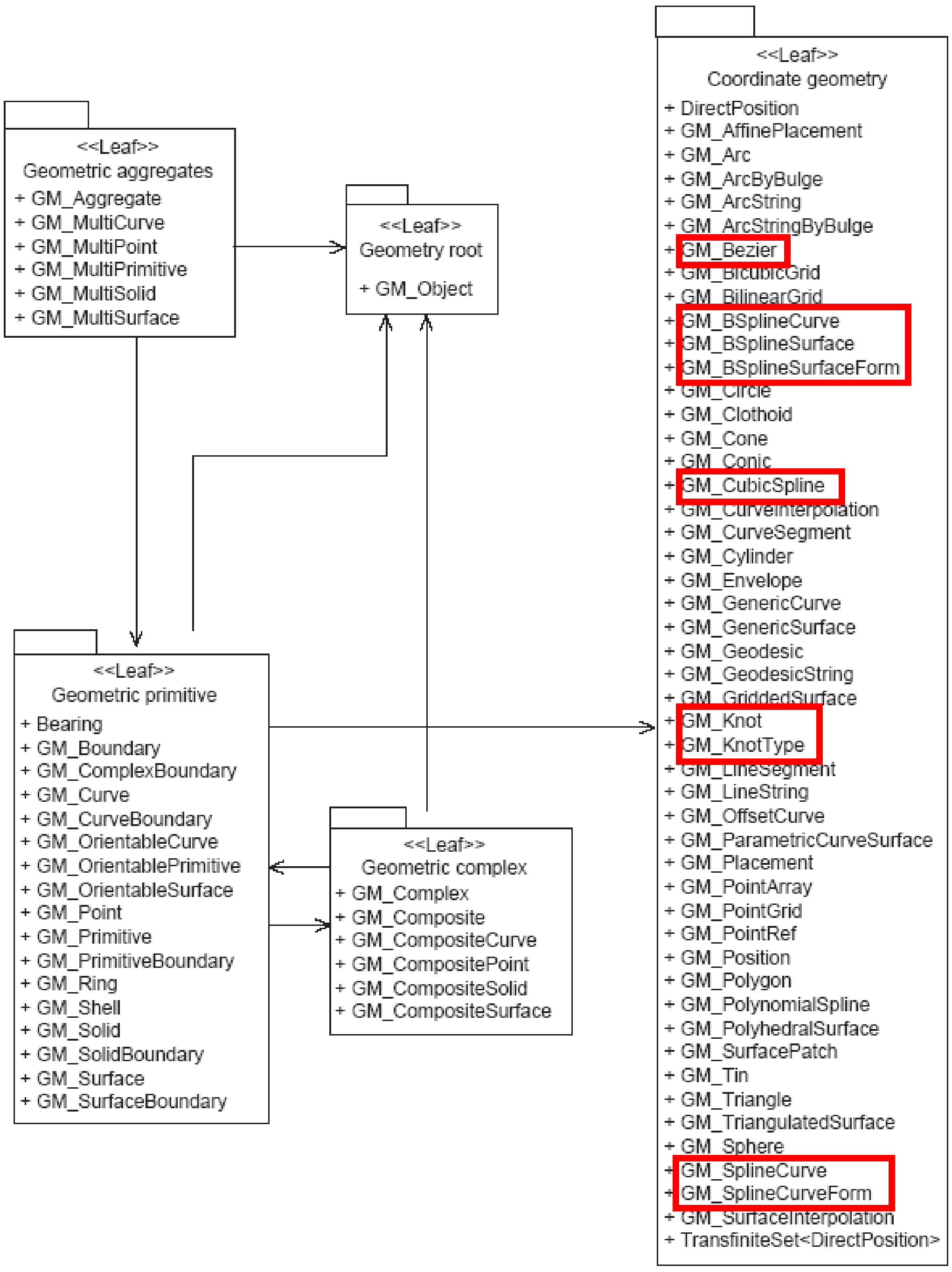
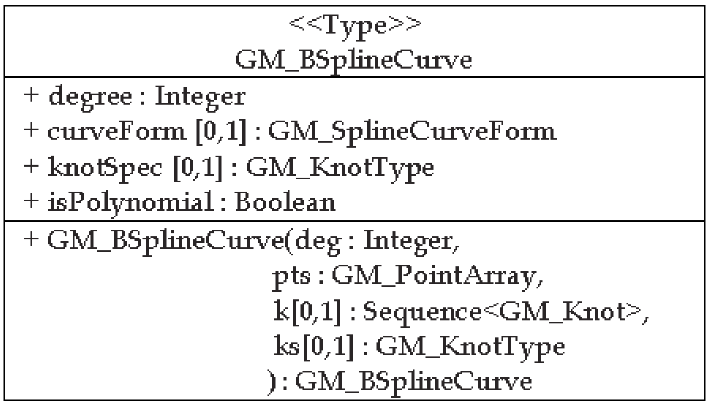


Figure 5.2: Geometric package in the **Spatial Schema** [7]. The Bézier type, B-spline type and their related types are marked with red boxes.



<<Type>>
GM_BSplineCurve
+ degree : Integer
+ curveForm [0,1] : GM_SplineCurveForm
+ knotSpec [0,1] : GM_KnotType
+ isPolynomial : Boolean
+ GM_BSplineCurve(deg : Integer,
pts : GM_PointArray,
k[0,1] : Sequence<GM_Knot>,
ks[0,1] : GM_KnotType
) : GM_BSplineCurve

Figure 5.3: GM_BSplineCurve package [7]

in SFS and other OGC **Implementation specifications**. The SQL schema in SFS is for two target SQL environments: **SQL92** and **SQL92 with Geometry Types**. The difference between **SQL92** and **SQL92 with Geometry Types** is simply as the difference between their names that geometry types exist in the latter environment while not in the first one. Both environments are recommended to standardize the following aspects:

Spatial type A spatial DBMS should support the following spatial types (Figure 5.4):

- Geometry
- Point
- Curve
- LineString, Line, LinearRing
- Polygon
- Surface
- GeometryCollection
- MultiPoint
- MultiCurve
- MultiLineString
- MultiPolygon
- MultiSurface

The difference between Multi type and non-Multi type is that the first type is a geometric collection of the latter type.

SFS also asserted the rules that define valid geometry types. For example, the assertions for polygons are:

1. Polygons are topologically closed.
2. The boundary of a polygon consists of a set of linea rings that make up its exterior and interior boundaries.
3. No two rings in the boundary cross, the rings in the boundary of a polygon may intersect at a Point but only as a tangent.
4. A polygon may not have cut lines, spikes or punctures.
5. The interior of every polygon is a connected point set.
6. The exterior of a polygon with 1 or more holes is not connected. Each hole defines a connected component of the exterior.

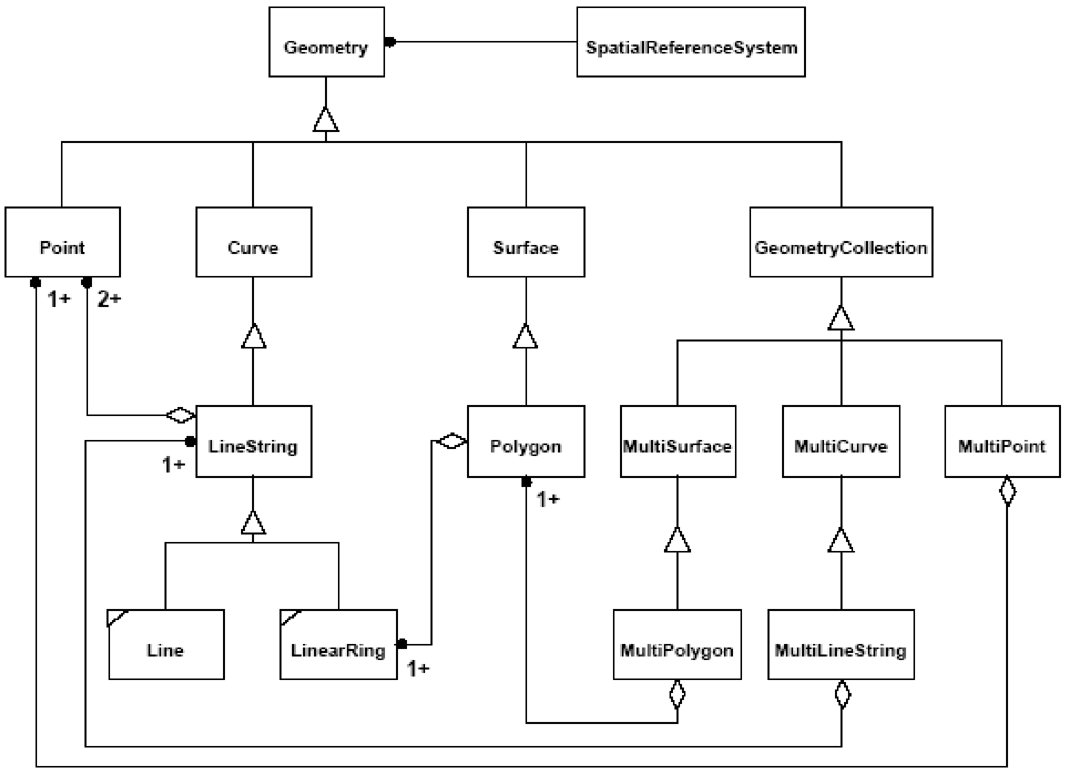


Figure 5.4: Geometry Class Hierarchy in SFS

These rules should be checked during insertion of spatial data, and invalid data should not be accepted when these rules are violated.

Exchange formats The Well-Known Text (WKT) representation and the Well-Known Binary (WKB) representation are recommended to be used to exchange (import/export) spatial data. WKT/WKB provides standard textual/binary representations for spatial reference system information. The nine geometries above, including simple or non-simple, closed or non-closed, can be represented accurately using WKT/WKB. For example:

A Polygon with one exterior ring and one interior ring represented with WKT:

```
POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))
```

A POINT(1 1) represented with WKB:

```
0101000000000000000000F03F000000000000F03F
```

This is a collection of:

- Byte order : 01
- WKB type : 01000000
- X : 000000000000F03F
- Y : 000000000000F03F

Spatial operators/functions A spatial database cannot 'manage' spatial data types without spatial operators/functions. SFS also defines a lot of spatial functions, including functions for constructing a geometry value given its WKT/WKB, functions that test spatial relationships, functions for distance relationships, functions that implements spatial operators and spatial functions on each spatial data types. SFS doesn't define any validation function. Validation of spatial data should be checked innerly before insertion.

We list the names of these spatial functions below to give readers an overview. Explanation for each function can be found in [16].

The functions for constructing a geometry value given its WKT include:

- GeomFromText(geometryTaggedText String, SRID Integer) : Geometry
- PointFromText (pointTaggedText String, SRID Integer): Point
- LineFromText(lineStringTaggedText String, SRID Integer) : LineString
- PolyFromText(polygonTaggedText String, SRID Integer): Polygon
- MPointFromText (multiPointTaggedText String, SRID Integer): MultiPoint
- MLineFromText (multiLineStringTaggedText String, SRID Integer): MultiLineString
- MPolyFromText(multiPolygonTaggedText String, SRID Integer): MultiPolygon
- GeomCollFromTxt(geometryCollectionTaggedText String, SRID Integer): GeomCollection
- BdPolyFromText(multiLineStringTaggedText String, SRID Integer): Polygon
- BdMPolyFromText(multiLineStringTaggedText String, SRID Integer): MultiPolygon

The functions for constructing a geometry value given its WKB include:

- GeomFromWKB(geometryTaggedWKB String, SRID Integer) : Geometry
- PointFromWKB(pointTaggedWKB String, SRID Integer): Point
- LineFromWKB(lineStringTaggedWKB String, SRID Integer) : LineString
- PolyFromWKB(polygonTaggedWKB String, SRID Integer): Polygon
- MPointFromWKB (multiPointTaggedWKB String, SRID Integer): MultiPoint
- MLineFromWKB (multiLineStringTaggedWKB String, SRID Integer): MultiLineString
- MPolyFromWKB(multiPolygonTaggedWKB String, SRID Integer): MultiPolygon
- GeomCollFromTxt(geometryCollectionTaggedWKB String, SRID Integer): GeomCollection
- BdPolyFromWKB(multiLineStringTaggedWKB String, SRID Integer): Polygon
- BdMPolyFromWKB(multiLineStringTaggedWKB String, SRID Integer): MultiPolygon

The functions that test spatial relationships include:

- Equals(g1 Geometry, g2 Geometry) : Integer
- Disjoint(g1 Geometry, g2 Geometry) : Integer
- Touches(g1 Geometry, g2 Geometry) : Integer
- Within(g1 Geometry, g2 Geometry) : Integer
- Overlaps(g1 Geometry, g2 Geometry) : Integer
- Crosses(g1 Geometry, g2 Geometry) : Integer
- Intersects(g1 Geometry, g2 Geometry) :Integer
- Contains(g1 Geometry, g2 Geometry) : Integer
- Relate(g1 Geometry, g2 Geometry, patternMatrix String) : Integer

The functions for distance relationships include:

- Distance(g1 Geometry, g2 Geometry) : Double Precision

The functions that implements spatial operators include:

- Intersection (g1 Geometry, g2 Geometry) : Geometry
- Difference (g1 Geometry, g2 Geometry) : Geometry
- Union (g1 Geometry, g2 Geometry) : Geometry
- SymDifference(g1 Geometry, g2 Geometry) : Geometry
- Buffer (g1 Geometry, d Double Precision) : Geometry
- ConvexHull(g1 Geometry) : Geometry

There're also spatial functions for each spatial data types. For example, the spatial functions for 'curve' are:

- StartPoint(c Curve) : Point
- EndPoint(c Curve) : Point
- IsClosed(c Curve) : Integer
- IsRing(c Curve) : Integer
- Length(c Curve) : Double Precision

Metadata The set of feature tables should be registered in the **GEOMETRY_COLUMNS metadata view**, in which metadata of each geometry column will be registered in a row. **GEOMETRY_COLUMNS metadata views** store the identity of the feature table of which it is a member, the spatial reference system ID, the type of geometry for the column, and the coordinate dimension.

Spatial Reference System A spatial DBMS should also develop its **Spatial Reference System**, which identifies the coordinate system for all geometries stored in the geometric columns, and gives meaning to the numeric coordinate values for any geometry instance stored in the columns. The columns of **Spatial Reference System** tables are the Spatial Reference System Identifier (SRID), the Spatial Reference System Authority Name (AUTH_NAME), the Authority Specific Spatial Reference System Identifier (AUTH_SRID) and the Well-known Text description of the Spatial Reference System (SRTEXT). The Spatial Reference System Identifier (SRID) constitutes a unique integer key for a Spatial Reference System within a database[16].

Several commercial DBMSs have already published their products which partially embraces the OGC's specifications. Some of these spatial DBMSs are discussed in the next section.

5.2 Spatial support in current databases

Several spatial DBMSs are available on the market now. In order to take a closer look at current spatial DBMSs, four mainstream spatial DBMSs: **Ingres [20]**, **Informix [21]**, **Postgres [23]** and **Oracle [17]**, are chosen and studied in aspects of their supported spatial types, import formats, spatial operators/functions, spatial index, metadata and **Spatial Reference System**.

5.2.1 Ingres

Ingres is a well known OpenSource DBMS developed by CA (Computer Associations). Ingres gives nice support to spatial data types, although it doesn't support OGC specification.

Spatial types The spatial data are supported by the **Ingres Spatial Object Library** in Ingres. There're nine spatial data types, all in 2D:

- point
- lseg
- line
- long line
- box

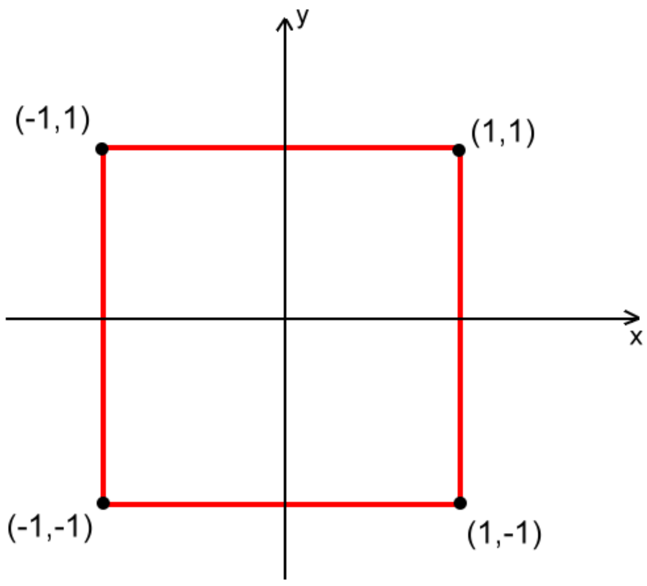


Figure 5.5: A polygon with 4 vertices

- polygon
- long polygon
- circle
- nbr

These data types actually represent four geometry types: point, line, polygon and circle. Points are defined by their x coordinates and y coordinates; lines are defined by begin point and end point; polygon type contains number of points(vertices) and an array of points' coordinates; circle type is defined by its center point and radius.

The following example shows how to insert a polygon with four vertices: (-1,-1), (-1,1), (1,1), (1,-1), as Figure 5.5:

```
CREATE TABLE example_table
(id integer,shape polygon(4)); //4 is the number of vertices

INSERT INTO example_table VALUES ( 1 ,
'((-1,-1),(-1,1),(1,1),(1,-1))',
);
```

Exchange formats WKT and WKB are not supported.

Spatial operators/functions Supported spatial operators include equality operator, binary operator(inside, intersects and overlaps) and overlap operator. Supported spatial functions include area, length, perimeter and distance.

Spatial index In Ingres we are also able to make R-tree index on spatial columns. R-Tree organizes data in a tree-shaped structure, with a **minimum bounding rectangle**(MBR) at the nodes. MBR indicate the farthest extent of the data that is connected to the subtree below. A search using an R-tree index can quickly descend the tree to find objects in the general area of interest and then perform more exact tests on the objects themselves. An R-tree index can improve performance because it eliminates the need to examine objects outside the area of interest. Without an R-tree index, a query would need to evaluate every object to find those that match the query criteria.

Metadata Not mentioned.

Spatial Reference System Not mentioned.

5.2.2 Informix

IBM's Informix developed spatial functionalities in its **Spatial DataBlade Module**, and partially implemented the OGC standard **Simple Features Specification for SQL** in this module. Accordingly,

Spatial types Six spatial types mentioned in SFS are implemented in Informix, they are

- ST_Point
- ST_MultiPoint
- ST_LineString
- ST_MultiLineString
- ST_Polygon
- ST_MultiPolygon.

ST_Point has not only x and y coordinate values, but may include z coordinate and an m value. An m(measure) coordinate is a value that conveys information about a geographic feature.

Below is an example showing how to create a spatial table and insert the polygon in Figure 5.5 using IBM Informix load format in Informix:

```
CREATE TABLE example_table (
id integer,
shape ST_Polygon); //spatial column

INSERT INTO example_table VALUES (
1,
'5 polygon (-1 -1, -1 1, 1 1, 1 -1)'
);
```

Exchange formats Exchange formats of spatial types can be IBM Informix load format, WKT, WKB and shape representation, a widely used industry standard defined by ESRI(Environmental Systems Research Institute). The IBM Informix load format is like: 'SRID, TYPE(coordinate, ..., coordinate)', where SRID is a integer which references its **Spatial Reference System**; TYPE is the geometric name such as point, polygon, etc.; coordinates of points/vertices are specified between () one bye one, separated by comma. ESRI's shape representation is beyond the scope of this report, so it will not be explained here. A full description can be found from ESRI website at http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf.

Spatial operators/functions There're about 100 2-dimensional spatial operators and functions in **Informix Spatial DataBlade Module**. All SFS-compliant functions have the prefix ST_, while functions that extend SFS have the prefix SE_. The tasks for these functions can be:

- Generate a geometry from a well-known text representation. For example, the function ST_GeomFromText() takes a well-known text representation and returns an ST_Geometry.
- Generate a geometry from a well-known binary representation. For example, the function ST_GeomFromWKB() takes a well-known binary representation and returns an ST_Geometry.
- Generate a geometry from an ESRI shape representation. For example, the function SE_GeomFromShape() takes an ESRI shape representation and returns an ST_Geometry.

- Convert a geometry to an external format. For example, the function ST_AsText() returns the well-known text representation of a geometry object.
- Manipulate the ST_Point data type. For example, the function ST_X() returns the X coordinate of a point.
- Manipulate the ST_LineString and ST_MultiLineString data types. For example, the function ST_PointN() returns the nth point.
- Manipulate the ST_Polygon and ST_MultiPolygon data types. For example, the function ST_Area() calculates the area of polygons.
- Obtain parameters of a geometry. For example, the function ST_Distance() returns the shortest distance to another geometry.
- Determine the properties of a geometry. For example, the function ST_IsValid() determines whether a geometry is topologically valid.
- Determine if a certain relationship exists between two geometries. For example, the function ST_Intersects() determines whether two geometries intersect.
- Compare geometries and return a new geometry based on how the geometries are related. For example, the function ST_Intersection() computes the intersection of two geometries.
- Generate a new geometry from an existing geometry. For example, the function ST_Envelope() generates the bounding box of a geometry.
- Manage collections. For example, the function ST_NumGeometries() returns the number of geometries in the collection.
- Nearest-neighbor queries. For example, the function SE_Nearest() retrieves nearby geometries in increasing distance order.
- Assist in managing spatial reference systems. For example, the function ST_SRID() returns a geometrys spatial reference ID.
- Administration. For example, the function SE_MetadataInit() reinitializes the spatial reference system memory cache.

The Informix Spatial DataBlade module implemented a spatial validation function: ST_IsValid(). Actually spatial data are validated before accepted, so ST_IsValid() always returns TRUE. This function may be used to validate spatial data supplied by other implementations of the OGC specifications.

The following example returns any geometry that overlaps the window defined with ST_PolyFromText:

```
SELECT name from example_table
WHERE SE_EnvelopesIntersect(shape,
ST_PolyFromText('polygon((20000 20000,60000 20000,60000 60000,20000 60000,20000 20000))',
5));
```

Spatial index Informix supports R-tree index. To create an R-tree index for our example, use the following statement:

```
CREATE INDEX shape_idx
on example_table (shape ST_Geometry_ops)
USING RTREE;
```

Metadata After creating a spatial table or inserting a spatial column to a normal table, we need to register their metadata in the **geometry_columns** table, which has a rather similar structure with the **GEOMETRY_COLUMNS metadata view** in SFS. Required information includes user name, spatial column name, column type, srid, etc.

Spatial Reference System Before creating a spatial column, setup of **Spatial Reference System** is required in the **Spatial_Reference Table**. The columns of this table include srid, description of the spatial reference system, the name of the standard or standards body cited for the reference system, scale factors, etc.

5.2.3 Postgres

Postgres is a popular DBMS developed by a team of developers who subscribe to the PostgreSQL development mailing list. PostGIS adds support for geographic objects to the Postgres as an option. PostGIS' support of spatial data can be revealed from the following aspects:

Spatial types PostGIS supports seven spatial types in SFS, they are:

- POINT
- MULTIPOINT
- LINESTRING
- MULTILINESTRING
- POLYGON
- MULTIPOLYGON
- GEOMETRYCOLLECTION

All of these types can be 2-, 3- or 4-dimensional.

Below is an example showing how to create a spatial table and insert the polygon in Figure 5.5 using the Well Known Texture representation:

```
CREATE TABLE example_table (id int4);

SELECT AddGeometryColumn('example_table', 'shape', 128, 'POLYGON', 2 ); //square is
column name, 128 is SRID, 2 is dimension

INSERT INTO example_table(id, shape) VALUES
(1, GeometryFromText('POLYGON(-1 -1, -1 1, 1 1, 1 -1)', 128));
```

Notice that in PostGIS, a spatial column can only be added to existing tables using 'AddGeometryColumn' function instead of the common used SQL statement: 'alter table'.

Exchange formats WKT and WKB are used to exchange spatial data in PostGIS.

Spatial operators/functions PostGIS also has both **OGC compliant spatial functions** and their own **extension functions**. Below is a list of some OGC compliant spatial functions, they can be grouped as Management Functions, Geometry Relationship Functions, Geometry Processing Functions, Geometry Accessors and Geometry Constructors.

Management Functions:

- AddGeometryColumn(varchar, varchar, varchar, integer, varchar, integer)
- DropGeometryColumn(varchar, varchar, varchar)
- SetSRID(geometry)

Geometry Relationship Functions:

- Distance(geometry,geometry)
- Equals(geometry,geometry)
- Disjoint(geometry,geometry)
- Intersects(geometry,geometry)
- Touches(geometry,geometry)
- Crosses(geometry,geometry)
- Within(geometry,geometry)
- Overlaps(geometry,geometry)

- Contains(geometry,geometry)
- Intersects(geometry,geometry)
- Relate(geometry,geometry, intersectionPatternMatrix)
- Relate(geometry,geometry)

Geometry Processing Functions:

- Centroid(geometry)
- Area(geometry)
- Length(geometry)
- PointOnSurface(geometry)
- Boundary(geometry)
- Buffer(geometry,double,[integer])
- ConvexHull(geometry)
- Intersection(geometry,geometry)
- SymDifference(geometry,geometry)
- Difference(geometry,geometry)
- GeomUnion(geometry,geometry)
- GeomUnion(geometry set)
- MemGeomUnion(geometry set)

Geometry Accessors Functions:

- AsText(geometry)
- AsBinary(geometry)
- SRID(geometry)
- Dimension(geometry)
- Envelope(geometry)
- IsEmpty(geometry)
- IsSimple(geometry)
- IsClosed(geometry)
- IsRing(geometry)
- NumGeometries(geometry)
- GeometryN(geometry,int)
- NumPoints(geometry)
- PointN(geometry,integer)
- ExteriorRing(geometry)
- NumInteriorRings(geometry)
- InteriorRingN(geometry,integer)
- EndPoint(geometry)
- StartPoint(geometry)
- GeometryType(geometry)
- X(geometry)
- Y(geometry)
- Z(geometry)

There're several Geometry Constructors Functions with the form AFromB(), where the characters in A can be 'Geometry', 'Point', 'Line', 'Polygon', 'Linestring', 'MPoint', 'MLine', 'MPoly', and 'GeomColl'; characters in B can be 'Text' or 'WKB'.

PostGIS implemented a spatial validation function: isvalid(geometry), as an extension functions. This function returns true if the geometry is valid.

The following example calculate the area of the polygon in Figure 5.5.

```
SELECT area(shape) FROM example_table WHERE id = 1;
```

Spatial index PostGIS supports two kinds of spatial index methods: R-tree index and GiST (Generalized Search Trees) index. The GiST index breaks up data into "things to one side", "things which overlap", "things which are inside" and can be used on a wide range of data-types, including GIS data [23].

Metadata Metadata of spatial columns are stored in the **GEOMETRY_COLUMNS** table. These metadata should contain name of the spatial table, name of the spatial column, spatial dimension, SRID and spatial type.

Spatial Reference System Spatial reference systems are defined in the **SPATIAL_REF_SYS** table. Columns of this table fully follow the OGC specification 'Simple Features Specification for SQL'.

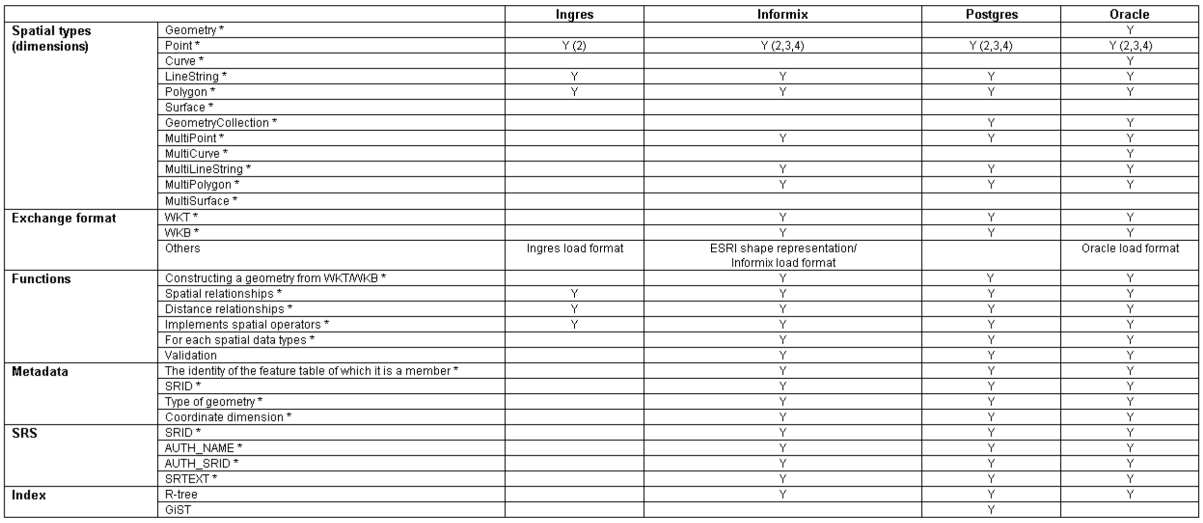
5.2.4 Oracle

Oracle series began to support spatial data in its option **Oracle Spatial** since Oracle 8i. Oracle Spatial is compliant with **Simple Features Specification for SQL**. The supported spatial types are: point, line, polygon, and their multi-version. The types are stored with information such as data type, dimension (2, 3, or 4), coordinates, interior or exterior, holes, etc. System table **Spatial_Reference_System** stores the information of spatial reference system. System view **USER_SDO_GEOM_METADATA** stores the meta data. Oracle Spatial will be further studied in the next chapter.

5.2.5 A summary

Spatial data types are supported and handled well in the four DBMSs above. Three of the four are compliant with OGC specification 'Simple Feature Specification for SQL', except Ingres. There're point, line and polygon in all of them, while curve, surface and more complex geometries are still absent. Spatial data types can be 2-, 3- or 4-dimensional, but operations and functions of 3- or 4-dimensional spatial data are actually done by first projecting these data into 2-dimension. WKT and WKB formats are used in most of the five, some of them also developed their own formats and utilities, which are quite helpful to copy/retrieve spatial data into/from spatial databases.

The spatial functionalities of the four spatial DBMSs are summarized in Figure 5.6.



		Ingres	Informix	Postgres	Oracle
Spatial types (dimensions)	Geometry *				Y
	Point *	Y (2)	Y (2,3,4)	Y (2,3,4)	Y (2,3,4)
	Curve *				Y
	LineString *	Y	Y	Y	Y
	Polygon *	Y	Y	Y	Y
	Surface *				
	GeometryCollection *			Y	Y
	MultiPoint *		Y	Y	Y
	MultiCurve *				Y
	MultiLineString *		Y	Y	Y
	MultiPolygon *		Y	Y	Y
	MultiSurface *				
Exchange format	WKT *		Y	Y	Y
	WKB *		Y	Y	Y
	Others	Ingres load format	ESRI shape representation/ Informix load format		Oracle load format
Functions	Constructing a geometry from WKT/WKB *		Y	Y	Y
	Spatial relationships *	Y	Y	Y	Y
	Distance relationships *	Y	Y	Y	Y
	Implements spatial operators *	Y	Y	Y	Y
	For each spatial data types *		Y	Y	Y
	Validation		Y	Y	Y
Metadata	The identity of the feature table of which it is a member *		Y	Y	Y
	SRID *		Y	Y	Y
	Type of geometry *		Y	Y	Y
	Coordinate dimension *		Y	Y	Y
SRS	SRID *		Y	Y	Y
	AUTH_NAME *		Y	Y	Y
	AUTH_SRID *		Y	Y	Y
	SRTEXT *		Y	Y	Y
Index	R-tree		Y	Y	Y
	GiST			Y	

Figure 5.6: Spatial functionalities of Ingres, Informix, Postgres and Oracle, * represents SFS features

Chapter 6

Oracle Spatial

Partially compliant with OGC specification, Oracle Spatial 9i is one of the most powerful spatial DBMSs on market. This chapter continues the exploration of current spatial DBMSs in the last chapter by zooming into Oracle spatial. Six important aspects of a spatial DBMSs: supported spatial type, their exchange formats, spatial operators and functions, metadata, **Spatial Reference System**, and spatial index, will be detailedly studied for Oracle Spatial 9i in this chapter.

6.1 Spatial types

Before explaining the spatial types supported by Oracle Spatial, an Entity Relationship diagram in Figure 6.1 would be helpful to understand the object-relational model in Oracle Spatial 9i.

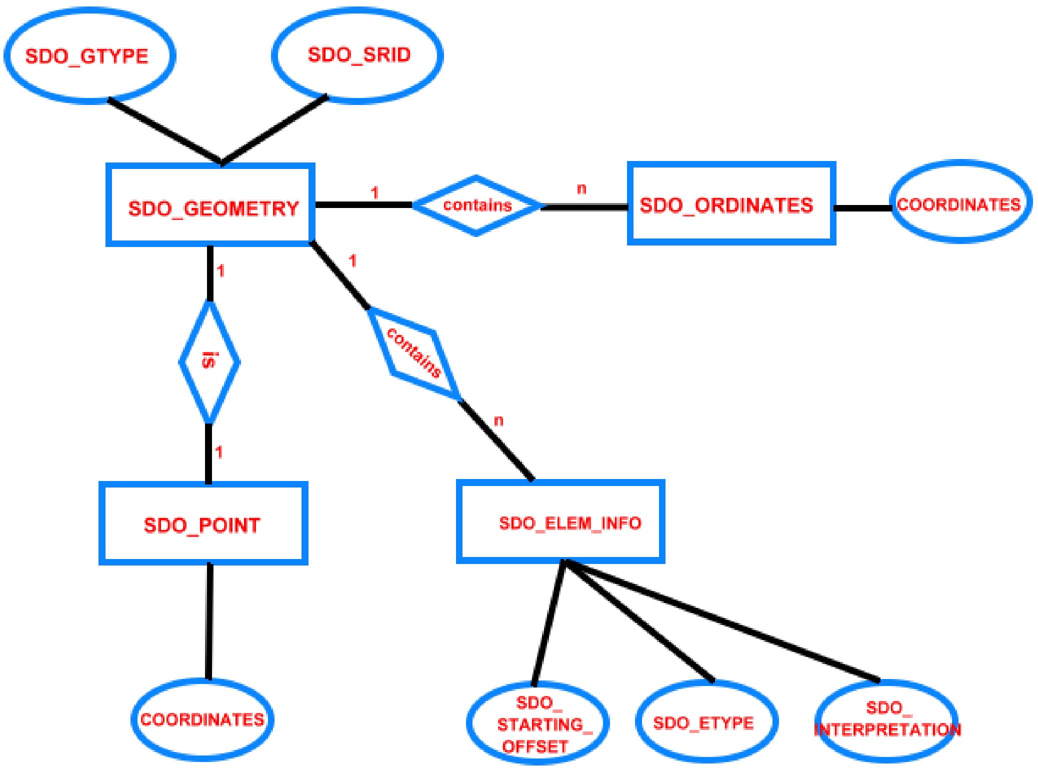


Figure 6.1: ER diagram for Oracle Spatial

Unlike other spatial DBMSs in which different spatial types represent different geometries, there's only one spatial type: SDO_GEOMETRY in Oracle Spatial. Several kinds of geometries can be represented by setting attributes of SDO_GEOMETRY. Oracle Spatial defines the object type SDO_GEOMETRY as[17]:

```
CREATE TYPE sdo_geometry AS OBJECT (
SDO_GTYPE NUMBER,
SDO_SRID NUMBER,
SDO_POINT SDO_POINT_TYPE,
SDO_ELEM_INFO SDO_ELEM_INFO_ARRAY,
SDO_ORDINATES SDO_ORDINATE_ARRAY);
```

SDO_GTYPE The SDO_GTYPE(geometry type) value is 4 digits in the format dltt. d is the number of dimension (2, 3, or 4); l specifies which dimension (3 or 4) contains the measure value; tt represents geometry types, for example, dl01 represents point, dl02 represents line/curve, dl03 represents polygon, etc. Supported geometry types are:

- UNKNOWN_GEOMETRY
- POINT
- LINE/CURVE
- POLYGON(with/without holes)
- COLLECTION
- MULTIPOINT
- MULTILINE/MULTICURVE
- MULTIPOLYGON.

SDO_SRID This attribute is used to relate a spatial reference system to this geometry. "null" values means this geometry doesn't related to any reference system.

SDO_POINT When SDO_POINT value is not null, this geometry is just a single point, otherwise it is one of other geometries. Using SDO_POINT is convenient when there're only point geometries in a layer.

SDO_ELEM_INFO This attribute indicates the format of coordinates in SDO_ORDINATES by repeating the following attributes:

SDO_STARTING_OFFSET This attribute indicate the starting offset of current element's first coordinate in SDO_ORDINATES;

SDO_ETYPE Type of current element, can be simple elements or compound elements. Especially, geometries which are unsupported by Oracle Spatial can be represented by setting SDO_ETYPE to zero. Geometries with type 0 elements must contain at least one nonzero element, which should be an approximation of the unsupported geometry.

SDO_INTERPRETATION Interpretation for SDO_ETYPE. Especially, when this is an zero type element, the SDO_INTERPRETATION value for the type zero element can be any numeric value, and applications are responsible for determining the validity and significance of the value[17].

SDO_ORDINATES Arrays of coordinates are stored here with the format specified in SDO_ELEM_INFO.

The SQL statements below create a spatial table with a spatial column, then insert the polygon in Figure 5.5 into this table.

```
CREATE TABLE example_table(
id NUMBER,
shape SDO_GEOMETRY);
```

```
INSERT INTO example_table VALUES(
1,
SDO_GEOMETRY(
2003, --two-dimensional polygon
NULL,
NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),
  --1003 indicates simple polygon, 1 indicates vertices are connected by straight line
                                                                               segments
SDO_ORDINATE_ARRAY(-1,-1, -1,1, 1,1, 1,-1)
                                                    --coordinates for four vertices
));
```

Notice that the SQL statement above is not the only possibility to represent this polygon. We can also set SDO_ETYPE to 1003 and SDO_INTERPRETATION to 3, which indicates a rectangle. In this case, only coordinates of top-right and bottom-left vertices should be specified in SDO_ORDINATE_ARRAY.

6.2 Exchange format

Because Oracle Spatial represents different spatial types by setting attribute of SDO_GEOMETRY, WKT and WKB cannot be used as the exchange formats of spatial data. However, if one want to define a **Spatial Reference System**, WKT should be used to describe the user-defined SRS.

6.3 Spatial operators and functions

Oracle Spatial support a large number of operators and functions for spatial data types. Spatial operators require the spatial columns to be indexed first, and they must be used in the 'where' clause of queries. Spatial functions differ from operations in that they don't require the indexing. Oracle Spatial's spatial functions can be grouped into: Relationship, Validation, Single-object operations and Two-object operations. Both of operators and functions are included in the **Spatial PL/SQL application programming interface** (API).

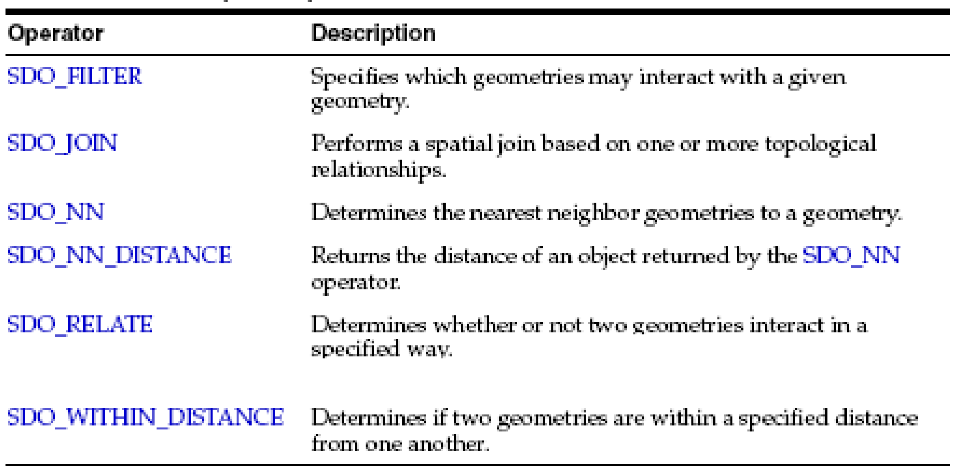
Figure 6.2 lists the main operators.

Figure 6.3 lists operators, provided for convenience, that perform an SDO_RELATE operation of a specific mask type.

Figure 6.4 lists the spatial functions in Oracle Spatial. The can be grouped into:

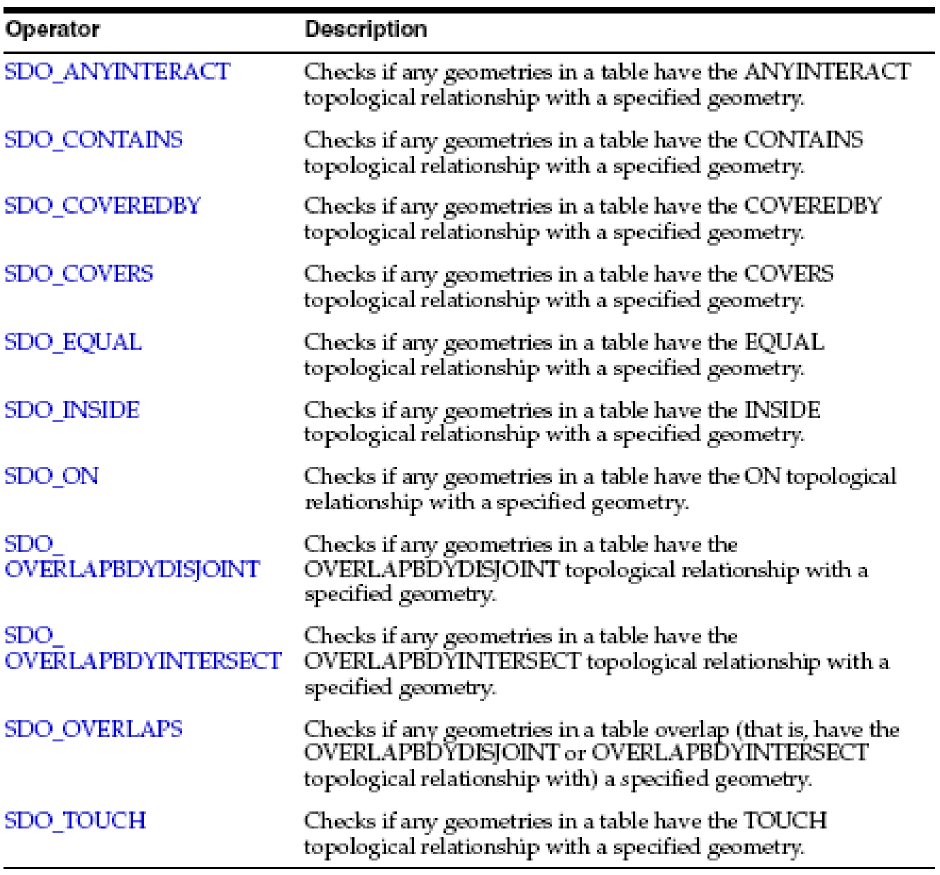
- Relationship (True/False) between two objects: RELATE, WITHIN_DISTANCE
- Validation: VALIDATE_GEOMETRY_WITH_CONTEXT, VALIDATE_LAYER_ WITH_CONTEXT
- Single-object operations: SDO_ARC_DENSIFY, SDO_AREA, SDO_BUFFER, SDO_CENTROID, SDO_CONVEXHULL, SDO_LENGTH, SDO_MAX_MBR_ORDINATE, SDO_MIN_MBR_ORDINATE, SDO_MBR, SDO_POINTONSURFACE
- Two-object operations: SDO_DISTANCE, SDO_DIFFERENCE, SDO_INTERSECTION, SDO_UNION, SDO_XOR

The two validation functions: VALIDATE_GEOMETRY_WITH_CONTEXT and VALIDATE_LAYER_ WITH_CONTEXT, are used to check type consistency and geometry consistency. For type consistency, Oracle Spatial checks whether: 1. The SDO_GTYPE is valid. 2. The SDO_ETYPE values are consistent with the SDO_GTYPE value. 3. The SDO_ELEM_INFO_ARRAY has valid triplet values. For geometry consistency, the checking follows several rules such as: polygons are not self-crossing; no two vertices on a line or polygon are the same; polygons are oriented correctly (Exterior ring boundaries must be



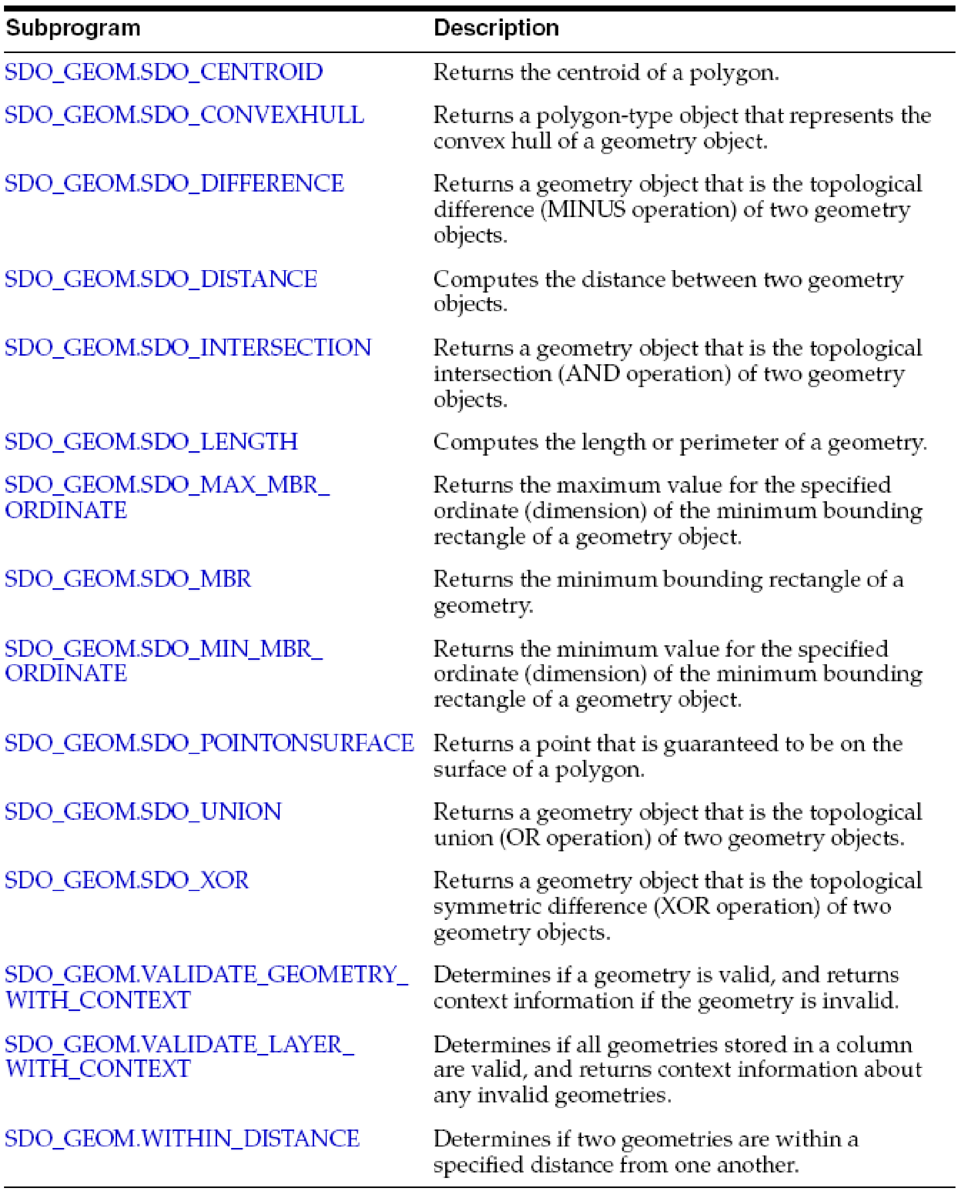
Operator	Description
SDO_FILTER	Specifies which geometries may interact with a given geometry.
SDO_JOIN	Performs a spatial join based on one or more topological relationships.
SDO_NN	Determines the nearest neighbor geometries to a geometry.
SDO_NN_DISTANCE	Returns the distance of an object returned by the SDO_NN operator.
SDO_RELATE	Determines whether or not two geometries interact in a specified way.
SDO_WITHIN_DISTANCE	Determines if two geometries are within a specified distance from one another.

Figure 6.2: Main Spatial Operator of Oracle Spatial



Operator	Description
SDO_ANYINTERACT	Checks if any geometries in a table have the ANYINTERACT topological relationship with a specified geometry.
SDO_CONTAINS	Checks if any geometries in a table have the CONTAINS topological relationship with a specified geometry.
SDO_COVEREDBY	Checks if any geometries in a table have the COVEREDBY topological relationship with a specified geometry.
SDO_COVERS	Checks if any geometries in a table have the COVERS topological relationship with a specified geometry.
SDO_EQUAL	Checks if any geometries in a table have the EQUAL topological relationship with a specified geometry.
SDO_INSIDE	Checks if any geometries in a table have the INSIDE topological relationship with a specified geometry.
SDO_ON	Checks if any geometries in a table have the ON topological relationship with a specified geometry.
SDO_ OVERLAPBDYDISJOINT	Checks if any geometries in a table have the OVERLAPBDYDISJOINT topological relationship with a specified geometry.
SDO_ OVERLAPBDYINTERSECT	Checks if any geometries in a table have the OVERLAPBDYINTERSECT topological relationship with a specified geometry.
SDO_OVERLAPS	Checks if any geometries in a table overlap (that is, have the OVERLAPBDYDISJOINT or OVERLAPBDYINTERSECT topological relationship with) a specified geometry.
SDO_TOUCH	Checks if any geometries in a table have the TOUCH topological relationship with a specified geometry.

Figure 6.3: Convenience Operators for SDO_RELATE Operations



Subprogram	Description
SDO_GEOM.SDO_CENTROID	Returns the centroid of a polygon.
SDO_GEOM.SDO_CONVEXHULL	Returns a polygon-type object that represents the convex hull of a geometry object.
SDO_GEOM.SDO_DIFFERENCE	Returns a geometry object that is the topological difference (MINUS operation) of two geometry objects.
SDO_GEOM.SDO_DISTANCE	Computes the distance between two geometry objects.
SDO_GEOM.SDO_INTERSECTION	Returns a geometry object that is the topological intersection (AND operation) of two geometry objects.
SDO_GEOM.SDO_LENGTH	Computes the length or perimeter of a geometry.
SDO_GEOM.SDO_MAX_MBR_ ORDINATE	Returns the maximum value for the specified ordinate (dimension) of the minimum bounding rectangle of a geometry object.
SDO_GEOM.SDO_MBR	Returns the minimum bounding rectangle of a geometry.
SDO_GEOM.SDO_MIN_MBR_ ORDINATE	Returns the minimum value for the specified ordinate (dimension) of the minimum bounding rectangle of a geometry object.
SDO_GEOM.SDO_POINTONSURFACE	Returns a point that is guaranteed to be on the surface of a polygon.
SDO_GEOM.SDO_UNION	Returns a geometry object that is the topological union (OR operation) of two geometry objects.
SDO_GEOM.SDO_XOR	Returns a geometry object that is the topological symmetric difference (XOR operation) of two geometry objects.
SDO_GEOM.VALIDATE_GEOMETRY_ WITH_CONTEXT	Determines if a geometry is valid, and returns context information if the geometry is invalid.
SDO_GEOM.VALIDATE_LAYER_ WITH_CONTEXT	Determines if all geometries stored in a column are valid, and returns context information about any invalid geometries.
SDO_GEOM.WITHIN_DISTANCE	Determines if two geometries are within a specified distance from one another.

Figure 6.4: Spatial Functions

oriented counterclockwise, and interior ring boundaries must be oriented clockwise), etc. The difference between VALIDATE_GEOMETRY_WITH_CONTEXT and VALIDATE_LAYER_ WITH_CONTEXT is that the latter one validates all geometries stored in a column are valid, instead of a single geometry.

Although Oracle Spatial enables us to define new geometry types, all these operators and functions cannot be applied to these user-defined types because Oracle Spatial doesn't recognize them. In order to make the spatial database fully supports 3D freeform curves and surfaces, we need to create the operators and functions for user-defined types by writing PL/SQL or JAVA codes. PL/SQL (Procedural Language extensions to SQL) is a procedural language which allows us to combine SQL statements with standard procedural programming [19]. Implementation of such operators and functions will be finished later in the practical part of my MSc thesis research.

6.4 Metadata

Oracle Spatial requires spatial users to update meta data in a view called USER_SDO_GEOM_METADATA after creating spatial columns. The required meta information are:

TABLE_NAME The name of the table with spatial columns.

COLUMNE_NAME The name of the spatial column.

DIMINFO DIMINFO registers the dimension name, their value range and tolerance value.

SRID Associated coordinate reference system, set to 0 if none associated.

The following SQL statement insert the metadata of example_table:

```
INSERT INTO USER_SDO_GEOM_METADATA
VALUES(
'example_table',
'shape',
SDO_DIM_ARRAY( --10X10 grid
SDO_DIM_ELEMENT('X', 0, 10, 0.005), --X axis
SDO_DIM_ELEMENT('Y', 0, 10, 0.005), --Y axis
),
NULL --SRID
);
```

6.5 Spatial Reference System

Information of the **Spatial Reference System** is stored in the table MDSYS.CS_SRS. The structure of MDSYS.CS_SRS is shown in Figure 6.5.

6.6 Spatial index

Index can improve the efficiency of database query. Index of spatial data in Oracle Spatial are stored in two metadata views: xxx_SDO_INDEX_INFO and xxx_SDO_INDEX_METADATA, where xxx can be USER or ALL[17]. Type 0 elements are not indexed by Oracle Spatial.

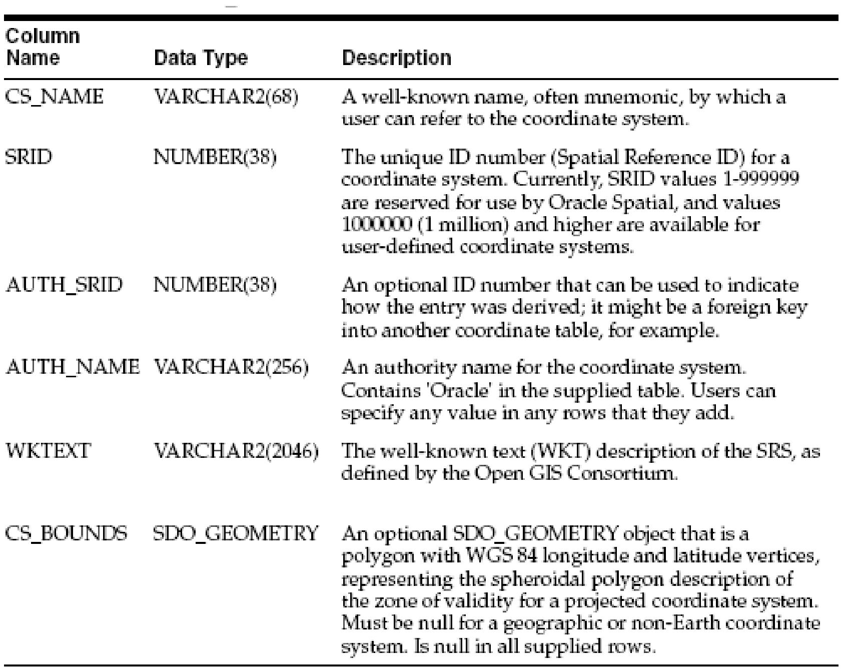
The SQL for creating a spatial index is:

```
create index index_name on spatial_table(spatial_column) INDEXTYPE is MDSYS.SPATIAL_INDEX;
```

The emphasized attributes in lowercase like *attribute* should be specified.

By default, a **R-Tree** index is created. Quadtree can be used besides R-Tree, although discouraged.

The following SQL statement creates an R-Tress index on the spatial column in Chapter 6.1:



Column Name	Data Type	Description
CS_NAME	VARCHAR2(68)	A well-known name, often mnemonic, by which a user can refer to the coordinate system.
SRID	NUMBER(38)	The unique ID number (Spatial Reference ID) for a coordinate system. Currently, SRID values 1-999999 are reserved for use by Oracle Spatial, and values 1000000 (1 million) and higher are available for user-defined coordinate systems.
AUTH_SRID	NUMBER(38)	An optional ID number that can be used to indicate how the entry was derived; it might be a foreign key into another coordinate table, for example.
AUTH_NAME	VARCHAR2(256)	An authority name for the coordinate system. Contains 'Oracle' in the supplied table. Users can specify any value in any rows that they add.
WKTEXT	VARCHAR2(2046)	The well-known text (WKT) description of the SRS, as defined by the Open GIS Consortium.
CS_BOUNDS	SDO_GEOMETRY	An optional SDO_GEOMETRY object that is a polygon with WGS 84 longitude and latitude vertices, representing the spheroidal polygon description of the zone of validity for a projected coordinate system. Must be null for a geographic or non-Earth coordinate system. Is null in all supplied rows.

Figure 6.5: MDSYS.CS_SRS Table

```
CREATE INDEX example_spatial_idx
ON example_table(shape)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

If spatial columns are spatial indexed, we can use spatial query to return topological intersection, area, distance, validity, etc. The spatial query in Oracle Spatial in done in two levels: the first level is called **primary filter**, which compares geometry approximations and return superset of accurate result; the second level is called **secondary filter**, where the result from primary filter is further compared with accurate computation to make exact result. Suppose there's another geometry with name attribute 'polygon_b'. The following example shows how to get the topological intersection of 'polygon_a' and 'polygon_b':

```
SELECT SDO_GEOM.SDO_INTERSECTION(a.shape, b.shape, 0.005)
                                                  --0.005 is the tolerance value
FROM example_table a, example_table b
WHERE a.id= 1 AND b.id= 2;
```

Chapter 7

Conclusions and Recommendations

This report presented my research work and equipped me with a solid background for the practical part of my MSc thesis work. Based on the study in the previous chapters, conclusions on representing freeform curves/surfaces and managing spatial data in spatial DBMSs are given, then considerations for my MSc final project and recommendations for further research are indicated.

My research work shows that:

- NURBS is excellent and more competitive in representing freeform curves and surfaces than Bézier and B-Spline. This is largely due to its nice mathematical and geometrical properties:
  - NURBS can represent nearly all kinds of shapes, including **conic sections**, which are very important for CAD/GIS applications.
  - The **local modification** property make shape design using NURBS very convenient.
  - NURBS curves and surfaces are both **affine invariant** and **projective invariant**, which are quite important for transformations in CAD/GIS applications.

  Therefore NURBS will be used in the practical part of my MSc thesis, to implement the freeform spatial data types in a Geo-DBMS.

- OGC specifications should consider involving the NURBS types. Now there're several freeform spatial data types, like Bézier and B-Spline, in OGC specifications, but NURBS is still not mentioned.

- GIS applications should be able to support NURBS types. GIS aims at representing and analyzing the real world, where freeform curves and surfaces are very common. However, currently only simple geometries are supported in GIS applications. NURBS curves and surfaces are very popular in CAD applications and have been recognized as very important tools for geometric design [2]. As the rapid develop of GIS research, NURBS types are also necessary for GIS applications to model the real world.

- Spatial DBMS is playing a more and more important role in GIS field. In order to make GIS fully support freeform shapes, freeform spatial data types should also be implemented in spatial DBMSs. Oracle Spatial is an appropriate spatial DBMS to implement the freeform spatial data types, because (a) it has powerful support to spatial data types; (b) it enables user-defined spatial data types.

From this research assignment, my MSc final project will try to:

- Choose a mathematical model from Bézier/B-spline/NURBS to represent the freeform curves/surfaces, find what are the criteria for valid freeform curves/surfaces, and implement freeform curve/surface spatial types in a spatial DBMS based on the mathematical model.

- Determine whether OGC specifications are going to be considered or not.

- Determine which spatial functions and operators are needed for the freeform types, then implement these spatial functions and operators.

- Implement validation functions for the freeform spatial types.
- Define 3D objects by combining freeform surfaces using the freeform spatial types.
- Implement 3D index for the freeform spatial types.
- Choose a CAD/GIS application which is able to exchange data with spatial DBMSs and visualize freeform shapes.
- Exchange freeform types with the CAD/GIS application without losing consistency.

Further research on freeform curves/surface in spatial DBMSs and CAD/GIS fields can consider the following aspects:

- A complete set of spatial functions and operators for freeform curves/surfaces types.
- Conversion tools between freeform geometry types and flat geometry types.
- Visualization techniques based on precise geometry instead of polygonal approximation.

Bibliography

[1] C.A. Arens, *Modelling 3D spatial objects in a Geo-DBMS using a 3D primitive*, MSc thesis, 2003

[2] Les Piegl and Wayne Tiller, *The NURBS book 2nd edition*, Springer, 1997

[3] Carl de Boor, *A practical guide to splines*, Springer, 2001

[4] Harmut Prautsch, Wolfgang Boehm and Marco Paluszny, *Bézier and B-spline techniques*, Springer, 2002

[5] Rogers David F, *An introduction to NURBS: with historical perspective*, Morgan Kaufmann, 2001

[6] Alberto Paoluzzi, *Geometric programming for computer-aided aesign*, Wiley, 2003

[7] ISO, *ISO standard 19107-Geographic information*, Internet Organization for Standardization, 2002

[8] Les Piegl and Wayne Tiller, *Computing offsets of NURBS curves and surfaces*, Computer-Aided Design Vol.31 (p147-156), 1999

[9] Les Piegl, *On NURBS: a survey*, IEEE Computer Graphics and Applications, Vol.11, No.1 (p55-71), 1991

[10] Elaine Cohen, Tom Lyche and Larry L.Schumake, *Algorithms for degree-raising of splines*, ACM TOG Vol4 No.3, 1985

[11] Bronsvoort W.F., Jansen F.W. and Post F.H., *Geometric modelling lecture reader*, 2003-2004

[12] Peter van Oosterom, Jantien Stoter, Wilko Quak and Sisi Zlantanova, *The balance between geometry and topology*, 10th International Symposium on Spatial Data Handling, 2002

[13] Siyka Zlatanova, Alias Abdul Rahman, Morakot Pilouk, *3D GIS: current status and perspectives*, Proceedings of ISPRS/8-12 July/Ottawa Canada/CDROM/8p, 2002

[14] OGC, *Abstract specifications overview*, available at http://www.opengis.org, 1999

[15] OGC, *The OGC technical committee policies & procedures*, available at http://www.opengis.org, 2000

[16] OGC, *OpenGIS Simple Features Specification for SQL*, available at http://www.opengis.org, 1999

[17] Oracle, *Oracle Spatial users guide and reference*, available at http://www.oracle.com/technology/documentation, 2003

[18] Oracle, *Oracle Spatial 10g data sheet*, http://www.oracle.com/technology/products/spatial/

[19] Steven Feuerstein, *Oracle PL/SQL programming 2nd edition*, O'Reilly, 1997

[20] Ingres, *Object management extension user guide*, available at http://opensource.ca.com/projects/ingres/documents, 2003

[21] Informix, *IBM Informix Spatial DataBlade Module users guide*, available at http://www.informix.com, 2003

[22] IBM DB2, *IBM DB2 Spatial ExtenderUsers guide and reference*, available at http://www-306.ibm.com/software/data/db2/,2003

[23] Postgres, *PostGIS manual*, available at http://postgis.refractions.net/documentation.php, 2002

[24] Bentley, *MicroStation online document*, available at http://www.bentley.com

[25] Mathworld, *URL: mathworld.wolfram.com*

[26] OpenGL, *OpenGL online Reference*, available at http://www.opengl.org

[27] AutoCAD, *AutoCAD 2004 help documents*, 2004