Direction as a Spatial Object: A Summary of Results *

Shashi Shekhar, Xuan Liu
Computer Science Department, University of Minnesota
EE/CS 4-192, 200 Union St. SE., Minneapolis, MN 55455
telephone: (612)624-8307
[*shekhar*|*xliu*]@*cs.umn.edu*
http://www.cs.umn.edu/research/shashi-group

Abstract

Direction is an important spatial relationship that is used in many fields such as geographic information systems(GIS) and image interpretation. It is also frequently used as a selection condition in spatial queries. Previous work has modeled direction as a relationship predicate between spatial objects. Conversely, in this paper, we model direction as a new kind of spatial object using the concepts of vectors, points and angles. The basic approach is to model direction as a unit vector. Given an ordered pair (p_1, p_2) of spatial points, one can define a direction, which can take the values of North, Northwest, 3 o'clock, etc. to represent the corresponding qualitative directional predicates on (p_1, p_2) in previous work. Direction objects can also have quantitative operations such as direction-composition and direction-reverse. This novel view of direction has several obvious advantages: First, it allows us to define the orientation of spatial objects; Second, a richer set of predicates and operators on direction and orientation can be defined; Third, new spatial data types such as oriented spatial objects and unbounded spatial objects can be defined. Finally, the object view of direction makes it easy to do direction reasoning, which is useful in spatial query processing and optimization.

Keywords: Direction, Orientation, unbounded object, oriented spatial object.

1 Introduction

Direction is often used in daily life when people communicate about a geographic space. It is frequently used as a selection condition in spatial queries or used for similarity accessing in image databases. Example queries used in army battlefield visualization are "Is there anything *over* the ridge?," "Put me in the tank *left* of that building," and "Let's move to the *north* of the tree." The first example refers to a viewer-based orientation, the second can be defined on either the intrinsic orientation of the building(object-based) or a viewer, and the third example refers to the absolute direction with respect to the tree. In order to handle the kind of queries that contain direction constraints in the selection criteria, a spatial database system should provide a way for users to specify the absolute direction and the viewer/object based orientation.

The common means of dealing with direction in GIS is to view direction as a spatial relation between objects[4, 18, 8, 22, 3, 14, 9, 6, 20, 13]. In this paper, we view direction from a different perspective: as a spatial object. The basic approach here is to model direction as a unit vector and orientation as a set of directions. As a spatial object, direction can have its own attributes, its own operators, so that a richer set of predicates and operators on direction and orientation can be defined. Second, new spatial data types such as oriented spatial objects and unbounded spatial objects, can be easily defined using the direction object. The object view of direction also makes it easy to do quantitative direction reasoning by using only simple vector algebra, which also reduces the large number of inference rules that is commonly needed. This is useful in the processing and optimization of spatial queries that contain direction constraints. We propose a new spatial object hierarchy and model the direction for three systems: absolute, object-based and viewer-based orientation.

We will not discuss the philosophical issue of whether directions are object/entities or not in the strictest sense. Absolute directions like North, South, East, West can be defined using a coordinate system without reference to any other spatial object. Thus these can be considered to objects in their own right. Object view for relative directions (e.g. left, front) needs further thought because they are defined with respect to coordinate systems attached to other objects. This issue gets complicated by the fact that a vector can be defined in Mathematics independently in context of a vector space or via referring to an ordered pair of points in an affine space. The issue needs to be explored further.

The organization of this paper is as follows: In section 2, we do a literature survey of the work on modeling direction and compare it to our work. We define a mathematical framework and propose ADTs in section 3. Direction modeling for three systems are defined in section 4, and a new spatial object hierarchy is developed in section 5. In section 6, we model directions between non-point spatial objects by using new spatial objects defined in section 5. Finally,

[1]This work is sponsored in part by the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. This work was also supported in part by NSF grant #9631539.

we conclude the paper with discussions and suggestions for future research.

2 Related work and our contributions

The research work on direction modeling has been carried out in several areas such as geographic information systems and image analysis. Most of the work is on how to capture the semantics of direction relations, and further, how to do spatial reasoning on the direction[3, 4, 6]. There are two major direction reference frames used to model direction in 2D space: the cone-based model[14], and the projection-based model[4, 6]. Frank[1] compared these two models and found the projection-based reference frame to be better in many aspects. Most attention has been paid to point-based objects. The most common way to model directions between extended objects is through the object's Minimum Bounding Rectangle(MBR), where direction relations are obtained by applying Allen's [2] interval relations along the x and y axis, in which case, 169 different relations[3] can be distinguished. some work based on MBR has been proposed on picture indexing in pictorial databases[16, 22] and some work aligns each boundary box to the object's major axis[12], which makes it possible to satisfy different reference frames[9]. But since MBRs are geometric approximations of spatial objects, they may be too coarse, and hence result in inconsistency and misinterpretation. Freska [5] proposed an alternative method: semi-intervals to formalize the one-dimensional temporal relation based on incomplete knowledge of the object. To overcome the limitations of MBR for the extended object model, Egenhofer [8] introduced a Direction-Relation Matrix to represent cardinal directions. Based on the projection-based frame, it partitions the space around the reference object and records into which direction tiles a target object falls. But this model still has limitations in the modeling of line objects, and it is limited to 2D space. Little work has been done on directions in 3D space [7].

The previous work modeled direction as a boolean spatial relation between spatial objects. This seems to be a natural mapping of direction relations that is used in geographic space. But this modeling method has some limitations. Operations on direction are limited. It is not possible to perform quantitative operations, such as vector-addition, translation, etc. Oriented(directed) objects and unbounded objects cannot be represented in the spatial data model.

This paper models direction as a spatial object: a unit vector. This novel view of direction has several advantages over the binary boolean relation view. Being modeled as a spatial object, a direction object can have its own attributes and operator set. The implementation of operators can use vector algebra, making a richer set of predicates and operators on direction feasible. Secondly, new spatial data types such as oriented objects and unbounded objects can be defined at the abstract object level. The object view of direction also makes direction reasoning easy. Basic vector algebra is sufficient for inferencing new directions, and no special or new qualitative rules are needed. This reduces the complexity of direction reasoning, and involving appropriate quantitative information is useful in spatial query processing and optimization in spatial databases.

3 Basic Concepts and proposed ADTs

3.1 Points, Vectors, Angles

The basic concepts we use here are points, vectors and angles, as illustrated in figure 1. A point has a position in

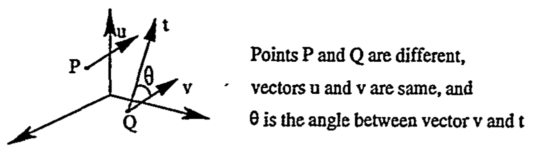

u
t
P
θ
v
Q
Points P and Q are different,
vectors u and v are same, and
θ is the angle between vector v and t

Figure 1: Diagram of points, vectors and angles

space, which can be described using coordinates in a coordinate system. The only characteristic that distinguishes one point from another is its position. A vector, in contrast, has both magnitude and direction but no fixed position in space. An angle between two vectors represents the direction deviation from one vector to the other. Here, by angle we mean the smaller one between two vectors, like θ in diagram 1. In this paper, we denote points by capitalized letters, such as P, and denote vectors as lower case letters with an arrow above, such as $\vec{v}$.

For simplicity we will use the Cartesian coordinate system to represent points and vectors in this paper. The Cartesian coordinate system in 3D is a right-handed rectangular coordinate system, with three axes x, y and z perpendicular to each other and intersecting at the origin. A point P in space is represented as an ordered triple (x, y, z), where x, y and z are rectangular components of P on the x, y and z axes , respectively. Let $\vec{U_x}, \vec{U_y}$ and $\vec{U_z}$ represent three unit vectors which are the basis vectors of the coordinate system. A vector $\vec{v}$ can be defined as a linear combination of these basis vectors in the form of: $\vec{v} = a\vec{U_x} + b\vec{U_y} + c\vec{U_z}$, where a, b and c are some numbers.

Operands	OPs	Definition
Point(s)	-	$\vec{pq} = Q - P = (x_q - x_p)\vec{U_x} + (y_q - y_p)\vec{U_y} + (z_q - z_p)\vec{U_z}$
	+	$\vec{a} + \vec{b} = (x_a + x_b)\vec{U_x} + (y_a + y_b)\vec{U_y} + (z_a + z_b)\vec{U_z}$
	-	$\vec{a} - \vec{b} = (x_a - x_b)\vec{U_x} + (y_a - y_b)\vec{U_y} + (z_a - z_b)\vec{U_z}$
Vector(s)	$\odot$	$\vec{a} \odot \vec{b} = \lvert a\rvert\lvert b\rvert\ cos\theta = x_a x_b + y_a y_b + z_a z_b$
	$\times$	$\vec{a} \times \vec{b} = \lvert a\rvert\lvert b\rvert\ sin(\theta)\ \vec{u}$, where $\vec{u}$ is a unit vector, perpendicular to $\vec{a}$ and $\vec{b}$
	scale	$m\vec{a} = mx_a\vec{U_x} + my_a\vec{U_y} + mz_a\vec{U_z}$, where m is a number
Point, Vector	+	$R = P + \vec{a} = R(x_p + x_a, y_p + y_a, z_p + z_a)$ a point away from P with distance $\lvert\vec{a}\rvert$ along the direction of $\vec{a}$

Table 1: Operations on vectors and points

Table 1 summarizes representative operations on points and vectors. Here $P(x_p, y_p, z_p)$ and $Q(x_q, y_q, z_q)$ are two points, θ is the angle between vectors $\vec{a}$ and $\vec{b}$, where $\vec{a} = x_a\vec{U_x} + y_a\vec{U_y} + z_a\vec{U_z}$ and $\vec{b} = x_b\vec{U_x} + y_b\vec{U_y} + z_b\vec{U_z}$. We can see that subtraction operation(-) is applied to point pairs, resulting in a vector. The operations of addition(+), subtraction(-), scale multiplication, dot product(⊙), and cross product(×) are available between vector operands. The definitions are given in terms of the components in the Cartesian coordinate system. There is only one operation(+) available between a point and a vector which actually produces another point. The associativity, commutativity and distribution properties of these operators will be examined in the future work towards integrating vector objects in query languages.

3.2 Direction and Orientation

Direction is defined as a unit vector, i.e., a vector with its magnitude equal to 1. Table 2 defines the operations on directions.

Operations	Definition
composition	$\vec{d1} + \vec{d2} = \frac{\vec{d1}+\vec{d2}}{\lvert\vec{d1}+\vec{d2}\rvert}$
deviation	$cos\theta = \vec{d_1} \odot \vec{d_2}$
reverse	$(-1) \times \vec{d1}$
between[1]	$\vec{d}$ between $\vec{d_1}$ and $\vec{d_2}$ if $\exists c_1 > 0, c_2 > 0$ s.t. $\vec{d} = c_1\vec{d_1} + c_2\vec{d_2}$
among	$\vec{d}$ among $\vec{d_1}$, $\vec{d_2}$ and $\vec{d_3}$ if $\exists c_1 > 0, c_2 > 0, c_3 > 0$ s.t.$\vec{d} = c_1\vec{d_1} + c_2\vec{d_2} + c_3\vec{d_3}$

Table 2: Operations of Direction

The operations on directions can be classified into three categories. The first category is the operations that produce new directions. *Composition* and *reverse* are operations in this category. The *composition* operation is actually achieved by *vector-addition*, and the resulting vector is scaled down by its magnitude to be a unit vector, which represents the new direction. The *reverse* operator produces the opposite direction vector. The second category is to calculate the deviation between two directions. Operator *deviation* calculates the cosine of the angle between two directions, and hence gives the direction deviation of one direction from the other. A pair of vectors are orthogonal if their dot-product returns zero, i.e., they have 90^0 deviation. The last category is to test the relationship among directions. The operators *between* and *among* belong to this category. In figure 2, $\vec{d}$ is between $\vec{d_1}$ and $\vec{d_2}$; however, $\vec{d_1}$ is not between $\vec{d}$ and $\vec{d_2}$. As we will see in later sections, these

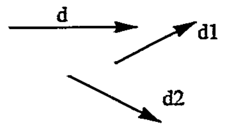


Figure 2: between operator

three categories of direction operations make the modeling of direction concise and flexible.

Orientation is modeled as a spatial object which consists of a point of origin and N pair-wise orthogonal directions, where N is the dimension of the embedding space. The origin point and the N directions form a Cartesian coordinate system. In 3D space, the three directions may be labeled the Back-Front, Left-Right, and Below-Above directions of the orientation. Formally, we can define orientation and its operations in 3D space as follows:

Orientation is a quadruple O= $\langle$OP, $\vec{front}, \vec{right}, \vec{above}\rangle$, where OP is a point, and $\vec{front}$, $\vec{right}$, $\vec{above}$ are three orthogonal directions. It has two operations:

- $translate(O, \vec{v}) = Orientation\ O' = \langle translate(OP, \vec{v}), \vec{front}, \vec{right}, \vec{above}\rangle$;
- $rotate(O, rotationMatrix) = Orientation\ O' = \langle OP, \vec{front_n}, \vec{right_n}, \vec{above_n}\rangle$, where $(\vec{front_n}, \vec{right_n}, \vec{above_n}) = (\vec{front}, \vec{right}, \vec{above}) \odot rotationMatrix$;

An example of the rotationMatrix which rotates the orientation along the $\vec{above}$ axis for an angle θ is[17]:

$$R^{\theta}_{\vec{above}} = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Table 3 gives an illustration of these operations.

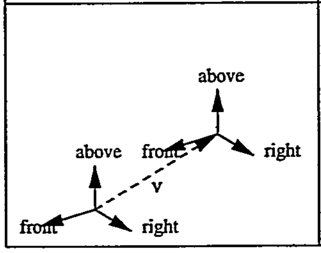
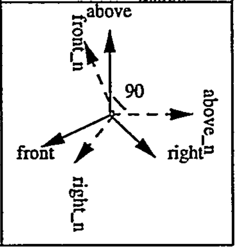
translate(O, $\vec{v}$)	rotate(O, $R^{\theta}_{\vec{above}}$)
	 above front_n 90 above_n front right right_n

Table 3: Operations on orientation

3.3 Abstract Data Types for Modeling Direction

As a summary of the above discussion, we give the definition of ADTs for vector, direction and orientation in Table 4. The C++ like syntax is used here. The column labeled *attributes* declares the member variables of each class, and the *operations* column declares the interfaces of operations for each class. The vector class has three member variables of the real type. These variables represent the three coefficients when the vector is written as the linear combination of basis vectors of the coordinate system. The constructor constructs a vector object, given three real arguments. The five vector operators are declared as member operator functions. Direction is defined as a subclass of the vector class. It can be constructed either from a vector or from an ordered triple. Besides inheriting attributes and operators from vector class, direction also adds new operators such as *between* and *among*. The orientation class has four member variables which form a Cartesian coordinate system. *Translate* and *rotate* are declared as the member operator functions of the orientation class.

4 Applications

In this section, we will deal with point-based objects for simplicity. There are three different perspective systems We define absolute directions(e.g. north, south), object-based directions(e.g. left, above), and viewer-based directions.

[1]This definition works well as long as vector $\vec{d_1}$ is not parallel to vector $\vec{d_2}$. The parallel case can be handled in a user defined manner.

ADT	attributes	representative operations
vector	x-comp: float; y-comp: float; z-comp: float;	vector(float, float, float); // constructor float magnitude(); vector operator + (vector); vector operator - (vector); vector operator scale(); float operator ⊙ (vector); vector operator × (vector);
direction subclass of vector	constraint = unit magnitude	direction(float, float, float); direction(vector); direction operator Composition(direction); direction operator reverse(); float operator deviation (direction); boolean between (direction, direction); boolean among(direction, direction, direction);
orientation	$OP : point$; $\vec{front}$: direction; $\vec{right}$: direction; $\vec{above}$: direction;	orientation(float, direction, direction,direction); orientation operator translate(vector); orientation operator rotate(rotate-matrix);

Table 4: Abstract Data Types for direction and orientation

4.1 Absolute Direction

Absolute direction in embedding space is defined as a relationship among objects based on their locations in embedding space. For simplicity here, we will ignore the elevation and earth's curvature, and use a projection-based model to map the space to a 2D local coordinate system with north up and east right[2].

We model absolute directions in two ways: constant direction objects and direction predicates. Constant direction objects are created for frequently asked directions, e.g: $\vec{east}$, $\vec{west}$, $\vec{north}$, $\vec{south}$, $\vec{NW}$, $\vec{NE}$,$\vec{SE}$,$\vec{SW}$. Table 5 illustrates how we can define the constant directions in terms of coordinates in local embedding space. The unit vector is denoted by an ordered pair(a, b) which represents vector $a\vec{U}_{east} + b\vec{U}_{north}$.

unit vector	(1,0)	(0,1)	(-1,0)	(0,-1)
directions	$\vec{east}$	$\vec{north}$	$\vec{west}$	$\vec{south}$
unit vectors	$(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$	$(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$	$(-\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$	$(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$
directions	$\vec{NE}$	$\vec{NW}$	$\vec{SW}$	$\vec{SE}$

Table 5: constant absolute direction

Introducing constant direction objects provides much flexibility in describing and deciding any direction. By using the operator *deviation*, for any given direction, we can calculate how much deviation is from one constant direction. We can also produce a new direction, given the deviation from a specific constant direction. This is very useful in the layout of facilities.

Second, we also provide corresponding direction predicates to make it convenient for users to specify the direction relation between two objects. They are *East*, *West*, *South*, *North*, *Northwest*, *NorthEast*, *SouthWest*, *SouthEast*. For any two objects, assuming their centroids are P_1, P_2, we represent the relationship "P_1 *is to the east of* P_2" by predicate $East(P_1, P_2)$. The predicate is *true* iff equation $\frac{\vec{east} \odot \vec{P_2P_1}}{|\vec{P_2P_1}|} = 1$ holds. Other predicates can be defined similarly.

[2]For a global view, we can extend to the ellipsoidal coordinate system[11]

The above constant directions and predicates give precise direction calculation, that is, only one angle matches each case. Sometimes users may only be interested in approximate direction, such as a direction range: *between north and northwest*. The operator *between* makes this approximate direction predicate possible. If we want to test whether P_1 is *between north and northwest* of P_2, we can use the *between* operator in C++ notation as: $\vec{P_2P_1}.between(\vec{north}, \vec{NW})$.

4.2 Object-based direction

Object-based direction is the direction of the target object with respect to the orientation of the reference object. The reference object is an oriented object, while the target object may or may not have orientation. We will use O_B to represent the orientation of object B, and $O_B.\vec{front}$, $O_B.\vec{right}$ and $O_B.\vec{above}$ are the three directions of B's orientation. By using the direction operator *reverse* , the directions of B's behind, left and below can be described as $-O_B.\vec{front}$, $-O_B.\vec{right}$ and $-O_B.\vec{above}$, respectively.

Given a target point object A, and a reference point object B, we can decide which direction A is relative to B by the vector dot-product. First, a vector $\vec{BA} = A - B$ is obtained. The direction of point object A with respect to point object B's orientation can be decided by the dot-product between vector $\vec{BA}$ and the three directions $O_B.\vec{front}$, $O_B.\vec{right}$ and $O_B.\vec{above}$.

Direction predicates	FDP	RDP	ADP
A in front of B= $front(A, B)$	>0	0	0
A behind of B = $behind(A, B)$	<0	0	0
A right to B = $right(A, B)$	0	>0	0
A left to B = $left(A, B)$	0	<0	0
A above B = $above(A, B)$	0	0	>0
A below B = $below(A, B)$	0	0	<0
$\vec{BA}.between(O_B.\vec{front}, O_B.\vec{right})$	>0	>0	0
$\vec{BA}.among(O_B.\vec{front}, O_B.\vec{above}, O_B.\vec{right})$	>0	>0	>0

Table 6: Object-based orientation

Table 6 gives some examples of calculating object-based direction. The first column consists of example direction predicates and the remaining three columns are the conditions needed to satisfy the predicates. Here, FDP, RDP, and ADP represent three dot products: $\vec{BA} \odot O_B.\vec{front}$, $\vec{BA} \odot O_B.\vec{right}$ and $\vec{BA} \odot O_B.\vec{above}$ respectively. The predicate returns *true* if and only if all three corresponding conditions hold. The first six rows illustrate the calculation of straight directions, such as: front, left, above, etc. As in row 1, if the dot-product of vector $\vec{BA}$ and direction $O_B.\vec{front}$ is greater than zero, and the dot-products between vector $\vec{BA}$ and the other two directions of orientation O_B are both zeros , then $front(A, B) = true$, which means that A is in front of B. These direction predicates are for precise direction checking, whereas the other two categories of predicates are for direction range checking. Operator *between* is used to test if A is located in the region between two directions. Here *between* is used in C++ notation. The predicates in this category are for the directions in 2D space. The predicates in third categories test if A is among three directions in a 3D space. We represent these predicates using the operator *among*.

4.3 Viewer-based direction

Viewer-based direction refers to the direction relation which is measured from the viewer's perspective.

There are three related components in this system: target object A, reference object B, and the viewer. The viewer has his/her own orientation, where objects A and B may or may not be oriented objects. Given these three components, we can estimate the direction of object A relative to object B from the viewer's perspective in a way similar to that used with the object-based system. The similar predicates but w.r.t. the viewer are defined, and are calculated by the dot-product between vector $\vec{BA}$ and the three directions of the reference orientation. The reference orientation here is the viewer's orientation: O_V, and hence the three dot-products needed to calculate are $\vec{BA} \odot \vec{O_V.front}$, $\vec{BA} \odot \vec{O_V.right}$, and $\vec{BA} \odot \vec{O_V.above}$. A similar table can be obtained.

In the viewer-based direction system, the viewer may change his/her orientation or position(unlike objects, which are still). Using the operators *rotate* and *translate* on the orientation, we can easily generate new orientations.

5 Defining New Spatial Types Using Direction and Vector

Some work has been done on modeling space into a spatial geometry hierarchy. Open GIS Consortium[10] proposed a spatial object hierarchy for bounded shape objects in 2D space as shown in figure 3. The hierarchy consists of four

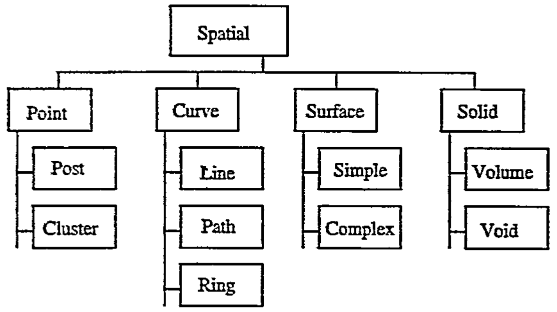


Figure 3: Bounded Spatial Object Hierarchy

spatial data types: points, curves, surfaces, and solids. Each data type can be further classified into several subtypes. Since these data types model only bounded shape objects, the hierarchy has limitations in modeling objects such as maps, rivers, roads and buildings which have their own directions/orientations. For example, with few exceptions, printed maps are all oriented with north up. This is not only for better visualization, but for more flexibility with spatial queries. It allows queries like: "Which state is to the *left* of Minnesota?" which are frequently asked by non-geographers who have little sense of north, west, etc.

By using the direction and orientation objects we defined above, we can extend the spatial hierarchy to include oriented objects and unbounded objects. In the new spatial hierarchy, maps and roads can be defined as oriented objects.

5.1 Oriented(directed) spatial object

There are many objects in geographic space that have their intrinsic directions/orientations in addition to their locations and shapes, such as buildings, rivers and roads. As we discussed above, modeling maps as oriented objects provides users more flexibility in query specification. It does allow users who have little knowledge of geographic space can query about directions intuitively based on his/her own orientation.

Table 7 gives some queries that can only be answered when the reference objects and/or viewers are modeled as oriented objects.

Queries
Put me in front of the building.
Is there anything behind the ridge?
Let's move back a little.
How far is the next intersection down this road?

Table 7: Queries Need Orientation Information

Introducing *Orientation* class into spatial object hierarchy makes it easy to formalize oriented objects. In addition to the attributes such as *location* and *shape* for the geometric shape objects, an oriented object has an attribute *orientation*, which is an instance object of class orientation. The operators available for oriented objects include both topological operators and direction and orientation operators.

5.2 Open Shapes

Open shapes mean the geometries whose boundaries are unknown, or unimportant, or infinite.

Examples of open spatial objects are open lines, open planes, and open spaces. A common way to define these in 3D Cartesian geometry is in terms of an equation such as the parametric equation[15]: $x = x_0 + at, y = y_0 + bt, z = z_0 + ct$ for a line through the point (x_0, y_0, z_0), Ax+By+C=0 for a plane, and Ax+C=0 for a 3D space. It would be more convenient if there were corresponding object classes so that users could describe them at an abstract level and use them to describe the space. The *direction* object could be used for this purpose.

Defining Open line, Open Rectangle

We define open lines and open rectangle regions using directions and points here. Table 8 explains how to formalize each open object with examples in figure 4.

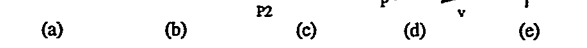


Figure 4: Examples of open shapes

There are several subtypes in each type. For an open line, it could be 1-end open(figure 4a), which needs a start point and a direction to define it, or it could be 2-end open(4b), which needs an intermediate point and a direction to describe it. The open rectangle also has three subtypes: 1-side

Type	Descriptions	figure
openline1(1-end open line)	start-point, direction	(a)
openline2(2-end open line)	intermediate-point, direction	(b)
openrect1 (1-side open rectangle)	$vertex_1$,$vertex_2$,direction where direction perpendicular to segment formed by the two vertices	(c)
openrect2 (2-side open region)	two 1-end open lines from the only vertex	(d)
openrect3 (3-side open rectangle)	2-end open line, direction	(e)

Table 8: Defining open objects

open rectangle (4c), described by two vertices of the rectangle and one direction which is perpendicular to the edge connected by the two vertices. A 2-side open rectangle is described using two 1-end open lines which start from the vertex of the rectangle. Also a 3-side open rectangle could be defined by a 2-end open line and a direction. It is obvious that extending to an open cube is easy.

5.3 New Spatial Data Type Hierarchy

Figure 5 illustrates the extended spatial data type hierarchy which consists of shape objects, vectors and oriented(directed) objects. The spatial data types and their operators can be embedded in an extended query language to implement a spatial query language.

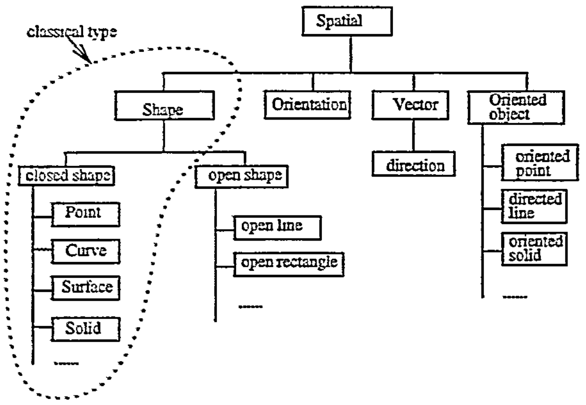


Figure 5: Extended Spatial Data Type Hierarchy

Many new operators are available in the extended spatial object model. Between vector objects and shape objects, affine transformations can be performed. The addition of these operations makes some GIS applications easy. For example, Urban planning needs interactive placement/orientation of facilities, location-based query exploration can use affine operations, and using these operations also can improve viewer flexibility, so that we do not have to look down on earth from space. Instead, a viewer can view from any direction by performing rotation transformation.

6 Modeling Direction Between Non-point Objects

The extended spatial object hierarchy increases our power to describe geographic space by providing new spatial types such as open shapes. We can use open shapes to model the directions between extended objects by converting the calculation of directional relationships to the calculation of topological relationships between objects.

Given two objects TO and RO, we want to decide the direction of target object TO related to reference object RO. Using the approach in [8], first we obtain the MBR of object RO and partition the space around object RO into nine direction tiles based on the MBR of object RO, which is a rectangle ABCD. as in figure 6. We represent each direction tile by the object in new spatial hierarchy as in table 9.

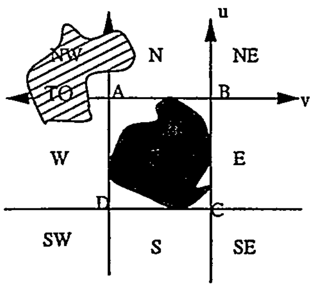


Figure 6: Object TO and RO

Rectangle	representation for figure 6
NW	$openrect2(openline2(A,\vec{u}),openline2(A,-\vec{v}))$
N	$openrect1(A,B,\vec{u})$
NE	$openrect2(openline2(B,\vec{u}),openline2(B,\vec{v}))$
E	$openrect1(B,C,\vec{v})$
SE	$openrect2(openline2(C,-\vec{u}),openline2(C,\vec{v}))$
S	$openrect1(C,D,-\vec{v})$
SW	$openrect2(openline2(D,-\vec{u}),openline2(D,-\vec{v}))$
W	$openrect1(A,D,-\vec{v})$
O	rectangle(A,B,C, D)

Table 9: Direction tile

Eight of the nine direction tiles are open rectangles. NW, NE, SE, SW are 2-side open rectangles, and N, W, E, S are 1-side open rectangles. In order to test the direction of object TO related to object RO, we can test into which direction tile TO falls. The calculation is then converted to the overlap relationship between object TO and the direction tiles. For example,

$$North(TO,RO) <==> Overlap(TO,N)$$
$$NorthWest(TO,RO) <==> Overlap(TO,NW)$$

we can calculate the other directions in the same way. Finally, we union all the tiles that overlap with A. For example, in figure 6, A is North and NW and West of object B.

7 Discussions

In this paper, we model direction as a new kind of spatial object using the concepts of vectors, points and angles. The basic approach here is to model direction as a unit vector. Besides enabling qualitative reasoning on which most of the previous work has focused, direction objects can also have quantitative operations such as *direction-addition*, *direction-multiplication* and operator *translate* on orientation. This novel object view of direction has several obvious advantages: It allows the definition of the orientation of spatial objects; it gives a richer set of predicates and operators on direction and orientation. And new spatial data types such as oriented spatial objects and unbounded spatial objects can be defined. Finally, quantitative direction reasoning, which is useful in spatial query processing and optimization, can be implemented simply by basic vector algebra.

The benefits of the new viewpoint are obvious and promising. In future work, we would integrate vector abstract data types with an extensible query language to evaluate their efficiency in the context of various GIS applications[19, 21]. We would also like to explore the use of vectors to model values of spatial attributes such as wind-velocity, magnetic field of the earth, etc.

Acknowledgments

This work is sponsored in part by the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. This work was also supported in part by NSF grant #9631539 Thanks to Dr. Sanjay Chawla for technical commentary and to Christiane McCarthy for helping to improve the readability of the paper.

References

[1] A.Frank. Qualitative Spatial Reasoning: Cardinal Directions as an Example. *International Journal of Geographical Information Systems*, 10(3):269–290, 1996.

[2] J. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[3] D.Papadias, M. Egenhofer, and J. Sharma. Hierarchical Reasoning about Direction Relations. In *Fourth ACM Workshop on Advances in Geographic Information Systems*, pages 105–112. ACM, 1996.

[4] A. Frank. Qualitative Spatial Reasoning about Cardinal Directions. In *Auto carto 10, D.Mark and D. White, eds., Baltimore, MD*, pages 148–167, 1991.

[5] C. Freksa. Temporal Reasoning Based on Semi-Intervals. *Artificial Intelligence*, 54:199–227, 1992.

[6] C. Freksa. Using Orientation Information for Qualitative Spatial Reasoning. *Theories and Methods of Spatio-Temporal Reasoning Geographic Space*, 639:162–178, 1992.

[7] Klaus-Peter Gapp. Basic Meanings of Spatial relations: Computation and Evaluation in 3D space. *AAAI*, 2:1393–1398, 1994.

[8] R. Goyal and M. Egenhofer. The Direction-Relation Matrix: A Representation of Direction Relations for Extended Spatial Objects . In *UCGIS Annual Assembly and Summer Retreat, Bar Harbor, ME*, 1997.

[9] G.Retz-Schmidt. Various Views on Spatial Prepositions. *AI Magazine*, 9(2):95–105, 1988.

[10] Open GIS Consortium Inc. Opengis simple features specification for sql. http://www.opengis.org.

[11] Open GIS Consortium Inc. Spatial reference systems. http://www.opengis.org/techno/specs.htm.

[12] E. Jungert. The observer's point of view: An Extension of Symbolic Projections. *Theories and Methods of Spatial-Temporal Reasoning in Geographic Space*, 639:179–195, 1992.

[13] D. Papadias and T. Sellis. Qualitative representation of Spatial Knowledge in Two-Dimensional Space. *VLDB Journal*, 3(4):479–516, 1994.

[14] Donna J. Peuquet and Zhan Ci-Xiang. An Algorithm to Determine the Directional Relationship Between Arbitrarily-shaped Polygons in the plane. *Pattern Recognition*, 20(1):65–74, 1987.

[15] M.H. Protter and C.B. Morrey. *Modern Mathematical Analysis.* Addison-Wesley Publishing Company, Inc., 1996.

[16] P.W.Huang and Y.R. Jean. Using 2D C^+-String As Spatial Knowledge Representation For Image Database Systems. *Pattern Recognition*, 30(10):1249–1257, 1994.

[17] David F. Rogers. *Mathematical Elements for Computer Graphics.* McGraw-Hill Publishing Company, 1990.

[18] S. Shekhar, S. Ravada A.Fetterer, X.Liu, and C.T. Lu. Spatial Databases: Accomplishments and Research Needs. *University of Minnesota technical report, Csci TR97-016*, 1997.

[19] S. Shekhar, S. Ravada, V. Kumar, and etc. Parallelizing a GIS on a Shared-Address Space Architecture. *IEEE Computer*, 29(12):42–48, December 1996.

[20] Shashi Shekhar, Mark Coyle, and etc. Data Models in Geographic Information Systems. *CACM*, 40(4):103–111, 1997.

[21] Shashi Shekhar and Duen-Ren Liu. CCAM: A Connectivity-Clustered Access Method for Networks and Network Computations. *TKDE*, 9(1):102–119, 1997.

[22] Y.I.Chang and B.Y. Yang. A Prime-Number-Based Matrix Strategy for Efficient Iconic Indexing of Symbolic Pictures. *Pattern Recognition*, 30(10):1–13, October 1997.