Representing Polygon Areas  
and Testing Point-in-Polygon Containment in a Relational Database.
Jim Gray, Alex Szalay
29 August 2002, revised 14 December 2002

Any half-space \( H \) of the N dimensional space \( S \) can be expressed as \( H = \{x \in S \mid f(x) > 0\} \) for some function \( f \). The intersection of a set of half-spaces \( \{H_i\} \), defines a convex hull of points. \( C = \{x \in S \mid x \in H_i \text{ for all } H_i\} \). A domain \( D \) is the union of a set of convex hulls \( D = \{x \in C_i\} \).

To be concrete, in two dimensions a planar half-space is defined by \( ax + by > c \) and a polygon is defined by the intersection of three or more of these half-spaces, as in the figure at right. If the set of equations for the convex represented as a set of triples \( C = \{(a,b,c)\} \) then a point \((x,y)\) is in the convex if and only if \( ax + by > c \) for all \((a,b,c) \in C\).

These ideas can be translated into relational terms quite simply:
A domain is just a placeholder for a name and an ID.

\[
\text{create Domain } \begin{cases}
\text{domainID int identity() primary key,} \\
\text{type char(16), -- short description} \\
\text{comment varchar(8000), -- long description} \\
\text{predicate varchar(8000), -- compiled test for containment.} \\
\text{-- see fDomainPredicate() below.}
\end{cases}
\]

Half-spaces and convexes are represented together in a second table (we do 3D half spaces here and represent \( a,b,c,d \) as \( x,y,z,l \)).

\[
\text{create HalfSpace } \begin{cases}
\text{domainID int not null }, -- domain name \\
\text{foreign key references Domain(domainID)} \\
\text{convexID int not null, -- grouping a set of \( \frac{1}{2} \) spaces} \\
\text{halfSpaceID int identity(), -- a particular \( \frac{1}{2} \) space} \\
\text{x float not null, -- the \((x,y,z)\) parameters} \\
\text{y float not null, -- defining the \( \frac{1}{2} \) space} \\
\text{z float not null,} \\
\text{l float not null, -- the constant ("c" above)} \\
\text{primary key(domainID, convexID, halfSpaceID)}
\end{cases}
\]

The following SQL query all the convex hulls containing point \( @x, @y, @z \).

\[
\text{select convexID} \\
\text{from HalfSpace} \\
\text{where @x * x + @y * y + @x * z < l} \\
\text{group by all convexID} \\
\text{having count(*) = 0}
\]

This query groups all the half-spaces by convex hull. For each convex hull it asks how many of the half-spaces do NOT contain the point. If that answer is zero (\( \text{count(*)} = 0 \)), then the point is inside all the half-spaces and so inside the convex hull.

The key observation is that that HalfSpace table represents a domain in as a disjunct (or) of one or more convexes. Each convex is a conjunct (and) of all of its component half-spaces. A half-space may be negated by just changing the sign on \( x,y,z,l \), so the half-space table is just the disjunctive normal form representation of the domain. This representation has some shortcomings: it ignores points exactly on the surface (the inequality is strict), and more importantly, it does not easily support “holes” in a convex.
Once constructed they can be manipulated with the Boolean operations.

The ‘OR’ function just adds in all half-space rows from each of the two source domains with the new domain name and with the convexID renumbered.

The “AND” predicate is more subtle, it intersects each convex from the first domain with each convex from the second domain. If there are N and M convexes in the original domains, then there will be NxM convexes in the conjunction. A good algorithm would simplify this, discarding empty convexes (we have not implemented that – it looks difficult in SQL). This is just an application of deMorgan’s Law:

\[(A1 \cup A2) \& (B1 \cup B2) = A1&B1 \cup A1&B2 \cup A2&B1 \cup A2&B2\]

The Negation predicate is by far the most complex. It needs to build a new set of convexes that draw a negative half-space from each of the original ½ spaces. The “and” and “or” were simple SQL statements, the negation required a recursive definition doing the Cartesian product of the (negation of) each half space in the first convex with the negation of all the other convexes in the domain.

Given this machinery one can define a simple function to return all the domains containing point (x,y,z).

This query runs at the rate of 100K to 1M rows per second per cpu (the inner loop is embarrassingly parallel). Conversely if one has many points and wants all the points in a certain domain @domainID the query is

Again, this query runs at the rate of 100K to 1M comparisons per second. One can go about fifty times faster by translating the predicate into an expression of the form:

or (and ((p.x*a.x + p.y*a.y + p.z*a.z) >= a.l))

one would then combine with a select... where <predicate> and do an sp_execute of the resulting string. Unfortunately, sp_execute cannot be placed inside a function (exec cannot be inside a function)
so one must use a temporary table and a stored procedure rather than using the more efficient table-valued variable.

The routine fDomainPredicate(@domainID) indeed returns the compiled string. The precompiled string can be stored with the domain.

declare @query varchar(8000)
set @query =
    'select * '                        -- return all points
    'from Points p ' +                -- in the area
    'where ' + fDomainPredicate(@domainID) -- satisfying the predicate
    'exec (@query)'                   -- execute the query

This runs at 3 µs per point or at the rate of a million half-space tests per second per cpu if it is not IO bound. Deriving the predicate costs less than 1ms and the fixed cost of executing the predicate on a small set of points is about 6ms (all this on a 1Ghz machine). So, if more than ten thousand points are to be tested, the fixed cost of compilation is paid for by the speedup in point comparisons.

Until now the discussion has been in terms of 3D Euclidian space, because that is easiest to visualize. But, the algorithms and data structures apply to higher dimensions (more than 3D) by adding more parameters. They also apply to half spaces defined by higher-order polynomials (quadratic rather than linear equations.) Further, the algorithms apply to non-Euclidian spaces like the surface of the sphere. The thing that makes the algorithms work is the triangle inequality inherent in any metric space.

Our application is astronomy, so we are particularly interested in the celestial sphere. We represent spherical areas as a set of positive and negative convex-areas. Each convex area is a sequence of spherical edges defined by a plane intersecting the sphere. The plane is in turn defined by a normal unit vector \(v = (v_x, v_y, v_z)\) and length \(l\). Point \(p = (x, y, z)\) on the unit sphere is “inside” the plane if \((xyz)\cdot(v_x, v_y, v_z) > l\). A point is inside a convex area if it is inside each of the edges. Non-convex areas may be composed as the union of several convex areas. Swiss-cheese areas with holes in them can be composed of positive and negative convex areas. Figure 2 shows a complex convex area and also shows the 2-dimensional dot-product test for “inside” an edge.

**Rude questions:**

Why not use an index structure and bounding boxes?  
Ans: these ARE bounding boxes.  
Ans: we have few (hundreds) bounding boxes.  
Ans: good idea  

But what about Masks: then we have Millions of them.  
Ans: RIGHT! We need to limit mask searches to sections of the sky (frames or stripes or???)

Shouldn’t you simplify the convexes (discard redundant half-spaces, discard empty convexes)?  
Ans: Yes.

Neighbors
Appendix: The code

-- SkyServer Domain, Convex, and spatial test functions.
--
-- Creates Tables
--   Domain
--   HalfSpace
--
-- Creates functions.
--   fDomainsContainPoint -- returns table of domains containing a point.
--   fDomainPredicate -- computes predicate for a domain
--   fDomainNot -- helper function for spDomainNot
--
-- Creates procedures
--   spDomainNew -- create a new domain
--   spDomainNewConvex -- add a convex to a domain
--   spDomainNewConvexConstraint -- add a constraint to a convex
--   spDomainDrop -- drop a domain
--   spDomainOr -- create a new domain as OR of two others.
--   spDomainAnd -- create a new domain as AND of two others
--   spDomainNot -- create a new domain as NOT of another.

-- Dec-2149-2002 Jim: started

set nocount on

use tempdb

if exists (select * from sysobjects where id = object_id(N'HalfSpace'))
  drop table HalfSpace
if exists (select * from sysobjects where id = object_id(N'domain'))
  drop table domain

GO
--/H Defines a spatial subset (a union of convexes.
--
--/T A domain is a set of convexes that are bounded by a set of halfspaces
--/T defined by planes.
--/T the plane has the normal vector x,y,z and is at distance l along that
--/ vector.
--/T point cx, cy, cz is inside the space if cx*x + cy*y + cz*z <= l
--/T the opposite space is the negative of these 4 numbers.
--/T A domain has an ID, a type ('stave', 'stripe', 'frame',...), and a comment
--/ Domains may also carry a compiled predicate computed by fDomainCompile

create table Domain(
    domainID int identity primary key,       --/D domain's unique ID
    type  varchar(16)   not null default '',  --/D stripe, stave, ..., user,
    comment  varchar(8000) not null default '',  --/D description (e.g. run #)
    predicate varchar(8000) not null default ''   --/D pred to test point inside
)
go

if exists (select * from sysobjects where id = object_id(N'HalfSpace'))
drop table HalfSpace
go
create table HalfSpace(
    domainID int not null   --/D the unique identifier of the domain
    foreign key references Domain(domainID),
    convexID int not null,   --/D the unique identifier of a convex hull
    planeID  int identity,  --/D id of the plane defining the halfspace
    x   float not null, --/D the xyz vector normal to the halfspace
    y   float not null, --/D the xyz vector normal to the halfspace
    z   float not null, --/D the xyz vector normal to the halfspace
    l   float not null, --/D the vector length.
    primary key (DomainID, ConvexID, PlaneID)
)

-- End of table definitions, start of procedure definitions.
if exists (select * from sysobjects where id = object_id(N'fDomainPredicate'))
drop function fDomainPredicate

create function fDomainPredicate(@domainID int)
-------------------------------------------------------------
--/H Returns predicate testing if a point (cx,cy,cz) point is inside the domain
-------------------------------------------------------------
--/T Parameters:
--/T <ti> domainID int Predicate will be computed for this domainID
--/T <p> The predicate is of the form:
--/T <br>  ( ( (1*cx+0*cy+0*cz)>=0.5) AND  ((0*cx+1*cy+0*cz)>=0.5)
--/T <br>  AND ((0*cx+0*cy+1*cz)>=0.5) AND  ((1*cx+1*cy+1*cz)>=0.5)
--/T <br>  )
--/T <br> OR  ( (1*cx+0*cy+0*cz)>=0.6) AND  ((0*cx+1*cy+0*cz)>=0.6)
--/T <br>  AND ((0*cx+0*cy+1*cz)>=0.6) AND  ((1*cx+1*cy+1*cz)>=0.6)
--/T <br>  )
--/T <br> returns the empty string if the predicate is too large (more than 7.8kB).
--/T <br>
--/T Sample call to update all domain predicates
--/T <br> update Domain
--/T <br>  set predicate = dbo.fDomainPredicate(domainID)
--/T <br> where domainID != ''
--/T <br>
returns varchar(8000)
as
BEGIN
declare @convexID int  -- the current convex
declare @oldConvexID int  -- the previous convex
declare @predicate varchar(8000); set @predicate = ''
declare @clause varchar(8000); set @clause = ''
declare @x float, @y float, @z float, @l float
-------------------------------------------------------------
-- this cursor reads all the edges of all the convexes of the area
declare domain cursor read_only for
select convexID,x,y,z,l
from HalfSpace
where domainID = @domainID
open domain

-- loop over the edges building an OR of each convex
-- and within a convex an AND of its edges containment.
while (1=1)
begin
fetch next from domain into  @convexID, @x, @y, @z, @l
if (@@fetch_status <> 0) break
-- logic to handle a new convex (an OR)
if ((@oldConvexID is not null) and (@convexID != @oldConvexID))
begin
if (@predicate != '') set @predicate = @predicate + ' OR ';
set @predicate = @predicate + '(' + @clause + ');
set @clause = ''
end
set @OldConvexID = @convexID
-- logic to handle an edge within a convex (an AND)
if (@clause != '') set @clause = @clause + 'and'
set @clause = @clause + '(' + cast(@x as varchar(12)) + '*cx+'
+ cast(@y as varchar(12)) + '*cy+'
+ cast(@z as varchar(12)) + '*cz<='
+ cast(@l as varchar(12)) + ')'
end
-- loop ended, now tack on last convex clause
-------------------------------------------------------------
-- if resulting string is close to the varchar limit of 8000
-- return the null predicate. (the 200 char buffer is for the select clause)
if (len(@predicate) + len(@clause) > 7800)
returns null

)}
begin
  set @predicate = ''
  return @predicate
end
if (@predicate != '') set @predicate = @predicate + ' OR '
set @predicate = ' (' + @predicate + '(' + @clause + '))'
-- close and deallocate the cursor
close domain; deallocate domain
-- return the result
return @predicate
end

```
-- create a new domain, output and return value is domain ID
if exists (select * from sysobjects where id = object_id(N'spDomainNew'))
drop procedure spDomainNew

create procedure spDomainNew ( @type varchar(16),
   @comment varchar(8000))
as
begin
insert domain values (@type, @comment, '')
return @@identity
end

-- start a new convex in a domain, output and return value is convexID
if exists (select * from sysobjects where id = object_id(N'spDomainNewConvex'))
drop procedure spDomainNewConvex

create procedure spDomainNewConvex(@domainID int) as
begin
declare @convexID int
select @convexID = max(convexID) + 1 from HalfSpace where domainID = @domainID
set @convexID = coalesce(@convexID, 0)
return @convexID
end

create procedure spDomainNewConvexConstraint ( @domainID int, @ConvexID int, @x float, @y float, @z float, @l float)

--/H add a new constraint to a convex in a domain, return constraintID
--/T Parameters:
--/T <li> domainID int      ID of the domain to get the new constraint
--/T <li> convexID int      ID of the convex to get the new constraint
--/T <li> x, y, z float   The 3-tuple that defines the constraint.
--/T <li> l float  The length of the vector.
--/T <br> "inside" points (cx,cy,cz) have the property (cx*@x+cy*@y+cz*@z > @l)
--/T <br> returns halfspaceID int  the unique ID of the new halfspace.
--/T <br>Sample call to add a constraint
--/T <br><samp>
--/T  <br> exec spDomainNewConvexConstraint @domainID, @convexID, @x, @y, @z, @l
--/T  </samp>
--/T <br> see also spDomainNew, spDomainNewConvex, spDomainDrop,...

as
begin
Insert HalfSpace values(@domainID, @convexID, @x, @y, @z, @l)
return @@identity
end

create procedure spDomainDelete( @domainID int)

--/H delete a domain and all its convexes and constraints
--/T Parameters:
--/T <li> domainID int      ID of the domain to be deleted
--/T <br>Sample call to delete a domain
--/T <br><samp>
--/T  <br> exec spDomainDelete @domainID
--/T  </samp>
--/T <br> see also spDomainNew, spDomainNewConvex, spDomainDrop,...

as
begin
delete HalfSpace where domainID = @domainID
delete Domain    where domainID = @domainID
return 0
end

if exists (select * from sysobjects where id = object_id(N'fDomainsContainPoint'))
    drop function fDomainsContainPoint
go
create function fDomainsContainPoint(@x float, @y float, @z float)
returns table as
-------------------------------------------------------------
--/H Return a table of all domains containing this point.
-------------------------------------------------------------
--/T Parameters:
--/T <li> x, y, z float  The 3-tuple the point.
--/T <li> returns a table of distinct domain IDs containing the point.
--/T <br>Sample call
--/T <br><samp>
--/T <br> select * from dbofDomainsContainPoint(@x,@y,@z)
--/T </samp>
--/T <br> see also spDomainNew, spDomainNewConvex, spDomainDrop,...
-------------------------------------------------------------
return (select  distinct domainID
from HalfSpace h
where (@x*x +  @y*y + @z*z) > l   -- in the area
group by all domainID, ConvexID
having count(*) = 0)
go
-------------------------------------------------------------
-- union two domains.
if exists (select * from sysobjects where id = object_id(N'spDomainOr'))
    drop procedure spDomainOr
go
create procedure spDomainOr (@d1 int, @d2 int,
    @type varchar(16),
    @comment varchar(8000) ) as
-------------------------------------------------------------
--/H Create a new domain containing the convexes of domains d1 and d2.
-------------------------------------------------------------
--/T The new domain will contain copies of the convexes of the two original domains.
--/T <br>Parameters:
--/T <li> d1 int       ID of the first domain.
--/T <li> d2 int       ID of the second domain.
--/T <li> type varchar(16)     short description of the domain (e.g. stripe)
--/T <li> comment varchar(8000) longer description of the domain (e.g. stripe details).
--/T <br> returns domainID int  the unique ID of the new domain.
--/T <br>Sample call get union two domains
--/T <br><samp>
--/T <br> exec @domainID = spDomainOr @d1, @d2, 'stripe', 'run 1 2 3'
--/T </samp>
--/T <br> see also spDomainNew, spDomainAnd, spDomainNot, spDomainDrop,...
-------------------------------------------------------------
begin
declare @newDomainID int
exec @newDomainID = spNewDomain @type, @comment
insert HalfSpace
select @newDomainID, ConvexID, x, y, z, l
from HalfSpace
where domainID = @d1
declare @convex2 int
select @convex2 = count(distinct convexID)+ 1
from HalfSpace
where domainID = @newDomainID
insert HalfSpace
select @newDomainID, ConvexID+@convex2, x, y, z, l
from HalfSpace
where domainID = @d2
return  @newDomainID
end
go
create procedure spDomainAnd ( @d1 int,
    @d2 int,
    @type varchar(16),
    @comment varchar(8000)
) as

begin
    declare @newDomainID int
    exec @newDomainID = spNewDomain @type, @comment

    declare @left  table (convexID int primary key, sequenceID  int identity(0,1))
    declare @right table (convexID int primary key, sequenceID  int identity(0,1))

    insert @left
    select distinct convexID
    from   HalfSpace
    where  domainID = @d1

    insert @right
    select  distinct convexID
    from   HalfSpace
    where  domainID = @d2

    declare @rightCount int
    select  @rightCount = count(*) from @right

    insert HalfSpace
    select @newDomainID, @rightCount*Left1.sequenceID + Right2.sequenceID, h1.x, h1.y, h1.z, h1.l   -- add in the constraints from the left
    from HalfSpace  h1,      -- the 2 half spaces
        @left as Left1, @right as Right2 -- the renumbering of the convexes
    where h1.domainID = @d1
    and h1.convexID = Left1.convexID

    insert HalfSpace
    select @newDomainID, @rightCount*Left1.sequenceID + Right2.sequenceID, h2.x, h2.y, h2.z, h2.l   -- add in the constraints from the right
    from HalfSpace  h2,      -- the 2 half spaces
        @left as Left1, @right as Right2 -- the renumbering of the convexes
    where h2.domainID = @d2
    and h2.convexID = Right2.convexID

    return  @newDomainID
end

go
-- Negative of a domain.
if exists (select * from sysobjects where id = object_id(N'fDomainNot'))
drop function fDomainNot

create function fDomainNot(@domain int, @convex int)
returns @us table(convexID int, planeID int, x float, y float, z float, l float) as
begin
--/H Recursive routine to create a new domain that is the complement (negation)
--/A of an existing domain
--/T <p> parameters:
--/T <li> @domain int,     -- ID of domain to be complemented
--/T <li> @convex int,     -- ID of convex, do the next convex AFTER this one
--/T <li> returns a table of convexes that complement the domain (convex+1...convexN)
--/T <li> convexID int,     -- 0...n ID of convexes in domain
--/T <li> planeID int,     -- ID of planes in convex 0,...p for each convex.
--/T <li> x,y,z,l float     -- the 4 floats defining the 1/2 space
--/T <br>
--/T Sample call:<br>
--/T <samp>
--/T <br> select * from dbo.fDomainNot(1,-1) -- -1 starts the recursion.
--/T </samp>
--/T <br>
--/T <br> by default there is a limit of 32 on the recursion depth,
--/T so at most 32 convexes can be negated.
begin
-- myConvex is the ID of this convex
-- we will take the cartesian product of it with the negative of the remaining convexes (if any)
-- then we will pick add each 1/2 space to each such negative convex.
declare @myConvex int
select @myConvex = min(convexID)
from HalfSpace
where domainID = @domain
and convexID > @convex

-- the ME table is a list of all the halfPlanes of this convex.
-- these halfPlanes have planeIDs 0,1,2...
-- so they are dense and zero based.
-- this is important for the math that follows.
declare @me table(planeID int identity(0,1),  x float, y float, z float, l float)
insert @me
select -x,-y,-z,-l
from HalfSpace
where domainID = @domain
and convexID = @myConvex

-- the planes is the count of 1/2 spaces in me
-- others is a count of 1/2 spaces in convexes after me.
declare @planes int
declare @others int
select @planes = count(*) from @me
select @others = count(convexID)
from HalfSpace
where domainID = @domain
and convexID > @myConvex

-- if there are no others, the recursion is at an end.
-- each 1/2 space is negated and makes a new convex in the answer
-- the edge ID becomes the convexID, and the edge id is 0.
-- notice that the convexIDs are dense 0,1,2... and edgeIDs start at 0
-- this is important to make the "convexID math work out.
if (@others = 0)
begin
insert @us (convexID, planeID, x, y, z, l)
select planeID, 0, x, y, z, l
end
from @me
end

else
-- if there are others, the then it is a two step process.
-- (1) recursively compute the negative domain of remaining convexes (call it him).
-- (2) cartesian product me x him
-- (3) insert each pair into the answer creating |me| x |him| edges
begin
  -- this table stores the cartesian product
  declare @Product table (mePlaneID int, meX float, meY float, meZ float, meL float, himConvexID int, himPlaneID int, himX float, himY float, himZ float, himL float)
  -- compute the cartesian product.
  insert @Product
  select me.*, him.*
  from @me as me,
       dbo.fNegateDomain(@domain, @myConvex) as him
  -- now insert each negative 1/2 space of my convex into each of his convexes
  -- as 1/2 space zero of the new convexes.
  -- note the "distinct" causes us to contribute this 1/2 space to only once to each
  -- new convex.
  insert @Product
  select distinct mePlaneID + @planes*himConvexID, 0, meX, meY, meZ, meL
  from @Product
  -- now insert all negative 1/2 space of his convexs into each new convex
  -- as 1/2 space 1,2,... of the new convexes (notice the +1).
  insert @Product
  select mePlaneID + @planes*himConvexID, himPlaneID+1, himX, himY, himZ, himL
  from @Product
  -- end of recursion.
end
-- return the answer.
return
go
if exists (select * from sysobjects where id = object_id(N'spDomainNot'))
    drop procedure spDomainNot

create procedure spDomainNot
    (@oldDomain int, -- ID of domain to be complemented
     @type varchar(16), -- type of domain (e.g. strip)
     @comment varchar(8000) -- describes the new domain
    )
    as
   -------------------------------------------------------------
    --/H  Create a new domain that is the complement (negation) of an existing domain
    --/A
    --/T <p> parameters:
    --/T <li> @oldDomain int, -- ID of domain to be complemented
    --/T <li> @type varchar(16), -- type of new domain: e.g. NotBox
    --/T <li> @comment varchar(8000) -- description of new domain: 'complement of a box at origin'
    --/T <li> returns ID of created domain.
    --/T <br>
    --/T Sample call:<br>
    --/T <samp>
    --/T  exec  @newDomainID = spNegateDomain @DomainID , 'NotBox, 'Complement of a 1x1x1 box at the origin'
    --/T </samp>
    --/T<br>
   -------------------------------------------------------------
    begin
    declare @newDomainID int -- ID of the new domain
    -- create the domain
    exec @newDomainID = spNewDomain @type, @comment

    -- insert complement convexes into the halfspace
    insert HalfSpace
        select @newDomainID, convexID, x, y, z, l
        from dbo.fNegateDomain(@oldDomain, -1) as negDomain
        -- return the new domain ID.
    return  @newDomainID
    end
    go
set nocount on
use tempdb
truncate table halfSpace
delete domain
go
declare @domainID int
declare @convexID int
declare @edgeID int
exec @domainID = spDomainNew 'spBox @ 0 & 2,2,2' , 'a 2x2x2 box at the origin and 2x2 box centered at 2,2,2'
exec @convexID = spDomainNewConvex @domainID
exec spDomainNewConvexConstraint @domainID, @convexID, -1, 0, 0, 1 -- bottom
exec spDomainNewConvexConstraint @domainID, @convexID, 1, 0, 0, 1 -- top
exec spDomainNewConvexConstraint @domainID, @convexID, 0, -1, 0, 1 -- back
exec spDomainNewConvexConstraint @domainID, @convexID, 0, 1, 0, 1 -- front
exec spDomainNewConvexConstraint @domainID, @convexID, 0, 0, -1, 1 -- left
exec spDomainNewConvexConstraint @domainID, @convexID, 0, 0, 1, 1 -- right
--insert Domain values (@domainID, 'box', 'a 2x2x2 box at (2,2,2)
exec @domainID = spDomainNew 'spBox @ -2' , 'a 2x2x2 box at the origin and centered at -2,0,0'
exec @convexID = spDomainNewConvex @domainID
exec spDomainNewConvexConstraint @domainID, @convexID, -1, 0, 0, -1 -- bottom
exec spDomainNewConvexConstraint @domainID, @convexID, 1, 0, 0, 3 -- top
exec spDomainNewConvexConstraint @domainID, @convexID, 0, -1, 0, -1 -- back
exec spDomainNewConvexConstraint @domainID, @convexID, 0, 1, 0, 3 -- front
exec spDomainNewConvexConstraint @domainID, @convexID, 0, 0, -1, -1 -- left
exec spDomainNewConvexConstraint @domainID, @convexID, 0, 0, 1, 3 -- right
-- A second domain (2x2x2 box centered at (-2, 0, 0)
exec @domainID = spDomainNew 'box at -2' , 'a 2x2x2 box at the origin and centered at -2,0,0'
exec @convexID = spDomainNewConvex @domainID
exec spDomainNewConvexConstraint @domainID, @convexID, -1, 0, 0, 3 -- bottom
exec spDomainNewConvexConstraint @domainID, @convexID, 1, 0, 0, -1 -- top
exec spDomainNewConvexConstraint @domainID, @convexID, 0, -1, 0, 1 -- back
exec spDomainNewConvexConstraint @domainID, @convexID, 0, 1, 0, 1 -- front
exec spDomainNewConvexConstraint @domainID, @convexID, 0, 0, -1, 1 -- left
exec spDomainNewConvexConstraint @domainID, @convexID, 0, 0, 1, 1 -- right
-- Create a set of points to test with.
create table points (cx float, cy float, cz float, hint char(32))
truncate table points
insert points values( 0,0,0,    'center')
insert points values( 0.5,0,0,  '1/2 up')
insert points values( -0.5,0,0, '1/2 down')
insert points values( 2,0,0,    '2 up')
insert points values( -2,0,0,   '2 down')
insert points values( .99,.99,.99, 'inside the corner')
insert points values( 1.001,1.001,1.001, 'into second box')
go
-- MORE TESTING HARNESS

-- code to find all the points inside any domain (along with the name of the domain.
-- a point could be in several domains.
-- should return all but "over the top" -- 6 out of 8, not 2 up or over the top.
select d.type, hint
from domain d,
    (select hint, h.domainID
     from Points p, HalfSpace h
     where (p.cx*h.x + p.cy*h.y + p.cz*h.z) > h.l   -- in the area
     group by all hint,domainID, ConvexID
     having count(*) = 0 )as c
where d.domainID = c.domainID
order by d.type

-- test the domain predicate.
select type, dbo.fDomainPredicate(domainID)
from domain

-- update the domain predicate = dbo.fDomainPredicate(domainID)
go

/* test the points in polygon predicates */
declare @command varchar(8000)
declare @domainID int
declare @type varchar(16)
declare  domain cursor read_only for select domainID, type from domain
open domain
while (1=1)
begin
    fetch next from domain into  @domainID, @type
    if (@@fetch_status <> 0) break
    select @command = 'select ''' + @type + ''' as domainName,' + 'hint as pointName
    from points where ' + predicate  from domain where domainID = @domainID
    print @command
    exec(@command)
end
close domain; deallocate domain

go
-- Speed test (1ms to get predicate, 6ms to do null compare, predicate runs at 1M
-- hapfspace/sec/cpu
create table t(a int)
declare @i int
set @i = 0
while (@i < 1000)
begin
    declare @command varchar(8000)
    select @command = 'select count(*) from points where ' + predicate  from domain where domainID = 1
    -- print @command
    insert t exec(@command)
    set @i = @i + 1
end
drop table t

go

-- test of the domain verbs
declare @cx float, @cy float, @cz float, @hint varchar(16)
declare points cursor read_only for select cx, cy, cz, hint  from points
open points
while (1=1)
begin
    fetch next from points into  @cx, @cy, @cz, @hint
    if (@@fetch_status <> 0) break
    select @hint as point, type as domainType, comment
    from domain, dbo.fDomainsContainPoint(@cx, @cy, @cz) as p
    where domain.domainID = p.domainID
end
close points; deallocate points
declare @domainID int
exec @domainID = spDomainOr 1, 2, '1+2', 'should be 3 convexes, 18 constraints'
select halfspace.*, domain.comment from domain, halfspace where domain.domainID = @domainID and halfspace.domainid = @domainID

declare @domainID int
exec @domainID = spAndDomain 1, 2, '1 and 2', 'should be 2 convexes, 24 constraints'
select halfspace.*, domain.comment from domain, halfspace where domain.domainID = @domainID and halfspace.domainid = @domainID

declare @domainID int
exec @domainID = spNegateDomain 2 ,'Not 2', 'should be 6 convexes, 6 constraints'
select halfspace.*, domain.comment from domain, halfspace where domain.domainID = @domainID and halfspace.domainid = @domainID

declare @domainID int
exec @domainID = spNegateDomain 1 ,'Not 1', 'should be 36 convexes, 72 constraints - all pairs of planes'
select halfspace.*, domain.comment from domain, halfspace where domain.domainID = @domainID and halfspace.domainid = @domainID

declare @domainID int
exec @domainID = spNegateDomain 11 ,'NotNot 2', 'should be 36 convexes, 72 constraints - all pairs of planes'
select halfspace.*, domain.comment from domain, halfspace where domain.domainID = @domainID and halfspace.domainid = @domainID