# 

# New Spatial Features in SQL Server Code-Named “Denali” Community Technology Preview 1

**SQL Server Technical Article**

**Writers:** Ed Katibah (Microsoft) and Milan Stojic (Microsoft)

**Technical Reviewers:** Michael Kallay (Microsoft), Marko Tintor (Microsoft) and Michael Rys (Microsoft)

**Published:** November 2010

**Applies to:** SQL Server Code-Named “Denali” Community Technology Preview 1 (CTP1)

**Summary:** SQL Server 2008 introduced spatial data support into the database server. This paper describes and discusses the new spatial features in SQL Server Code-Named “Denali” CTP1 that augment the existing SQL Server 2008 spatial functionality.

Copyright

This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This white paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in, or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

© 2010 Microsoft Corporation. All rights reserved.

Microsoft, Bing, and SQL Server are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

Contents

[Introduction 4](#_Toc276538041)

[Spatial Type Improvements 4](#_Toc276538042)

[Circular Arcs: Support for Three New Subtypes 4](#_Toc276538043)

[CircularString 5](#_Toc276538044)

[CompoundCurve 5](#_Toc276538045)

[CurvePolygon 6](#_Toc276538046)

[Collections of Curved Objects 6](#_Toc276538047)

[Existing Spatial Methods and Curved Objects 7](#_Toc276538048)

[New Methods for Circular Arcs 8](#_Toc276538049)

[New and Updated Methods and Aggregates for All Types 8](#_Toc276538050)

[Improved Precision 10](#_Toc276538051)

[Geography Type Enhancements 10](#_Toc276538052)

[FullGlobe: Support for Geography Objects Larger Than a Logical Hemisphere 10](#_Toc276538053)

[Other Considerations for the geography type 12](#_Toc276538054)

[New Methods for the geography Type 12](#_Toc276538055)

[Performance Improvements 13](#_Toc276538056)

[Spatial Index Improvements 13](#_Toc276538057)

[New Spatial Indexes: Geometry Auto Grid and Geography Auto Grid 13](#_Toc276538058)

[SPATIAL\_WINDOW\_MAX\_CELLS: New Spatial Index Hint 14](#_Toc276538059)

[Compression for Spatial Indexes 15](#_Toc276538060)

[Improved “Create Spatial Index” Build Time for Point Data 16](#_Toc276538061)

[New Nearest Neighbor Query Plan 16](#_Toc276538062)

[Other Performance Improvements 16](#_Toc276538063)

[Other Improvements 16](#_Toc276538064)

[Spatial Helper Stored Procedures 17](#_Toc276538065)

[General Engine Improvements That Affect Spatial Types 19](#_Toc276538066)

[Support for Persisted Computed Columns 19](#_Toc276538067)

[Improved CLR UDT Aggregate Performance 19](#_Toc276538068)

[Client-Side Library Changes 19](#_Toc276538069)

[Conclusion 20](#_Toc276538070)

# Introduction

This document covers the new additions in spatial data support, introduced in Microsoft SQL Server 2008, for the SQL Server Code-Named “Denali” Community Technology Preview 1 (CTP1) release. These new features are organized into the following sections:

* “Spatial Type Improvements”
* “Performance Improvements”
* “Other Spatial Improvements”

In order to fully digest this document the reader should be familiar with spatial data support in SQL Server 2008.

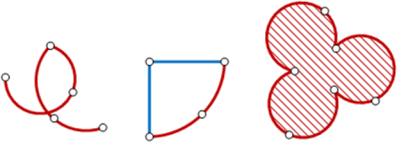
***Note****: The* ***Spatial results*** *tab in SQL Server Management Studio has not been updated to handle the new spatial features present in SQL Server Code-Named* *“Denali” CTP1. Consequently, the* ***Spatial results*** *tab has been disabled and is not currently available in SQL Server Management Studio for the SQL Server Code-Named* *“Denali” CTP1 build. You can, however, use SQL Server Management Studio from an existing installation based on SQL Server 2008 or SQL Server 2008 R2 to connect to an instance of SQL Server Code-Named* *“Denali.” The* ***Spatial results*** *tab is available for spatial query visualization, in this case, but it does not support some of the spatial features that are new with SQL Server Code-Named* *“Denali” CTP1, such as FullGlobe* ***geography*** *type support and circular arcs. Additionally, note that side-by-side installations of SQL Server Code-Named* *“Denali” CTP1 are not supported with existing SQL Server 2008 installations.*

# Spatial Type Improvements

SQL Server Code-Named “Denali” CTP1 introduces several significant enhancements to spatial types, such as support for new subtypes of circular arcs, new and updated methods and aggregates for all subtypes, improved precision, and updates to the geography type.

## Circular Arcs: Support for Three New Subtypes

Circular arcs are new to CTP1 and are based on the ISO SQL/MM, Part 3: Spatial standard. Circular arcs can be defined by themselves or they can be combined with line segments. Circular arcs can also be the basis for a new type of polygon that contains one or more curve components. For example, parallels on the globe (line of latitude) can be expressed as geography circular arcs. Sample circular arcs are illustrated here.



Circular arcs are supported by **geometry** and **geography** and can be defined using Well-known Text (WKT), Well-known Binary (WKB), and Geography Markup Language (GML) in SQL Server Code-Named “Denali.”

### CircularString

Circular strings are the basic curve subtype, corresponding to a LineString for linear data. Three points are used to define a segment with the start (first point) and end (third point) points and another point (second point) anywhere along the circular arc. Circular strings can be linked together where the last point of the previous curve becomes the first point of the next curve. Valid circular strings will always have an odd number of points, three or greater.

DECLARE @g GEOGRAPHY;  
SET @g = GEOGRAPHY::STGeomFromText('  
 **CIRCULARSTRING(0 -23.43778, 0 0, 0 23.43778)**  
)',4326);

### CompoundCurve

Compound curves enable you to define new curves that can be composed of circular strings only, or circular strings and linear strings. The end point of every component must be identical to the starting point of the next component.

Here is an example of a CompoundCurve made up of circular strings only.

DECLARE @g GEOGRAPHY;  
SET @g = GEOGRAPHY::STGeomFromText('  
 **COMPOUNDCURVE(  
 CIRCULARSTRING(0 -23.43778, 0 0, 0 23.43778),  
 CIRCULARSTRING(0 23.43778, -45 23.43778, -90 23.43778)  
 CIRCULARSTRING(-90 23.43778, -90 0, -90 -23.43778),  
 CIRCULARSTRING(-90 -23.43778, -45 -23.43778, 0 -23.43778)  
 )**',4326);

Here is an example of a CompoundCurve made up of circular strings and linear strings. Note that linear strings do not contain a keyword.

DECLARE @g GEOGRAPHY;  
SET @g = GEOGRAPHY::STGeomFromText('  
 **COMPOUNDCURVE(  
 (0 -23.43778, 0 23.43778),** --Linear Segment\* **CIRCULARSTRING(0 23.43778, -45 23.43778, -90 23.43778),   
 (-90 23.43778, -90 -23.43778),** --Linear Segment\* **CIRCULARSTRING(-90 -23.43778, -45 -23.43778, 0 -23.43778)  
 )**',4326);

*\*****Note****: You must remove the SQL comments (in green) from this example before it will execute correctly in Transact-SQL.*

### CurvePolygon

Curve polygons are similar to polygons, having at least one ring and zero or more holes (inner rings). Curve polygons are composed of linear strings, circular strings, and/or compound curves. Within a given ring, the first point as defined in a curve polygon component must be identical to the last point in a curve polygon component, just like standard polygon rings.

Here is an example of a CurvePolygon made up of compound curves, themselves made up of circular strings and linear strings.

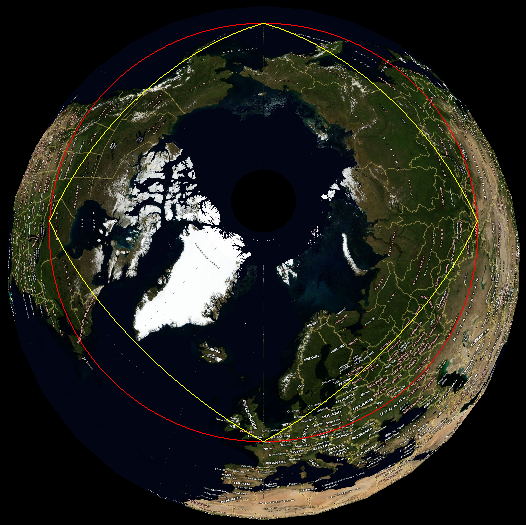
DECLARE @g GEOGRAPHY;  
SET @g = GEOGRAPHY::STGeomFromText('  
 **CURVEPOLYGON(  
 COMPOUNDCURVE(  
 (0 -23.43778, 0 23.43778),  
 CIRCULARSTRING(0 23.43778, -45 23.43778, -90 23.43778),   
 (-90 23.43778, -90 -23.43778),  
 CIRCULARSTRING(-90 -23.43778, -45 -23.43778, 0 -23.43778)  
 )**  
 )',4326);

### Collections of Curved Objects

In order to create a collection of curved objects with objects without curves, a geometry collection must be used.

DECLARE @g GEOGRAPHY  
SET @g = GEOGRAPHY::STGeomFromText('  
 **GEOMETRYCOLLECTION(  
 COMPOUNDCURVE(  
 CIRCULARSTRING(0 23.43778, -45 23.43778, -90 23.43778),  
 (-90 23.43778, -90 -23.43778)   
 ),   
 COMPOUNDCURVE(   
 CIRCULARSTRING(-90 -23.43778, -45 -23.43778, 0 -23.43778),   
 (0 -23.43778, 0 23.43778)   
 )**  
 )',4326);

The following illustration shows the difference between circular arcs and linear strings for the same set of coordinates.



CIRCULARSTRING(0 50, 90 50, 180 50, 270 50, 0 50) -- RED LINE

and

LINESTRING (0 50, 90 50, 180 50, 270 50, 0 50) –- YELLOW LINE

***Note****: The circular string defined in the previous example represents a circle on the globe, defined by five points in which the first and last coordinates are the same. You can create this feature as a curve polygon as follows.*

DECLARE @g GEOGRAPHY  
SET @g = GEOGRAPHY::STGeomFromText('  
 **CURVEPOLYGON(  
 CIRCULARSTRING(0 50, 90 50, 180 50, 270 50, 0 50)  
 )**',4326);

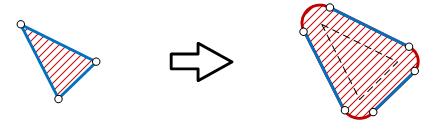
### Existing Spatial Methods and Curved Objects

All existing methods work on circular objects (STIntersects is illustrated here).



### New Methods for Circular Arcs

BufferWithCurves() uses circular arcs to construct buffered object with greatly reduced number of points compared to STBuffer().



STNumCurves() and STCurveN() are used for iteration through the list of the edges.

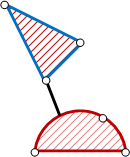
STCurveToLine() and CurveToLineWithTolerance() are used for approximating circular arcs with line segments within default and user specific tolerance.

***Note****: STCurveToLine and CurveToLineWithTolerance can be used to convert both LineStrings/MultiLineStrings and Polygons/MultiPolygons that have circular arc components to linear representations of the same dimensionality. This can use useful for:*

* *Spatial object representation in visualization programs*
* *Backward compatibility with earlier versions of SQL Server that have spatial data support*

## New and Updated Methods and Aggregates for Both Spatial Types

ShortestLineTo() returns a LineString that represents the shortest distance between two objects. The resulting LineString does not have to touch either parent object on a shape-defining vertex point.

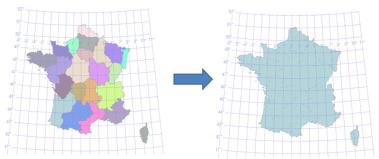


The followingaggregates are now available. They are exposed as static methods on **geography** and **geometry** types:

* UnionAggregate
* EnvelopeAggregate
* CollectionAggregate
* ConvexHullAggregate

For example, the following SELECT statement acts on and returns the data shown in the illustration.

SELECT GEOGRAPHY::UnionAggregate(geog) FROM Regions;



***Note****: The new aggregates are exposed in SQL Server only and are not exposed in the underlying spatial library.*

STLength() has been updated an now works on both valid and invalid LineStrings. This was done for LineStrings because the MakeValid method can remove overlapping parts. Typically a LineString is invalid due to overlapping segments, which may be caused by anomalies such as inaccurate GPS traces.

***Note****: STLength() does not remove overlapping/invalid segments. Additionally, STLength() includes overlapping and invalid segments in the length value it returns.*

MinDbCompatibilityLevel() is a new method is used for backward compatibility. It indicates whether spatial objects can be recognized by SQL Server 2008 and SQL Server 2008 R2.

DECLARE @C GEOMETRY  
SET @C = GEOMETRY::Parse('CIRCULARSTRING(0 50, 90 50, 180 50)')   
SELECT @C.**MinDbCompatibilityLevel()**  
--Result: 110

DECLARE @L GEOMETRY  
SET @L = GEOMETRY::Parse('LINESTRING (0 50, 90 50, 180 50)')  
SELECT @L.**MinDbCompatibilityLevel()**  
--Result: 100

***Note****: You can set the compatibility level of the database with the following Transact-SQL statement.*

--Set database compatibility level to SQL Server Code-Named "Denali"  
**ALTER DATABASE *database***   
**SET COMPATIBILITY\_LEVEL = 110**;

--Set database compatibility level to SQL Server 2008  
**ALTER DATABASE *database***  
**SET COMPATIBILITY\_LEVEL = 100**;

## Improved Precision

All constructions and relations are now done with 48 bits of precision, compared to 27 bits used in SQL Server 2008 and SQL Server 2008 R2. This can reduce the error caused by rounding of floating point coordinates for original vertex points by the internal computation.

For example, consider the following coordinate, which was processed using the STUnion() method in SQL Server 2008 but which was not involved in the resulting geometry.

Original Vertex Coordinate New Vertex Coordinate After Computation

82.339026 29.661245 82.339025999885052 29.662144999951124

In SQL Server Code-Named “Denali,” the greater numerical precision assists in the preservation of original coordinates of input points, in most cases. Here is the result of the same STUnion() method in SQL Server Code-Named “Denali,” shown earlier.

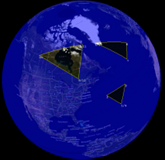
Original Vertex Coordinate New Vertex Coordinate After Computation

82.339026 29.661245 82.339026 29.662145

## Geography Type Enhancements

### FullGlobe: Support for Geography Objects Larger Than a Logical Hemisphere

The SQL Server Code-Named “Denali” Spatial Library supports objects that are larger than a logical hemisphere. Restricted to slightly less than a logical hemisphere in SQL Server 2008, spatial objects in SQL Server Code-Named “Denali” can now be as big as entire globe. A new type of object, called FULLGLOBE, can be constructed or received as a result of an operation. You can also construct objects that are “small holes” (see the following picture), because the interior for **geography** objects is defined by the orientation of its rings. There is no difference between exterior and interior rings in the **geography** data type.

 UNION  yields 

The first image shows an example of a “small hole” in a FullGlobe object, the second image shows a regular polygon, and the third image shows the union of the two, yielding a spatial object that covers the Earth with three small holes.

Here is a simple example that constructs a new FullGlobe object and executes a method on that object.

DECLARE @g GEOGRAPHY = GEOGRAPHY::STGeomFromText('**FULLGLOBE**',4326);  
SELECT @g.STArea()-- calculate the area of the WGS84 ellipsoid  
--Result: 510,065,621,710,996 meters squared

Vertex order is critical for the **geography** type. In SQL Server 2008 and SQL Server 2008 R2, if you enter an “incorrect” coordinate order for a geography polygon, you receive an error:

The specified input does not represent a valid geography instance because it exceeds a single hemisphere. Each geography instance must fit inside a single hemisphere. A common reason for this error is that a polygon has the wrong ring orientation.

For the SQL Server **geography** data type, ring order defined by the *left-foot rule*. The left-foot rule defines the interior region of the polygon (when you “walk” the boundary of a polygon, your left foot is always inside). Traditional outer ring/inner ring (hole) relationships can be modeled on the closed surface of the globe by using this definition.

***Note****: The term “left-foot rule” is used in deference to the term “left-hand rule,” which is already in use by physicists and mathematicians to describe phenomena other than polygon ring order.*  
The following examples demonstrate how the **geography** type deals with large (greater than a logical hemisphere) objects.  
  
Here is an example of a small (less than a logical hemisphere) polygon.

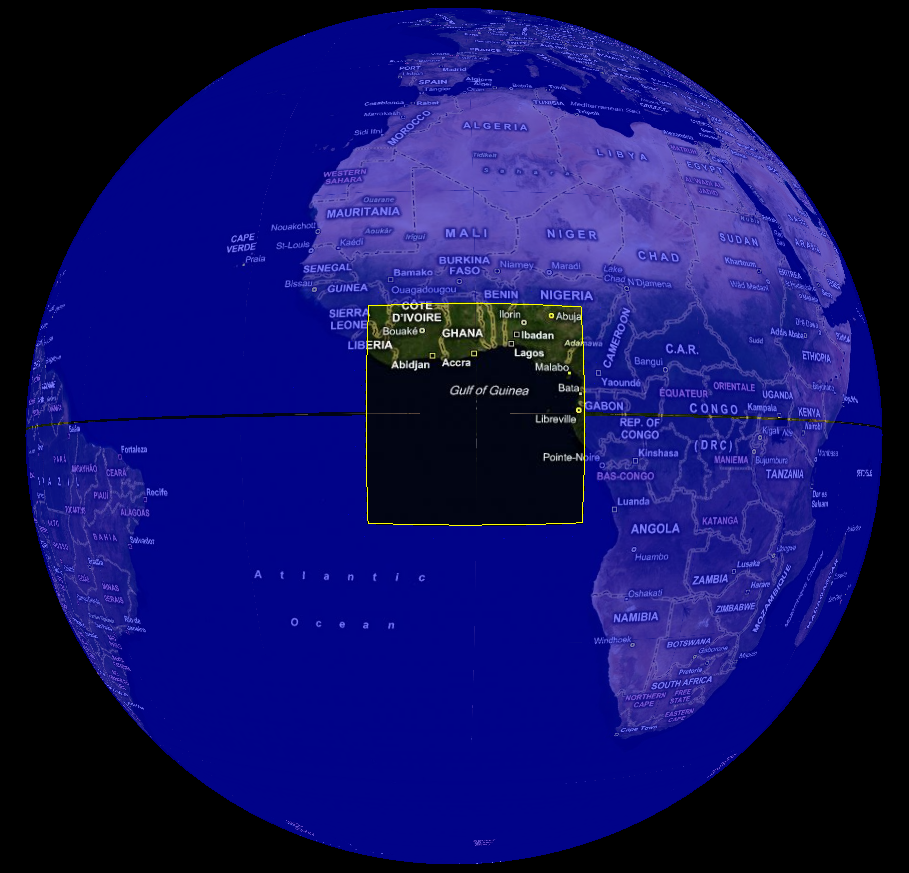
DECLARE @R GEOGRAPHY;   
SET @R = GEOGRAPHY::Parse('Polygon((-10 -10, 10 -10, 10 10, -10 10,   
 -10 -10))');  
SELECT @R;  
  
The resulting object can be visualized as follows.



The following Transact-SQL changes the coordinate order of the polygon ring to the opposite direction of the previous example.

DECLARE @R GEOGRAPHY;   
SET @R = GEOGRAPHY::Parse('Polygon((-10 -10, -10 10, 10 10, 10 -10,   
 -10 -10))');  
SELECT @R;

The resulting object is now the “rest of the globe.”



### Other Considerations for the geography type

There are several other considerations which need to be taken into account with the FullGlobe enhancements to the **geography** type:

* FullGlobe cannot be a member of a GeometryCollection
* WKT, WKB and GML (and its schema) now support FullGlobe objects
* Antipodal edges are not allowed (for example, Linestring(0 90, 0 -90)  is considered antipodal while LineString(0 90, 0 0, 0 -90) is not)
* There are now two types of arcs in **geography**: great circle arcs defined with two points and small circle arcs defined with three points
* Degenerate arcs will be treated as great circle arcs between the start and end points for the **geography** type (for the **geometry** type such arcs will be treated as straight edges)
* EnvelopeAngle() will return 180 for objects larger than a logical hemisphere and < 90 for smaller than a logical hemisphere objects

### New Methods for the geography Type

The **geography** type now allows invalid objects to be inserted into a table. The following new methods deal with such objects.

STIsValid() and MakeValid() allow invalid **geography** objects can be detected and corrected the same way as for the **geometry** type.

**Geography** polygons can now be accommodated without regard to ring orientation. This can lead to unintended behavior. The following new method allows such polygons to be corrected.

ReorientObject()can be used to reorient polygon rings for cases where they are constructed with the wrong orientation. LineStrings, curves and points will not be touched.

Here is an example.

DECLARE @R GEOGRAPHY = GEOGRAPHY::Parse('Polygon((-10 -10, -10 10, 10 10,   
 10 -10, -10 -10))');  
SELECT @R.**ReorientObject()**.STAsText();  
--Result: POLYGON ((10 10, -10 10, -10 -10, 10 -10, 10 10))

STWithin(), STContains(), STOverlaps(), and STConvexHull() methods have been added for the **geography** type, previously available only for the **geometry** type. With the exception of STConvexHull(), these new methods are supported by spatial indexes.

# Performance Improvements

SQL Server Code-Named “Denali” CTP1 includes features and enhancements that improve performance. These include new spatial indexes, a spatial index hint, compression, improved index build time, and a new query plan.

## Spatial Index Improvements

New spatial indexes provide further levels of tuning, and a new hint helps you fine-tune query performance.

### New Spatial Indexes: Geometry Auto Grid and Geography Auto Grid

A new auto grid spatial index is available for both spatial types (geometry\_auto\_grid and geography\_auto\_grid). The GRIDS parameter is no longer valid or needed when this index type is used. The new auto grid actually uses a different strategy to pick the right tradeoff between performance and efficiency. It uses eight levels (instead of four) for better approximation of objects of various sizes. The older style spatial index with four user-specified levels is still supported and is now referred to as a “manual grid” spatial index.

Here are some examples that use the new auto grid for the **geography** data type.

CREATE SPATIAL INDEX idxGeog  
 ON *table*(*column*)  
 USING **GEOGRAPHY\_AUTO\_GRID**;

or

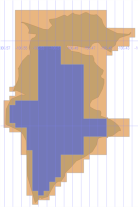
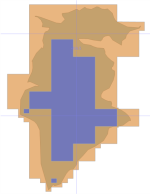
CREATE SPATIAL INDEX idxGeog  
 ON *table*(*column*)  
 USING **GEOGRAPHY\_AUTO\_GRID**  
 WITH (CELLS\_PER\_OBJECT = 32);

If an index type is not specified in the CREATE SPATIAL INDEX DDL, the appropriate auto grid for the specified spatial column data type is used by default. In the following example, if the spatial column data type is **geography**, the GEOGRAPHY\_AUTO\_GRID is used by default.

CREATE SPATIAL INDEX idxGeog

ON *table*(*column*);

The following figure illustrates spatial object approximation with 64, 128, and 256 cells\_per\_object, respectively, on the new eight-level auto grid. New auto grid spatial indexes have better continuous coverage and higher precision.



Spatial Object

Partially

contained

cells

Fully

contained

cells

The default number of cells per object for the new auto grid is 8 for **geometry** and 12 for **geography**. For original GEOMETRY\_GRID and GEOGRAPHY\_GRID indexes, the default for CELLS\_PER\_OBJECT remains the same as previous at 16.

***Note****: The CELLS\_PER\_OBJECT parameter is not used for point objects. It is used for all other objects, including multipoint objects.*

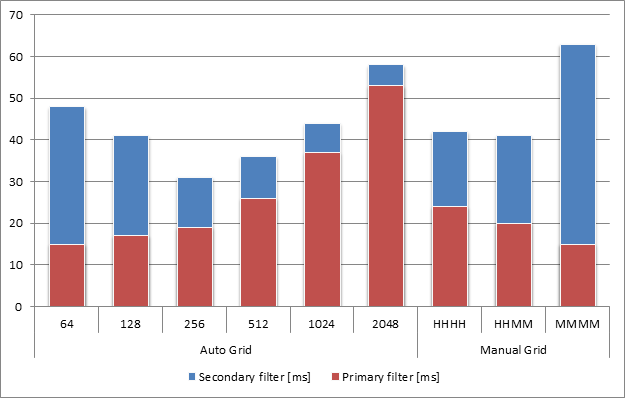
### SPATIAL\_WINDOW\_MAX\_CELLS: New Spatial Index Hint

This new spatial hint is critical for fine-tuning query performance using a spatial index. Dense spatial data often requires a higher SPATIAL\_WINDOW\_MAX\_CELLS, whereas sparse spatial data often demands a lower SPATIAL\_WINDOW\_MAX\_CELLS for optimum performance.

This hint can be used as follows.

SELECT \*  
 FROM table t with(**SPATIAL\_WINDOW\_MAX\_CELLS=1024**)  
 WHERE t.geom.STIntersects(@window)=1;

The default value is 512 for **geometry** and 768 for **geography**. Higher values make the spatial index more efficient, but the primary filter is slower. Lower values make the primary filter faster, but more time is spent in the secondary filter due to potentially poorer index efficiency, with the optimum somewhere in between. For example, in the following illustration, 256 cells is the optimum query window.



This chart shows query execution time (in milliseconds) compared to the old configurations (HHHH, HHMM, and MMMM in the three left-hand columns). Total times are combined primary filter times and secondary filter times. Making spatial indexes more precise improves overall performance until the tipping point is reached.

***Note****: For SQL Server 2008, this parameter was always set to 1024 and was not modifiable.*

### Compression for Spatial Indexes

Compression can be enabled on spatial indexes with Transact-SQL DDL.

CREATE SPATIAL INDEX idxGeom  
 ON *table*(*column*)  
 USING GEOGRAPHY\_GRID  
 WITH (  
**DATA\_COMPRESSION = *page|row***  
 );

On the basis of internal tests, spatial indexes with compression are 40-50 percent smaller, with a 5-10 percent performance overhead.

### Improved “Create Spatial Index” Build Time for Point Data

Spatial index build time for point data can be as much as roughly four to five times faster than it is under similar conditions in SQL Server 2008.

## New Nearest Neighbor Query Plan

A nearest neighbor query plan can be expressed as follows (the query elements highlighted in **red** are required for making the query optimizer process queries using a nearest neighbor query plan).

SELECT **TOP**(5) \*  
 FROM Restaurants r  
 WHERE r.type = ‘Italian’   
 AND r.position.**STDistance(**@me**) IS NOT NULL**  
 **ORDER BY** r.position.**STDistance**(@me);

It can also be expressed with limited maximum distance.

SELECT **TOP**(5) \*  
 FROM Restaurants r  
 WHERE r.type = ‘Italian’  
 AND r.position.**STDistance(**@me**) < @max\_range**  
 **ORDER BY** r.position.**STDistance**(@me);

This enhancement uses a spatial index to filter out rows and speed up query execution. With a spatial index, nearest neighbor execution time is up to two orders of magnitude faster (10 to 30 times) than a counterpart query in SQL Server 2008.

## Other Performance Improvements

The methods STIntersects(), STWithin(), STContains(), and STOverlaps() are now optimized when at least one of the operands is a point.

STDistance() performance between two points has been optimized.

STBuffer() has been optimized and improved for all cases. It’s faster and has lower memory footprint, especially for large and complex objects.

# Other Improvements

SQL Server Code-Named “Denali” also introduces other improvements to the way spatial data is handled. These improvements include two new stored procedures, general engine improvements, and client-side library changes.

## 

## Spatial Helper Stored Procedures

Two new helper stored procedures are available:

* **sp\_help\_spatial\_geography\_histogram**
* **sp\_help\_spatial\_geometry\_histogram**

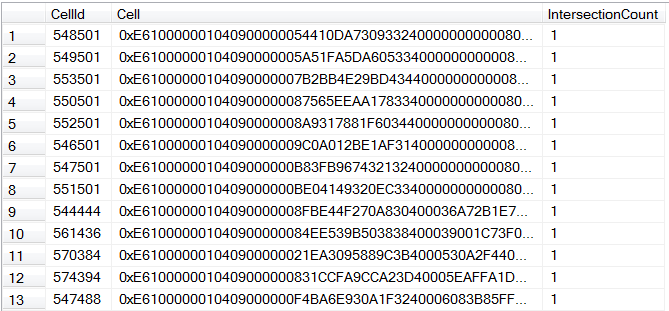
These routines can be used to evaluate the distribution of spatial data within a table over a given spatial column.

For example, the following query uses the **sp\_help\_spatial\_geography\_histogram** stored procedure to return information about the distribution of spatial data in the Mexico table for the geography column.

EXEC sp\_help\_spatial\_geography\_histogram Mexico,geography,1000,100

Arguments: <table name>, <spatial column>, <cell size>, <sample %>

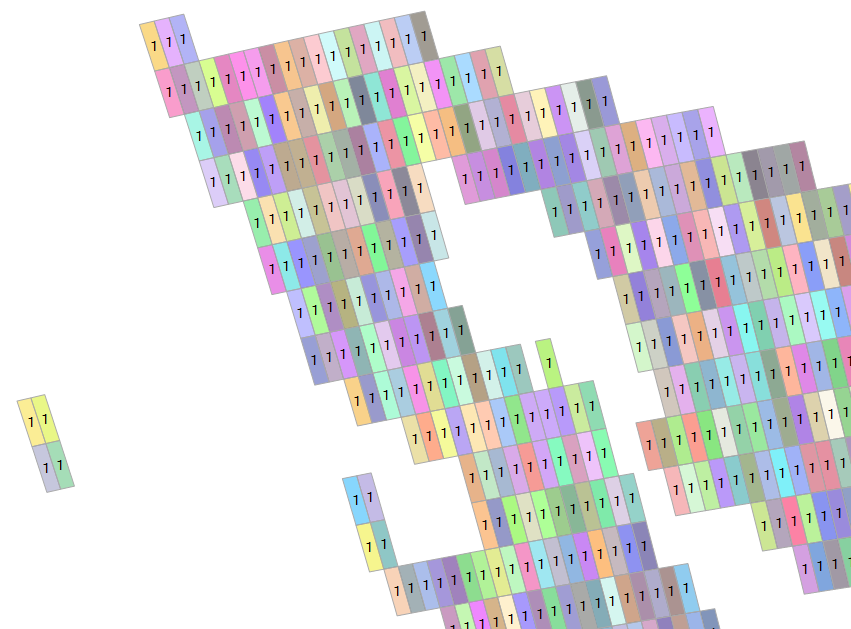
Here is sample output from the previous query.



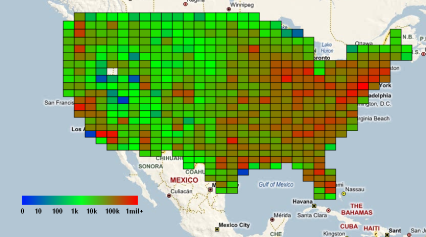
The output can be used for investigating spatial index efficiency or analyzing spatial data in general. The histograms can be also shown in the SQL Server Management Studio **Spatial results** tab.



The following is an SQL Server Management Studio-based illustration of the same display, showing greater detail (from the upper left-hand corner).



With some additional programming, data from these stored procedures can be used for external visualization, shown over Bing Maps.



## General Engine Improvements That Affect Spatial Types

SQL Server Code-Named “Denali” CTP1 includes support for persisted computed columns and improved CLR UDT performance.

### Support for Persisted Computed Columns

UDTs and spatial types can now be persisted in computed columns as shown in the following example.

CREATE TABLE location (  
 id int primary key,   
 x float(max),   
 y float(max),   
 range float(max),   
 **geom as**   
 **geometry::Point(x, y, 0)**  
 **.STBuffer(range) PERSISTED)**;

Improved CLR UDT Aggregate Performance  
CLR UDT aggregates in general (and by inference, new spatial aggregates) are improved and have much better performance. They are now 5 to 20 times faster compared to SQL Server 2008.

## Client-Side Library Changes

New sink interfaces, IGeometrySink110 and IGeographySink110, are available, and you should plan to use them in your future programming. They must be used for the following types of objects, which are incompatible with SQL Server 2008:

* Objects bigger than a logical hemisphere
* Objects with circular arcs

However, the old sinks will continue to work for SQL Server 2008 compatible objects.

Geometry and Geography builders (SqlGeometryBuilder and SqlGeographyBuilder) now support circular arc constructions.

A new method, Deserialize, has been added on the client library to both types. This method simplifies deserialization. The following code shows how to use it to deserialize SqlBytes.

using (SqlConnection con = new SqlConnection("..."))  
{  
 con.Open();  
 SqlCommand cmd = new SqlCommand("...", con);  
 SqlDataReader reader = cmd.ExecuteReader();  
 while (reader.Read())  
 {  
 SqlGeography g = SqlGeography.**Deserialize**(reader.GetSqlBytes(0));  
 Console.WriteLine(g)  
 }  
}

# Conclusion

The new spatial features in SQL Server Code-Named “Denali” CTP1 represent a significant milestone in the evolution of SQL Server spatial data support. The ability to support full globe spatial objects and circular arcs on the ellipsoid are industry firsts for relational database systems. The **geography** data type has achieved parity relative to the function and variety of methods that support the **geometry** data type. Overall performance, from spatial indexes to methods, has been significantly increased. These and other improvements to spatial promise a significant step forward in spatial prowess of the next generation of SQL Server.