Oracle’s Point Cloud datatype

Albert Godfrind
Geospatial and Multimedia Technologies
Oracle Corporation
Agenda

• A short history of Oracle Spatial technologies
• Point clouds and Databases: the challenges
• Separating the "physical" from the "logical"
• SDO_PC type structure
• Physical type: the point cloud blocks
• Loading point clouds
• Processing functions
• TINs: the SDO_TIN and tin blocks structures
• Generating TINs from point clouds
• Conclusion
Spatial Technologies at Oracle: A Long History

Oracle 7.1.6: Multi Dimension
- Points Only

Oracle 7.3.3: Spatial Data Option
- Points, Lines, Polygons
- Spatial Operators

Oracle 8.0.4: Spatial Cartridge
- Points, Lines, Polygons
- Spatial Operators
- 40% Faster Performance

Oracle 8i
- Spatial Data Type
- Spatial Operators
- Linear Referencing
- Coordinate Transformation
- Long transactions...

Oracle 9i
- Replication, partitioning
- Spatial aggregation
- Geodetic coordinates...

Oracle 10g
- Raster
- Topology
- Network modelling
- Spatial Mining
- MapViewer
- 3D support
- Solid modelling
- Web services

Oracle 11g
- 3D support
- Solid modelling
- Web services

Oracle 11g:
- Solid modelling
- Web services

Data Management Challenges of Point Clouds

• **Volume**: increasing LiDAR densities with technology – Billions of points
  - Multi-return 150 KHz * 4 = 600k/sec
  - Full-waveform 250 KHz * 2048 = 512,000,000/sec

• **But also**:
  - Lag-time from acquisition to analysis
  - Metadata access and management
  - Fusion with other geospatial data (terabytes)
  - Multi-user access and security
  - Versioning, Archiving
More Data Management Challenges

- Data Transformation
  - Surfaces (TIN, DEM, vector transformation)
- Projections
- Data integration
- Filtering, Visualization and Analysis
- Backup, Recovery and minimizing downtime
Process Flow

Las Files

Las loader

Point Cloud tables

Load point cloud

Flat files

Point Tables

Convert to Geometries

Query and Clip

Generate Tin

Tin tables

Query and Clip

Convert to Geometries
Storage Model for Point Clouds

• Separates logical from physical structures

• Logical structures
  • Tables containing an SDO_PC column
  • Contains generic attributes and footprint
  • Also contains a pointer to a PC block table

• Physical structures
  • “Block tables”
  • Contain point cloud blocks
  • Can be very large
  • Structure defined in SDO_PC_BLK object type
Storage Model

Logical structures
Contains point cloud metadata and footprint
Also contains pointers to one or more block tables

Physical structures
Point cloud block tables
Contain the points
Can be very large
Could be partitioned
Add new tables as necessary

pc 1 blocks
pc 2 blocks
pc 3 blocks
pc 4 blocks
pc 5 blocks
pc 6 blocks
Storage Model

PC table
- pc 1
- pc 2
- pc 3
- pc 4
- pc 5
- pc 6

PC blocks table 1
- pc 1 blocks
- pc 2 blocks
- pc 3 blocks

PC blocks table 2
- pc 4 blocks
- pc 5 blocks
- pc 6 blocks
Storage Model

PC table 1
- pc 1
- pc 2
- pc 3

PC table 2
- pc 4
- pc 5
- pc 6

PC blocks table
- pc 1 blocks
- pc 2 blocks
- pc 3 blocks
- pc 4 blocks
- pc 5 blocks
- pc 6 blocks
Creating Point Cloud Tables

- Use the SDO_PC type
- Can have any number of PC tables
- Include any combination of attributes
- Scenes can be searched on any attribute
- Also the spatial extent of the scene

```sql
CREATE TABLE LIDAR_SCENES(
    SCENE_ID           NUMBER PRIMARY KEY,
    COLLECTION_TS      TIMESTAMP,
    DESCRIPTION        CLOB,
    ... (any number of attributes) ...
    POINT_CLOUD        SDO_PC
);
```
The SDO_PC type

- PC_EXTENT is the footprint of the point cloud
- Needs a spatial index to support spatial searching

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE_TABLE</td>
<td>VARCHAR2 (70)</td>
</tr>
<tr>
<td>BASE_TABLE_COL</td>
<td>VARCHAR2 (1024)</td>
</tr>
<tr>
<td>PC_ID</td>
<td>NUMBER</td>
</tr>
<tr>
<td>BLK_TABLE</td>
<td>VARCHAR2 (70)</td>
</tr>
<tr>
<td>PTN_PARAMS</td>
<td>VARCHAR2 (1024)</td>
</tr>
<tr>
<td>PC_EXTENT</td>
<td>MDSYS.SDO_GEOMETRY</td>
</tr>
<tr>
<td>PC_TOL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>PC_TOT_DIMENSIONS</td>
<td>NUMBER</td>
</tr>
<tr>
<td>PC_DOMAIN</td>
<td>MDSYS.SDO_ORGSCL_TYPE</td>
</tr>
<tr>
<td>PC_VAL_ATTR_TABLES</td>
<td>MDSYS.SDO_STRING_ARRAY</td>
</tr>
<tr>
<td>PC_OTHER_ATTRS</td>
<td>SYS.XMLTYPE</td>
</tr>
</tbody>
</table>
The SDO_PC type

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE_TABLE</td>
<td>Name of the base table containing a column of type SDO_PC</td>
</tr>
<tr>
<td>BASE_TABLE_COL</td>
<td>Name of the column of type SDO_PC in the base table</td>
</tr>
<tr>
<td>PC_ID</td>
<td>ID number for the point cloud</td>
</tr>
<tr>
<td>BLK_TABLE</td>
<td>Name of the table that contains information about each block in the point cloud</td>
</tr>
<tr>
<td>PTN_PARAMS</td>
<td>Parameters for partitioning the point cloud</td>
</tr>
<tr>
<td>PC_EXTENT</td>
<td>SDO_GEOMETRY object representing the spatial extent of the point cloud (the minimum bounding object enclosing all objects in the point cloud)</td>
</tr>
<tr>
<td>PC_TOL</td>
<td>Tolerance value for points in the point cloud.</td>
</tr>
<tr>
<td>PC_TOT_DIMENSIONS</td>
<td>Total number of dimensions in the point cloud. Includes spatial dimensions and any nonspatial dimensions, up to a maximum total of 9.</td>
</tr>
<tr>
<td>PCDOMAINS</td>
<td>(Not currently used.)</td>
</tr>
<tr>
<td>PC_VAL_ATTR_TABLES</td>
<td>SDO_STRING_ARRAY object specifying the names of any value attribute tables for the point cloud</td>
</tr>
<tr>
<td>PC_OTHER_ATTRS</td>
<td>XMLTYPE object specifying any other attributes of the point cloud</td>
</tr>
</tbody>
</table>
Creating Point Cloud Block Tables

• Using the SDO_PC_BLK type

```sql
CREATE TABLE PC_BLK_01 OF SDO_PC_BLK (  
    PRIMARY KEY (  
        OBJ_ID, BLK_ID  
    )  
)  
LOB(POINTS) STORE AS SECUREFILE  
(COMPRESS HIGH NOCACHE NOLOGGING);
```

• Define a primary key on the block id
• Use SECUREFILE lobs (new structure in 11g)
• Allows compression of the LOBs!
  • (also encryption and de-duplication)
The SDO_PC_BLK type

- Describes one block of points

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJ_ID</td>
<td>NUMBER</td>
</tr>
<tr>
<td>BLK_ID</td>
<td>NUMBER</td>
</tr>
<tr>
<td>BLK_EXTENT</td>
<td>MDSYS.SDO_GEOMETRY</td>
</tr>
<tr>
<td>BLK_DOMAIN</td>
<td>MDSYS.SDO_ORGSCL_TYPE</td>
</tr>
<tr>
<td>PCBLK_MIN_RES</td>
<td>NUMBER</td>
</tr>
<tr>
<td>PCBLK_MAX_RES</td>
<td>NUMBER</td>
</tr>
<tr>
<td>NUM_POINTS</td>
<td>NUMBER</td>
</tr>
<tr>
<td>NUM_UNSORTED_POINTS</td>
<td>NUMBER</td>
</tr>
<tr>
<td>PT_SORT_DIM</td>
<td>NUMBER</td>
</tr>
<tr>
<td>POINTS</td>
<td>BLOB</td>
</tr>
</tbody>
</table>

- Contains the unique identifier of the block
  - Scene id (OBJ_ID, same as PC_ID) and block id (BLK_ID)
# The SDO_PC_BLK type

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJ_ID</td>
<td>ID number of the point cloud object</td>
</tr>
<tr>
<td>BLK_ID</td>
<td>ID number of the block.</td>
</tr>
<tr>
<td>BLK_EXTENT</td>
<td>Spatial extent of the block.</td>
</tr>
<tr>
<td>BLK_DOMAIN</td>
<td>(Not currently used.)</td>
</tr>
<tr>
<td>PCBLK_MIN_RES</td>
<td>Minimum resolution level at which the block is visible in a query. The block is retrieved only if the query window intersects the spatial extent of the block and if the minimum - maximum resolution interval of the query. Usually, lower values mean farther from the view point, and higher values mean closer to the view point.</td>
</tr>
<tr>
<td>PCBLK_MAX_RES</td>
<td>Maximum resolution level at which the block is visible in a query.</td>
</tr>
<tr>
<td>NUM_POINTS</td>
<td>Total number of points in the POINTS BLOB</td>
</tr>
<tr>
<td>NUM_UNSORTED_POINTS</td>
<td>Number of unsorted points in the POINTS BLOB</td>
</tr>
<tr>
<td>PT_SORT_DIM</td>
<td>Number of the dimension (1 for the first dimension, 2 for the second dimension, etc) on which the points are sorted.</td>
</tr>
<tr>
<td>POINTS</td>
<td>BLOB containing the points.</td>
</tr>
</tbody>
</table>
BLOB Structure

• The BLOB contains an array of points
• Each point encoded as
  • $d$ 64-bit floating point numbers ($d =$ the dimensionality of the point)
  • One 32 bit integer representing the point number
  • One 32 bit integer representing the partition number
• Future: compressed format
  • Storing coordinates as offsets from the origin of the block MBR
  • Using short integers
Initializing a Point Cloud

• Define the structure and organization of the point cloud
  • Resolution, dimensions, extent
  • Block capacity

• Specify the location of the blocks for each point cloud
  • Name of the point blocks table
  • Unique identifier in that table

```sql
INSERT INTO LIDAR_SCENES (SCENE_ID, POINT_CLOUD)
VALUES (1,
  SDO_PC_PKG.INIT(
    BASETABLE => 'LIDAR_SCENES',
    BASECOL => 'POINT_CLOUD',
    BLKTABLE => 'PC_BLK_01',
    PTN_PARAMS => 'BLK_CAPACITY=1000',
    PC_TOL => 0.005,
    PC_TOT_DIMENSIONS => 3,
    PC_EXTENT => SDO_GEOMETRY (2003, 4326, NULL,
      SDO_ELEM_INFO_ARRAY (1,1003,3),
      SDO_ORDINATE_ARRAY (-74, 40, -73, 41)
    )
  )
);```
Loading a Point Cloud

• Load a point cloud from a flat table

```
DECLARE
  PC  SDO_PC;
BEGIN
  SELECT POINT_CLOUD INTO PC
  FROM LIDAR_SCENES WHERE SCENE_ID = 1;
  SDO_PC_PKG.CREATE_PC (PC, 'INPUT_POINTS');
END;
/
```

• Structure of the input table

<table>
<thead>
<tr>
<th>RID</th>
<th>VARCHAR2(40)</th>
<th>Unique point identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_D1</td>
<td>NUMBER</td>
<td></td>
</tr>
<tr>
<td>VAL_D2</td>
<td>NUMBER</td>
<td></td>
</tr>
</tbody>
</table>

...
Loading a Point Cloud

- Input table could be a flat file
- Defined as an external table.

```
CREATE TABLE input_points (  
  rid             VARCHAR2(40),
  val_d1          NUMBER,
  val_d2          NUMBER,
  val_d3          NUMBER  
)  
ORGANIZATION EXTERNAL (  
  TYPE ORACLE_LOADER  
  DEFAULT DIRECTORY data_files  
  ACCESS PARAMETERS (  
    FIELDS TERMINATED BY "," (  
      rid,  
      val_d1,val_d2,val_d3  
    )  
  )  
  LOCATION ('input_points.dat')  
);
```

File “input_points.dat”

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>279</td>
<td>-73.99922</td>
<td>40.000002</td>
<td>74</td>
</tr>
<tr>
<td>280</td>
<td>-73.99921</td>
<td>40.000002</td>
<td>27</td>
</tr>
<tr>
<td>281</td>
<td>-73.99920</td>
<td>40.000002</td>
<td>76</td>
</tr>
<tr>
<td>282</td>
<td>-73.99919</td>
<td>40.000002</td>
<td>72</td>
</tr>
<tr>
<td>283</td>
<td>-73.99918</td>
<td>40.000002</td>
<td>91</td>
</tr>
<tr>
<td>284</td>
<td>-73.99917</td>
<td>40.000002</td>
<td>96</td>
</tr>
</tbody>
</table>
Getting LAS Data into Oracle

• Tests performed by Michael Smith and David Finnegan, US Army Corps or Engineers
• Presented at Oracle Spatial Users Conference, Tampa, Florida, April 23, 2009
Getting LAS Data into Oracle

- **Testing Machines:**
  - Sun t5240 2 UltraSparcT2+ 1.2Ghz - 64Gb Ram
  - Sun x4150 x86 2 Xeon X5460 3.16GHz - 8Gb Ram

- **Differences based on Chip Architecture**
  - x86 faster than Sparc

- **Speed of Temp location made a difference**
  - moving temp tablespace from SAS to SSD to RAM yielded ~20-40% increase in speed

- **Create PC without the results table**
  - not needed and saved ~20% time

- **Effect of Block Size**
  - small increase with larger size
Converting LAS Data to Points

- Java LAS loader to Normal Table
  - Sparc: 3.8 Mpts/min
  - x86: 22.7 Mpts/min
- External Table and LibLAS to Normal Table
  - new in 11.1.0.7, preprocessor option
  - PREPROCESSOR exec_dir:'las2txt'
  - OPTIONS '--parse Mxyz --stdout'
  - Sparc: 41.2 Mpts/min
  - x86: 99.96 Mpts/min
Creating Point Clouds

• Single Session
  • Sparc: 1.8 Mpts/min - 5k block size – ram temp
  • x86: 8.2 Mpts/min - 5k block size – ram temp
  • x86: 8.7 Mpts/min - 100K block size – ram temp

• Multiple Sessions
  • Used Apache Jmeter
  • Sparc:
    • 10, 20, 50, 100 simultaneous sessions
    • 1.2 – 2.5 Mpts/min
  • x86:
    • 6, 10, 20 sessions
    • 2.2 – 7.8 Mpts/min
Size Inflation!

- Original LAS File: 26 Mpts 505Mb
- Table and Index size:
  - Lobs (BasicFiles): 839.5 Mb
  - SecureFiles (no compression): 826.2 Mb
  - SecureFiles (medium compression): 223.4 Mb
Processing Point Clouds

- Select scenes using spatial operators
- Use any spatial operator to search through point blocks
  - SDO_ANYINTERACT
  - SDO_NN
  - SDO_FILTER
Processing Point Clouds

• CLIP_PC (Clip Point Cloud)
  • 2D or 3D query window
  • Returns points for any block whose extent intersects the query window
  • Only points that intersect the query window are returned
  • Creates a new SDO_PC, can be stored or used in queries

• TO_GEOMETRY
  • Gets the points (as a Point Cluster) from a PC
  • Can be from a CLIP_PC operation
**CLIP_PC: Clipping from a Point Cloud**

- Selects points from a point cloud that are within a spatial window.
- Can also select points based on specific dimension values.
- Results in an array of point blocks.

```sql
DECLARE
    PC SDO_PC;
BEGIN
    -- Get the scene to clip from
    SELECT POINT_CLOUD INTO PC
    FROM LIDAR_SCENES
    WHERE SCENE_ID = 1;

    -- Clip out the desired subset from the scene
    INSERT INTO CLIPPED_LIDAR_SCENES_BLOCKS
    SELECT * FROM TABLE (SDO_PC_PKG.CLIP_PC (INP => PC,
        IND_DIM_QRY => SDO_GEOMETRY(2003, 4326, NULL,
            SDO_ELEM_INFO_ARRAY (1, 1003, 3),
            SDO_ORDINATE_ARRAY (-73.99996, 40.000066,
                              -73.99994, 40.000080
            ),
            OTHER_DIM_QRY => NULL,
            QRY_MIN_RES => NULL,
            QRY_MAX_RES => NULL
        )
    )
END;
/```
Retrieving Point Cloud Data

- 1 km circular buffer moving in data range
- Calculate average Z value, max Z value
- Sparc:
  - 30 sessions – avg: 8.46 sec / session
  - 300 sessions – avg: 8.64 sec / session
- x86:
  - 30 sessions – avg: 0.66 sec / session
  - 250 sessions – avg: 1.14 sec / session
Triangulated Irregular Networks (TINs)
SDO_TIN: Triangulated Irregular Network

- Representation of surfaces / terrains
- Contains a network of irregularly placed triangles
- Each point (triangle node) has X, Y and Z coordinates

<table>
<thead>
<tr>
<th>Node No</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>
Storage Model

• Separates logical from physical structures

• Logical structures
  • Tables containing an SDO_TIN column
  • Contains metadata and footprint
  • Also contains a pointer to a TIN block table

• Physical structures
  • “Block tables”
  • Contain triangles
  • Can be very large
  • Structure defined in SDO_TIN_BLK object type
Storage Model

TIN table
- tin 1
- tin 2
- tin 3
- tin 4
- tin 5
- tin 6

TIN blocks table
- tin 1 blocks
- tin 2 blocks
- tin 3 blocks
- tin 4 blocks
- tin 5 blocks
- tin 6 blocks
Storage Model

TIN table
- tin 1
- tin 2
- tin 3
- tin 4
- tin 5
- tin 6

TIN blocks table 1
- tin 1 blocks
- tin 2 blocks
- tin 3 blocks

TIN blocks table 2
- tin 4 blocks
- tin 5 blocks
- tin 6 blocks
Storage Model

TIN table 1
- tin 1
- tin 2
- tin 3

TIN blocks table
- tin 1 blocks
- tin 2 blocks
- tin 3 blocks

TIN table 2
- tin 4
- tin 5
- tin 6

TIN blocks table
- tin 4 blocks
- tin 5 blocks
- tin 6 blocks
Creating TIN Tables

- Creating a TIN table

```
CREATE TABLE TINS (  
  ID NUMBER PRIMARY KEY,  
  TIN SDO_TIN
);  
```

- Creating a TIN block table

```
CREATE TABLE TIN_BLOCKS_01 OF SDO_TIN_BLK (  
  PRIMARY KEY (  
    OBJ_ID, BLK_ID  
  )  
)  
LOB(POINTS) STORE AS SECUREFILE (NOCACHE NOLOGGING)  
LOB(TRIANGLES) STORE AS SECUREFILE (NOCACHE NOLOGGING);  
```
The SDO_TIN type

- TIN_EXTENT is the footprint of the TIN
- May need a spatial index

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE_TABLE</td>
<td>VARCHAR2 (70)</td>
</tr>
<tr>
<td>BASE_TABLE_COL</td>
<td>VARCHAR2 (1024)</td>
</tr>
<tr>
<td>TIN_ID</td>
<td>NUMBER</td>
</tr>
<tr>
<td>BLK_TABLE</td>
<td>VARCHAR2 (70)</td>
</tr>
<tr>
<td>PTN_PARAMS</td>
<td>VARCHAR2 (1024)</td>
</tr>
<tr>
<td>TIN_EXTENT</td>
<td>MDSYS.SDO_GEOMETRY</td>
</tr>
<tr>
<td>TIN_TOL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>TIN_TOT_DIMENSIONS</td>
<td>NUMBER</td>
</tr>
<tr>
<td>TIN_DOMAIN</td>
<td>MDSYS.SDO_ORGSCL_TYPE</td>
</tr>
<tr>
<td>TIN_BREAK_LINES</td>
<td>MDSYS.SDO_GEOMETRY</td>
</tr>
<tr>
<td>TIN_STOP_LINES</td>
<td>MDSYS.SDO_GEOMETRY</td>
</tr>
<tr>
<td>TIN_VOID_RGNS</td>
<td>MDSYS.SDO_GEOMETRY</td>
</tr>
<tr>
<td>TIN_VAL_ATTR_TABLES</td>
<td>MDSYS.SDO_STRING_ARRAY</td>
</tr>
<tr>
<td>TIN_OTHER_ATTRS</td>
<td>SYS.XMLTYPE</td>
</tr>
</tbody>
</table>
The SDO_TIN_BLK type

- Describes one TIN block
- Contains a LOB with the points in that block.
- Contains another LOB with the triangles.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJ_ID</td>
<td>NUMBER</td>
</tr>
<tr>
<td>BLK_ID</td>
<td>NUMBER</td>
</tr>
<tr>
<td>BLK_EXTENT</td>
<td>MDSYS.SDO_GEOMETRY</td>
</tr>
<tr>
<td>BLK_DOMAIN</td>
<td>MDSYS.SDO_ORGSCL_TYPE</td>
</tr>
<tr>
<td>PCBLK_MIN_RES</td>
<td>NUMBER</td>
</tr>
<tr>
<td>PCBLK_MAX_RES</td>
<td>NUMBER</td>
</tr>
<tr>
<td>NUM_POINTS</td>
<td>NUMBER</td>
</tr>
<tr>
<td>NUM_UNSORTED_POINTS</td>
<td>NUMBER</td>
</tr>
<tr>
<td>PT_SORT_DIM</td>
<td>NUMBER</td>
</tr>
<tr>
<td>POINTS</td>
<td>BLOB</td>
</tr>
<tr>
<td>TR_LVL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>TR_RES</td>
<td>NUMBER</td>
</tr>
<tr>
<td>NUM_TRIANGLES</td>
<td>NUMBER</td>
</tr>
<tr>
<td>TR_SORT_DIM</td>
<td>NUMBER</td>
</tr>
<tr>
<td>TRIANGLES</td>
<td>BLOB</td>
</tr>
</tbody>
</table>
BLOB Structure

- The **POINTS** blob contains an array of points
- Each point encoded as
  - $d$ 64-bit floating point numbers ($d$ = the dimensionality of the point)
  - One 32 bit integer representing the point number
  - One 32 bit integer representing the partition number
- The **TRIANGLE** blob contains an array of triangles
  - Each triangle defined by 3 points
  - Points identified by their number in the POINTS blob
Package *SDO_TIN_PKG*

- Initialization of a TIN
  - `sdo_tin_pkg.init()`
- Loading a TIN from a point cloud
  - `sdo_tin_pkg.create_tin()`
- Extracting triangles in a spatial window
  - `sdo_tin_pkg.clip_tin()`
- Converting to SDO_GEOMETRY
  - `sdo_tin_pkg.to_geometry()`
Enhancements Under Consideration

- Refined granular access to SDO_PC
  - Create PC without needing input table
  - Insert/Update new Blocks of PC
  - Increase maximum dimensionality
- Enable query /update / indexing of values beyond spatial in PC
Integration with standard RDBMS features

• Moving massive amounts of LIDAR data between databases:
  • Use Transportable Tablespaces

• Storage control and scalability
  • Use ASM (Automatic Storage Management)
  • Dynamically add and use disk capacity

• Cluster and grid computing (RAC)
  • Dynamically add and remove processing nodes
  • Parallel processing and parallel queries

• Exadata Database Machines
  • Offload queries and I/Os to dedicated hardware
Open Source Enhancements

• Development of LibLAS library
  • Write SDO_PC from LAS
  • Write LAS from SDO_PC
  • Extend LibLAS to read other formats
    • TerraSolid .bin files
    • Others ..... 
  • Encourage 3rd party developers to make use of library

• Coordination with GDAL/OGR
  • LibLAS can currently write to OGR data types
  • GDAL can read/write SDO_RASTER
  • Enable GDAL to read from LibLAS as an OGR type
ALBERT GODFRIND
Spatial Solutions Architect
Geospatial and Multimedia Technologies
Data Server Division

Oracle Corporation
Greenside
400 av. Roumanille - BP 309
06906 Sophia-Antipolis
France

phone    +33 4 93.00.88.91
fax       +33 4 93.00.88.44
mobile    +33 6 09.97.27.23
albert.godfrind@oracle.com
http://www.oracle.com/