"Google Summer of Code 2007 Project"

PGRaster – A Coverage/Raster Model and Operations for PostGIS

Participant: Xing Lin (solo.lin@gmail.com)

Mentor: Timothy H. Keitt (tkeitt@gmail.com)

Last Update: 2007/07/06

Copyright and Trademarks

PostgreSQL is released under the BSD license.

For More information about PostgreSQL project, please refer to its website of http://www.postgresql.org

PostGIS adds support for geographic objects to the PostgreSQL object-relational database. In effect, PostGIS "spatially enables" the PostgreSQL server, allowing it to be used as a backend spatial database for geographic information systems (GIS), much like ESRI's SDE or Oracle's Spatial extension. PostGIS follows the OpenGIS "Simple Features Specification for SQL" and has been certified as compliant with the "Types and Functions" profile.

PostGIS has been developed by Refractions Research as a project in open source spatial database technology. PostGIS is released under the GNU General Public License. We continue to develop PostGIS, and have added user interface tools, basic topology support, data validation, coordinate transformation, programming APIs and much more. Our list of future projects includes full topology support, raster support, networks and routing, three dimensional surfaces, curves and splines and other features. Ask us about consulting services and implementing new features.

For more information about PostGIS project, please refer to its website of http://www.postgis.org.

PROJ.4 - Cartographic Projections Library

PROJ.4 has been placed under an MIT license. I believe this to be as close as possible to public domain while satisfying those who say that a copyright notice is required in some countries. The COPYING file read as follows:

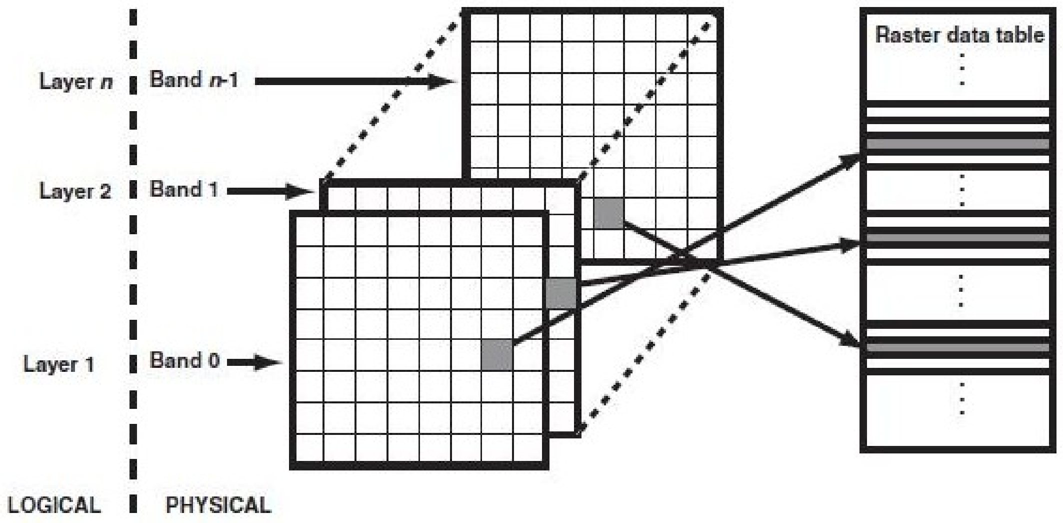
All source, data files and other contents of the PROJ.4 package are available under the following terms. Note that the PROJ 4.3 and earlier was "public domain" as is common with US government work, but apparently this is not a well defined legal term in many countries. I am placing everything under the following MIT style license because I believe it is effectively the same as public domain, allowing anyone to use the code as they wish, including making proprietary derivatives.

Though I have put my own name as copyright holder, I don't mean to imply I did the work. Essentially all work was done by Gerald Evenden.

0. General Ideas of Design

The proposed PostGIS PGRaster will use a generic raster data model that is component-based, logically layered, and multidimensional. The design is heavily influenced by Oracle®'s GeoRaster component. A new object type named PGRASTER will be defined as well as some other useful object types. A table with a column of type PGRASTER is called a PGRaster table. Each row in the PGRaster table denotes an geo-referenced images (satellite images, aerial photos...) or other raster coverage (DTM/DEM...). Other columns could be defined as needed. Functions will be defined and executed upon objects of type PGRASTER, such as the functions to create subset of raster coverage, the functions to interpolate a point within a certain raster coverage. All such new object types and functions are executable from SQL language.

The real raster (or image) data is stored in another toasted table that is connected to its corresponding PGRASTER object. Such tables are called Raster table, everyone of which contains special column of type RASTER to store raster data. Within a single RASTER object, there are a sequence of image pixels or coverage cells, each of which represent the smallest unit of information within a raster dataset.

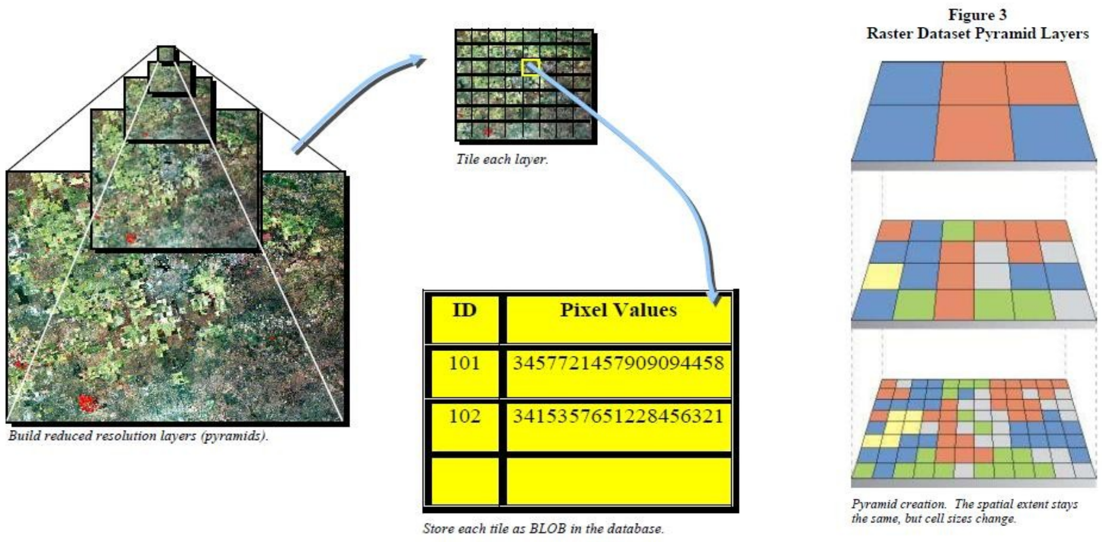


(PGRaster, Bands, and Raster data table. Source: Oracle® GeoRaster)

Images and raster dataset will be saved in tiles/blocks. It means the entire images or raster dataset will be divided into several smaller tiles of regular size before imported into image database. Each row in Raster table will only store one block/tile of raster data. Other columns will be required together with the RASTER column. This information will be used to help record the location of each blocks, so that they could be rebuilt as a whole when necessary. For multidimensional satellites images as well as RGB images, the blocking/tiling techniques could also be applied to the band dimension. The parameters of blocking storage could be defined by users. By default, it will be used with an proper set of parameters advised by PGRaster image database itself automatically.

In addition to reducing the data amount transferred over network, tiling/blocking could also be used to improve the performance of data visualization and analysis. The entire image data could be display onto the viewport of client window asynchronously tile by tile. It could save the time of response and improve user experience than show up a big final image after a long time waiting. Data analysis process could also be improved by optimizing certain algorithms to make use of the tiling storage system and multiple processors' environment.

Another technique that is widely used to improve the performance of large image browsing is called pyramid structure. Pyramid structures are built from "copy" datasets, each one resampled at a coarser resolution. These coarser copies of raster datasets will be much smaller in size than the original one but adequate for visualization at a lower scale (zoom ratio). Imaging you have a 400pixel*400pixel viewport in the client and a 800pixel*800pixel image in the database. You can just display a half-reduced version of the original image, which will appear nearly the same as you put all the pixels to a small viewport. But it will save 75% of data transfering from sever to client. When the user zoom in to a different scale, a special level of image in the pyramid structure with a proper accuracy will be packed, transferred to and rebuild at the client side. Pyramid structure could obviously improve the performance of the large image browsing via Internet application. The parameters that control the arrangement and storage of pyramid structures could be altered by user. By default, it will be used with a set of optimized parameters that are advised by PGRaster image database itself automatically. Blocking/tiling techniques will also be apply to pyramid levels also.

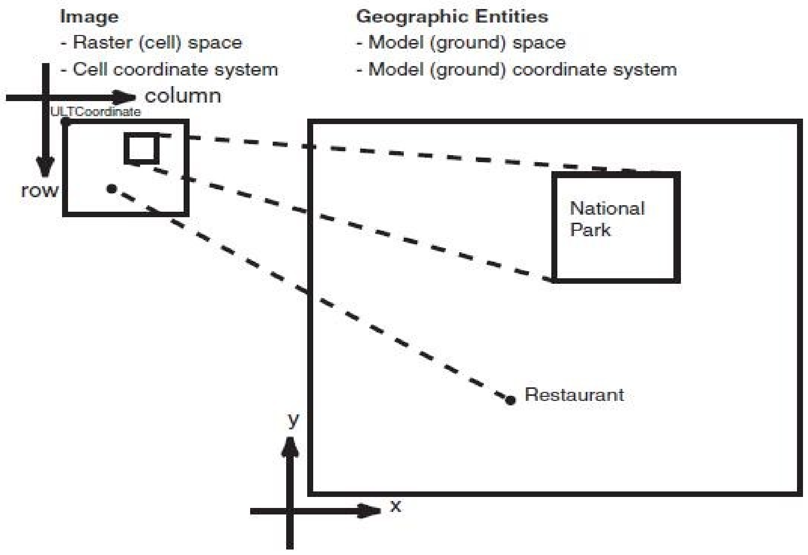

Tile each layer.
ID
Pixel Values
101
3457721457909094458
102
3415357651228456321
Build reduced resolution layers (pyramids).
Store each tile as BLOB in the database.
Figure 3
Raster Dataset Pyramid Layers
Pyramid creation. The spatial extent stays the same, but cell sizes change.

(Pictures source from ESRI® ArcSDE® 9.1 Raster. Should I contact ESRI for permission?)

Geo-reference is the thing that differs GIS raster/coverage datasets from ordinary image files. It tells the location of raster image within a geographic coordinate system, projected coordinate system or local coordinate system. The relationships between cell coordinates and model coordinates are modeled by PGRaster reference systems (mapping schemes). Similarly as Oracle® GeoRaster, the following reference systems are defined in PostGIS PGRaster:

- Spatial reference system, also called PGRaster SRS, which maps cell coordinates (row,column,vertical) to model coordinates (X,Y,Z). Using the spatial reference system with PGRaster data is referred to as georeferencing the data.PostGIS PGRaster SRS will include a specified spatial reference system in SRID (in PostGIS, a spatial spatial referenced system will be integrated with PostGIS SRID systems which is derived from the PROJ project), as well as the mapping schema between cell/pixel coordinate system and ground coordinate systems (or local coordinate systems). GCP or parameters for affine transformation could be applied to record such informations. More details about the spatial reference system will be discussed later (Georeferencing is discussed in Section 3).

- Temporal reference system, also called PGRaster TRS, which maps cell coordinates (temporal) to model coordinates (T).
- Band reference system, also called PGRaster BRS, which maps cell coordinates (band) to model coordinates (S, for Spectral).



(Mapping between ULTCoordinates and Model Coordinates. Source: Oracle® GeoRaster)

Initially, the PGRaster SRS will be implemented by setting up a mapping from cell coordinates system (up-left coordinate, ULTCoordinate) to model coordinates (geographic or projected). PGRaster currently supports six-parameter affine transformation that geo-references two-dimensional raster data. Such affline transformation will be recorded by the six-parameters as well as ground control point (GCP) in a special tables (in Oracle® GeoRaster terms, such tables are called value attribute table, VAT). There are also some other aspects about georeferencing of raster dataset, such as rectification and orthorectification. A special object of type GEOSRS will be set up to record such information for a raster dataset and be stored as part of metadata for PGRASTER objects.

Data compression and decompression will be one of the key techniques to be used in PostGIS PGRaster image database. Using compression before data storage could reduce storage space and amount of data to transfer via network. But it depends on the types of data to be compressed and kinds of application. Currently, PGRaster will provide two native data compression algorithm to reduce data storage space: JPEG2000 (Lossy) and LZW(Lossless). Some raster data, such as DEM/DTM data, could be used lossless data compression algorithm, because it is always involved in a data analysis process required the high accuracy of data. But sometimes for satellite images and aerial photos, the lossy compression algorithms could be compelling which could have a acceptable visualization effect, but highly reduce the data amount. This could be especially true for pyramid data. Of course, you can choose not to do any compression or decompression staff towards the raster dataset. The compression information will be recorded as part of PGRASTER object.

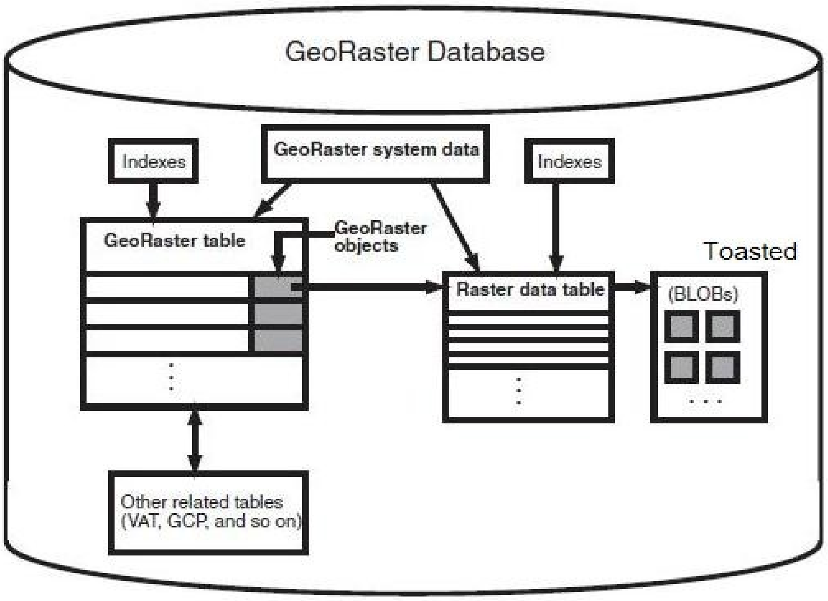
Indexing raster image is based on the spatial footprint of each block of raster data which will be stored as a GEOMETRY object in cell coordinates space. Certain kinds of spatial index (GiST-R Tree) will be applied on such GEOMETRY column to indexing raster images. When certain subset of the entire image is needed or user is navigating throughout the whole dataset, involved blocks of raster data could be easily located and selected out with the help of spatial indexing tied to the blocks.

As required by PostgreSQL user-extended data type schema, certain types of input and output functions need to be provided together with the definition of new object types. Because the specialty of raster data model (RAW, cell cloud....), the similar way to WKT and WKB will be invented to help the manual input of raster data (sometimes, people need to create some constant, blank, template, or filter raster dataset.) Common raster data encoding techniques will be adopted here, such as RLE (Run-Length Encoding), QTE(Quad-Tree Encoding) and so on. Some import and export tools will also be provided to handle the translation between PostGIS PGRaster and the famous raster data format (ESRI GRD/ASCII, ERDAS IMAGE, GEOTIFF and so on).

As well PostGIS the spatial extension for PostgreSQL, PostGIS PGRaster also need to have some system tables as data dictionary and place to save metadata. Certain system tables will be established after introduction new data types into PGRaster. Trigger and stored procedures will be set up to maintain the data integrity when inserting, modifying or deleting data from PGRaster table (you need to delete corresponding data records in Raster data table).

There are some other parts regarding the physical storage solution for raster data in ORDBMS, such as data type, value type, band interleaving (BSQ, BIP, or BIL), data padding, raster data encoding and so on. More detail information will be included in the following parts. For more informations, please refer to the corresponding section below.

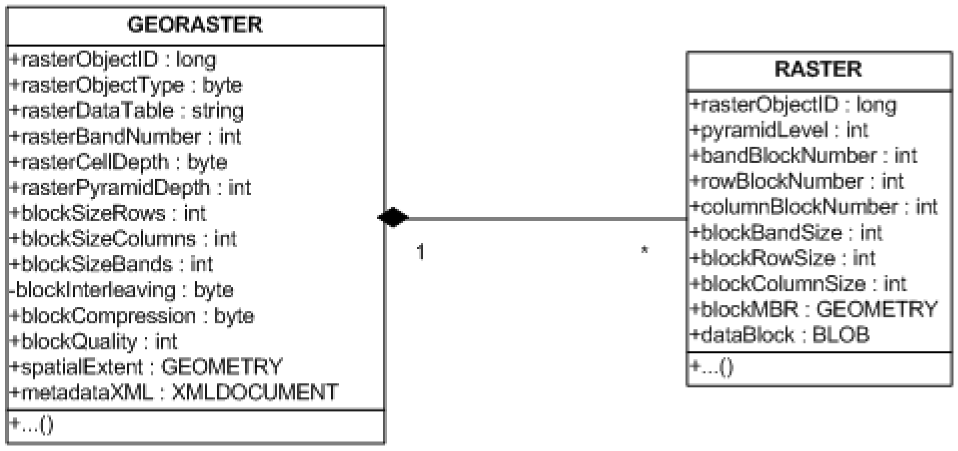
Finally, the raster dataset will be storage and processed within PostgreSQL/PostGIS ORDBMS in such as schema.



(Source: Oracle® GeoRaster. Modified to be proper within PostgreSQL environment)

1. User-Defined Object Types (1): PGRASTER & RASTER

As mentioned in the section of Introduction, the PGRaster extension for PostGIS will mainly include two new object types: the `PGRASTER` class which represents a whole raster dataset in a real application, and the `RASTER` class which will handle the storage of raster data in blocks. The whole raster dataset / image, viewed as a PGRASTER object, is stored as blocks/tiles within an external table, each row of such table denotes a RASTER object. There is a 1-to-many relationship between the PGRASTER object and RASTER object. The following is the main class diagram for the PGRaster extension of PostGIS.



6.1 PGRASTER Definition

PGRASTER object denotes a whole image/raster dataset for a certain region, such as New York city of the United States. You can have a table with a PGRASTER column (we call it PGRaster table in PostGIS PGRaster system) to store all the satellites for the whole country. Each row of this table has a instance of PGRASTER object, telling the remote sensing image for a certain city. Of course, you can have some other columns together such as:

```
CREATE TABLE us_images
{
      CityName TEXT,
      Image PGRASTER
};
```

Here you save all the geo-referenced satellite images in the column of *Image* which is in type of PGRASTER.

The object type of PGRASTER is defined as follow and more details about the components of this new user-defined object typs to PostGIS.

```
typedef struct _georaster
{
      long        rasterObjectID,
   /* byte        rasterObjectType, */
      byte        rasterDimensions,
      byte        rasterBandType,
      byte        rasterDataType,
      byte        rasterValueType,
      string      rasterDataTable,
      int         rasterBandCount,
      int         rasterCellDepth,
      byte        rasterPyramidEnabled,
      int         rasterPyramidDepth,
      int         blockSizeRows,
      int         blockSizeColumns,
      int         blockSizeBands,
      byte        blockPadding,
      byte        blockbandInterleaving,
      byte        blockCompression,
      int         blockQuality,
      double      nodataValue,
      GEOR_SRS    rasterSRS,
      GEOR_STA    rasterStatistics,
      GEOMETRY    spatialExtent,
   /* XMLDocument metadataXML * TBD */
} PGRASTER;
```

- `rasterObjectID` – This attributes is used to identify the unique `PGRASTER` object with PostGIS PGRaster systems. The data type of this attribute is long (serial???) and an special function will be invented to help generate a unique ID for the newly created `PGRASTER` object. Another alternative might be just set the `rasterObjectID` to be serial integers. This `rasterObjectID` will be used to referred related `PGRASTER` object in raster data table.
- `rasterDimensions` – An attribute in byte that tells the number of dimensions of this raster dataset. Currently, only the two-dimensional raster dataset is supported in PostGIS PGRaster. But in the future, it could be extended to support higher dimensional coverage, such as 3D raster object and even 4D raster object. Currently, the `rasterDimensions` could only be 2.
- `rasterBandType` – BandType means how the information in represented in bands. A enumerated data will be recorded in `rasterBandType` as a byte attribute. Currently, the following enumerated items are supported: BT_SINGLE=0, BT_RGB=1, BT_RGBA=2, BT_MULTI=3, and BT_OTHERS=4. Among them,
  - BT_GREY means there is only one band (white/black, or greyscale image) with the dataset;
  - BT_RGB means the 24 bits colorful image. In this type, the each cell of 3 channels of image data can only be stored as a whole in the blocking physical storage. More information will be included in the discussion of data type.
  - BT_RGBA means the 32 bits colorful image. In this type, the each cell of 3 channels of image data can only be stored as a whole in the blocking physical storage. More information will be included in the discussion of data type.

  - BT_MULTI means a multi-sepctral remote sensing dataset which could hold more than one multi-spectral bands of images.
  - BT_OTHERS means a unknown band type is set.

  Here the first three type are set for the optimization of image storage and processing which is mostly for visualization only.

- `rasterDataType` – means the type of data cell used to store the information in each geographic pixel in the raster dataset or image. The data type for recording raster cell information should be coincide with the `rasterCellDepth`. Currently, we support the following data type for raster cell data:
  - DT_1BIT = 0 which mean a boolean data unit and has a cell depth of 1.
  - DT_2BIT = 1 which mean a 2 bits unsigned integer and has a cell depth of 2.
  - DT_4BIT = 2 which mean a 4 bits unsigned integer and has a cell depth of 4.
  - DT_8BIT_U = 3 which mean a 8 bits unsigned byte with a cell depth of 8.
  - DT_8BIT_S = 4 which mean a 8 bits signed byte and has a cell depth of 8.
  - DT_16BIT_U = 5 which mean a 16 bits unsigned integer (short) and has a cell depth of 16.
  - DT_16BIT_S = 6 which mean a 16 bits signed integer (short) and has a cell depth of 16.
  - DT_32BIT_U = 7 which mean a 32 bits unsigned integer (long) and has a cell depth of 32.
  - DT_32BIT_S = 8 which mean a 32 bits signed integer (long) and has a cell depth of 16.
  - DT_24BIT_RGB = 9 which mean a 24 bits RGB pixel and has a cell depth of 24. *
  - DT_32BIT_RGBA = 10 which mean a 32 bits RGBA pixel and has a cell depth of 32. **
  - DT_32BIT_REAL = 11 which mean a 32 bits single floating point number (float) and has a cell depth of 32.
  - DT_64BIT_REAL = 12 which mean a 64 bits double floating point number (double) and has a cell depth of 64.

  (*) and (**) are set separately for the optimization of geo-referenced image data that are purely for visualization. In such situation, each pixel of a image will be treated as a whole in contrast to multi-spectral remote sensing datasets.

- `rasterValueType` – Types of value or referred as scale of measurement set up a limitation on how the raster data could be interpreted, processed or represented in a real application. In GIS, the following value types are usual.
  - VT_NOMINAL - a qualitative, non-numerical and non-ranking scale that classifies features on intrinsic characteristics. For example, in a land use classification scheme, polygons can be classified as industrial, commercial, residential, agricultural, public and institutional.
  - VT_ORDINAL - a nominal scale with ranking which differentiates features according to a particular order. For example, in a land use classification scheme, residential land can be denoted as low density, medium density and high density
  - VT_INTERVAL - an ordinal scale with ranking based on numerical values that are recorded with reference to an arbitrary datum. For example, temperature readings in degrees

    centigrade are measured with reference to an arbitrary zero (i.e. zero degree temperature does not mean no temperature)

  - VT_RATIO - an interval scale with ranking based on numerical values that are measured with reference to an absolute datum. For example, rainfall data are recorded in mm with reference to an absolute zero (i.e. zero mm rainfall mean no rainfall)
  - VT_IMAGE – a special value type that is set up for the optimization of image data that is purely for geo-visualization purpose.

- `rasterDataTable` – refers to the name of data table in PostgreSQL/PostGIS that contains the blocking data of this `PGRASTER` object.
- `rasterBandCount` – refers to the number of bands in this raster dataset. Normally, except for multi-spectral remote sensing dataset, the `rasterBandCount` will always equal to one (including RGB and RGBA images).
- `rasterCellDepth` – means how long in bits of data are used to record the information within this geo-referenced cell.
- `rasterPyramidDepth` – refers the number of pyramid levels available for this `PGRASTER` data. The pyramid levels are named from 0 to Depth-1. If this PyramidDepth equal to 1, it means that there is no Pyramid Structure available for visualization optimization. This information can't not be set by end-used. The PostGIS PGRaster will provide a specific set of functions to build, update or remove pyramid structures for a `PGRASTER` object.
- `blockSizeRows` – refers the size of block along the row dimension. This size of block apply to all data block stored in raster data table except the ones near the lower edge of cell space. By default, the block size will be (128,128,B), which means each block will store data of all bands within a region of 128 pixel * 128 pixel.
- `blockSizeColumns` – refers the size of block along the column dimension. This size of block apply to all data block stored in raster data table except the ones near the right edge of cell space. By default, the block size will be (128,128,B), which means each block will store data of all bands within a region of 128 pixel * 128 pixel.
- blockSizeBands - refers the size of block along the band dimension. This size of block apply to all data block stored in raster data table except the ones near the last bands. By default, the block size will be (128,128,B), which means each block will store data of all bands within a region of 128 pixel * 128 pixel. It means the blockSizeBands will equal to the bands count by default.
- `blockPadding` – whether to use data padding for the blocks near the image edges or not.
- `blockBandInterleaving` – Since data of multi-bands are stored together within a single block or file, it is very common to take the interleaving techniques to arrange the data of each bands. As a enumerated data, `blockBandInterleaving` records how the data of multi-bands images/raster dataest are arranged within the blocks. Three of them are popular:
  - BI_BSQ = 0 : band sequential. For example, the three bands remote sensing, if taking the BSQ band interleaving schema, the whole image data of first band will store first, then the second band, and finally the third band.

- BI_BIL = 1 : band interleaved by line. For example, the three bands remote sensing, if taking the BIL band interleaving schema, the 1st line of image data in first band will store first, then the 1st line of second band, and finally the 1st line of the third band. Then, the 2nd line of the first band, the 2nd of the second band and finally the 2nd line of the third band. Do it the same as band interleaved by line until all the data has been recorded.
- BI_BIP = 2 : band interleaved by pixel. For example, the three bands remote sensing, if taking the BIP band interleaving schema, the 1st pixel of image data in first band will store first, then the 1st pixel of second band, and finally the 1st pixel of the third band. Then, the 2nd pixel of the first band, the 2nd pixel of the second band and finally the 2nd pixel of the third band. Do it the same as band interleaved by pixel until all the data has been recorded. The pixel is following a sequence of from up-to-down and left-to-right throughout the cell space.

By default, the BI_BSQ will be taken which any user specified blockBandInterleaving information.

- `blockCompression` – denotes the attribute that records the data compression method to compress/decompress the blocked data in each row of raster data table. Currently, both lossy and losses compression methods are supported as following:
  - JPEG-B = 0 (I don't know whether I could find a good open-source JPEG2000 library.)
  - JPEG-F = 1 (I don't know whether I could find a good open-source JPEG2000 library.)
  - LZW (LZ77) = 2
  - NONE = 3

  By default, the lossless LZW/LZ77 compression method is taken. Because none raster data could be compressed/decompressed in a lossy way except for those image files which are merely for geo-visualization purpose. Users can also choose to save the uncompressed raster data. What's more, all the functions that could operated upon decompressed (uncompressed) data could also act on compressed ones because PostGIS PGRaster will decompress the data if necessary and then carry out the operation required. For more information, please refer to the following section for details.

  The JPEG2000 algorithms could only be applied to raster datasets that have one, three and four bands. The other dataset should choose the LZW/LZ77 for compression if needed. The JPGE2000 is especially useful for image data.

- `blockQuality` – If JPEG2000 algorithm is specified in the `blockCompression` section, you can also specify a quality index from 0 to 100 for it.
- `spatialExtent` – In PostGIS PGRaster, only regular extent of raster dataset is supported. Here the `spatialExtent` in `GEOMETRY` type is used to store the spatial extent (rectangle) in the ground coordinate systems or local coordinate systems. If this `PGRASTER` object has not been geo-referenced yet, this attribute will be set to NULL. More information about the PGRaster spatial reference system (SRS) will be introduced later.
- `NodataValue` – Nodata value is always used to pad those cells that have no valid information there, but for regular storage schema you need a value to be a placeholder. Nodata value is always choosed to be the unusual one beyond the normal range of raster data, but have

the same cell depth as normal data (sometime it is impossible to do so). If padding techniques is used in the blocked data, the `NodataValue` is used for padding.

- `rasterSRS` – rasterSRS attribute is used to stored the spatial referenced information for the `PGRASTER` object. It is an instance of new user defined object type called `GEOR_SRS`. More about the `GEOR_SRS` could be found in the following sections.
- `rasterStatistics` – the attribute used to recorded the pre-computed statistical data of the `PGRASTER` object after data import for the sake of performance optimization. Such statistical data could be of great help while carrying out some special functions upon `PGRASTER` objects. It is recorded in a new user-define data types named `GEOR_STA`. More about the `GEOR_STA` could be found in the following sections.

`PGRASTER` object type will be implemented as a user-defined object type into PostgreSQL as well as PostGIS spatial type of `GEOMETRY`.

6.2 RASTER Definition

Physically, the data of `PGRASTER` is blocked and stored in an external table call Raster table of type `RASTER`. Each row in the Raster table denotes an instance of blocking data. There could be more than one Raster table connected to a PGRaster table.

The `RASTER` object type as well as the Raster table is defined as follow and more details about each column in the Raster table will come after the definition.

```
Typedef struct _raster
{
      long        rasterObjectID,
      int         pyramidLevel,
      int         bandBlockNumber,
      int         rowBlockNumber,
      int         columnBlockNumber,
      int         blockBandSize,
      int         blockRowSize,
      int         blockColumnSize,
      GEOMETRY    blockMBR,
      BLOB        dataBlock
} RASTER;
```

- `rasterObjectID` – the raster object id to be referred as the hosting `PGRASTER` object in PGRaster table.
- `pyramidLevel` – denotes level in the pyramid structure for current block of data.
- `bandBlockNumber` – denotes the number of block along the band dimension.
- `rowBlockNumber` - denotes the number of block along the row dimension.
- `columnBlockNumber` – denotes the number of block along the column dimension.

- `blockBandSize` – denotes the actual block size along the band dimension. Normally, it will equal to the standard block size in the band dimension. But at the corners of cell space (row, column, band), it might be less than the standard block size recorded in the `PGRASTER` object.
- `blockRowSize` - denotes the actual block size along the row dimension. Normally, it will equal to the standard block size in the row dimension. But at the corners of cell space (row, column, band), it might be less than the standard block size recorded in the `PGRASTER` object.
- `blockColumnSize` - denotes the actual block size along the column dimension. Normally, it will equal to the standard block size in the column dimension. But at the corners of cell space (row, column, band), it might be less than the standard block size recorded in the `PGRASTER` object.
- `blockMBR` - records the minimal bounding rectangle (MBR) in type of `GEOMETRY` for each block of data in the cell coordinate system (ULTCoordinate). All the coordinates in this MBR geometry will be integers with a SRID of -1 (not spatial reference system is specified). This attribute will be used to index the blocked raster data using spatial indexes, such as GiST-R in PostgreSQL/PostGIS.
- `dataBlock` – a BLOB (or bytea in PostgreSQL) column to store the actual blocked raster data. It is toasted storage approach for the data. More details about the physical storage schema and blocking techniques will be illustrated in the following sections.

Among these attributes, the primary key will be defined as `{rasterObjectID, pyramidLevel, bandBlockNumber, rowBlockNumber, columnBlockNumber}`. A spatial index will also be established on the field of `blockMBR.`

2. User Defined Object Types (2)

(To Be Finished.......)

In addition to `PGRASTER` and `RASTER object` types, there are some other object types useful for end users to understand and handle the data stored in PostGIS PGRaster. In PostGIS PGRaster, these object types will also be defined and supported as followings.

2.1. GEOPIXEL

(coming soon...)

2.2.PIXEL

(coming soon...)

2.3.COLORPALETTE (Pseudo Color Table, PCT)

(coming soon...)

2.4.HISTOGRAM

(coming soon...)

2.5.GEOR_STA

(coming soon...)

2.6.GEOR_SRS: PGRaster Spatial Referenced System

(This object type and its storage will be included in the next seciton.)

3. Georeferencing

3.1.Geo-Referencing method for PostGIS PGRaster

As the same approach to Oracle® GeoRaster, the PostGIS PGRaster uses the low order polynomials when geo-referencing image or raster dataset according to the following six-parameter affine transformation formulas:

```
row = a + b * x + c * y
col = d + e * x + f * y
```

In these formulas:

- row = Row index of the cell in raster space
- col = Column index of the cell in raster space
- x = East-West position of the point on the ground or in model space.
- y = North-South position of the point on the ground or in model space.
- a, b, c, d, e, and f are coefficients, and they are stored in the SRS metadata.
- b*f – c*e should not be equal to 0 (zero).

In the formulas, if b = 0, f = 0, c = -e, and both c and e are not 0 (zero), the raster data is rectified, and the formula becomes:

```
row = a + c * y
col = d - c * x
```

This is the simplest case for georeferencing when the horizontal and vertical axises are parallel to the axises of ground coordinate system.

In the current release, the cellRepresentationType value must be UNDEFINED. In other words, a cell is just a scalar value (or an element of an array) without any shape defined in its cell space. However, when the PGRaster object is georeferenced, each cell covers a specific square or rectangular area or represents a point of this area in the model space. In the cell space, each cell has an integer coordinate. Through georeferencing, the cell's integer coordinate can be transformed into model coordinates, which identify an exact location of a point. This point or model coordinate may be either the upper-left corner or the center of the area represented by the cell in the model space.

If the original georeferencing information in the source data is in the inverse direction, such as in an ESRI world file, the transformation formulas are the following:

```
x = A * col + B * row + C
y = D * col + E * row + F
```

In this case, the preceding A, B, C, D, E, and F coefficients that you specify to the GEOR.georeference procedure are automatically adjusted internally to produce the correct georeferencing result: a, b, c, d, e, and f coefficients.

The model coordinates have the same unit as that of the specified SRID and should be in the value range defined by the model coordinate system. For example, if the PGRaster object is georeferenced to a geodetic coordinate system such as 8307, the unit of the model coordinates derived from the SRS must be decimal degrees, and Georeferencing 1-18 Oracle® Spatial GeoRaster values should be in the ranges of -180.0 to +180.0 or 0.0 to 360.0 for longitude and -90.0 to +90.0 for latitude. In this case, you cannot use meters or other unit.

3.2.GEOR_SRS Object type

As the same approach and schema to Oracle® GeoRaster, the PostGIS PGRaster will use a special object type to store the spatial referencing information for a concrete `PGRASTER` object. The `GEOR_SRS` object is defined as followings:

```
typedef struct _geor_srs
{
      boolean     isReferenced,
      boolean     isRectified,
      boolean     isOrthoRectified,
      long        SRID,
      double      spatialResolutionX,
      double      spatialResolutionY,
      double      spatialResolutionZ,
      double      spatialTolerance,
      byte        coordLocation,
      double      rowOff,
      double      columnOff,
      double      heightOff,
      double      xOff,
      double      yOff,
      double      zOff,
      double      rowScale,
      double      columnScale,
      double      heightScale,
      double      xScale,
      double      yScale,
      double      zScale,
      double      rowRMS,
      double      columnRMS,
      double      totalRMS,
      double[]    rowNumberator,
      double[]    rowDenominator,
      double[]    columnNumerator,
      double[]    columnDenominator,
} GEOR_SRS;
```

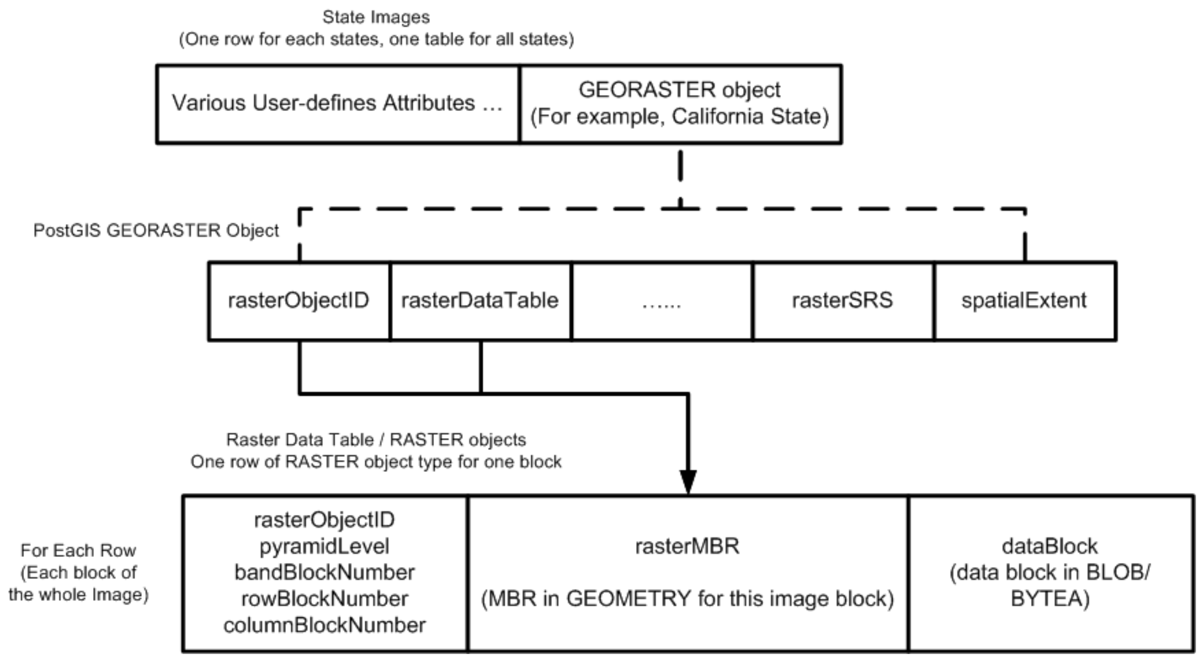
The followings comes the detail explanation for each attributes of the `GEOR_SRS` objects.

- `isReferenced` – TRUE if the PGRaster object is georeferenced; FALSE if the PGRaster object is not georeferenced.
- `isRectified` – TRUE if the PGRaster object is both georectified and georeferenced; FALSE if the PGRaster object is not georectified.
- `IsOrthoRectified` – TRUE if the PGRaster object is orthorectified, georectified, and georeferenced; FALSE if the PGRaster object is not orthorectified.
- `SRID` – SRID value of the model (ground) coordinate system.
- `spatialResolutionX` – spatial resolution along the X/Column dimension. The unit of resolution is derived from the ground coordinate system (SRID).
- `SpatialResolutionY` – spatial resolution along the Y/Row dimension. The unit of resolution is derived from the ground coordinate system (SRID).
- `SpatialResolutionZ` – spatial resolution along the Z/Height dimension. The unit of resolution is derived from the ground coordinate system (SRID). This field is reserved for 3D raster dataset and currently not being used.
- `spatialTolerance` – Tolerance value.
- `coordLocation` – A enumerated data that tells the model coordinate location representing either the upper-left corner or the center of each cell in the model space when cell coordinates (integer numbers) are converted to model coordinates (double numbers). Two values are supported, CL_CENTER=0, and CL_UPPERLEFT=1.
- `rowOff` – Reserved for future use regarding the offset along the row dimension of raster dataset in the cell coordinate system. Must be 0 (zero) for the current release.
- `ColumnOff` – Reserved for future use regarding the offset along the column dimension of raster dataset in the cell coordinate system. Must be 0 (zero) for the current release.
- `HeightOff` – Reserved for future use regarding the offset along the height dimension of 3D raster dataset in the cell coordinate system. Must be 0 (zero) for the current release.
- `xoff` – Must be 0 (zero) for the current release.
- `Yoff` – Must be 0 (zero) for the current release.
- `Zoff` – Must be 0 (zero) for the current release.
- `RowScale` – Must be 1 for the current release.
- `ColumnScale` – Must be 1 for the current release.
- `HeightScale` – Must be 1 for the current release.

- `Xscale` – Must be 1 for the current release.
- `Yscale` – Must be 1 for the current release.
- `Zscale` – Must be 1 for the current release.
- `RowRMS` – Must be NULL for the current release.
- `ColumnRMS` – Must be NULL for the current release.
- `TotalRMS` – Must be NULL for the current release.
- `rowNumberator` – parameters for the affine transformation from ground coordinate (or local coordinate system) to cell coordinate systems along the row dimension. pType, nVars, order, nCoefficients, and all coefficients of the numerator of the row polynomial, where pType=1, nVars=2, order=1, and nCoefficients=3. The three coefficients are a, b, c in the formulas in Section 6.1.
- `rowDenominator` – parameters for the affine transformation from cell coordinate system along the row dimension to ground coordinate (or local coordinate system). nVars, order, nCoefficients, and all coefficients of the denominator of the row polynomial, where pType=1, nVars=0, order=0, and nCoefficients=1. The value of the single coefficient must be 1 for the current release.
- `columnNumerator` – parameters for the affine transformation from ground coordinate (or local coordinate system) to cell coordinate systems along the column dimension. pType, nVars, order, nCoefficients, and all coefficients of the numerator of the column polynomial, where pType=1, nVars=2, order=1, and nCoefficients=3. The three coefficients are d, e, f in the formulas in Section 6.1.
- `columnDenominator` – parameters for the affine transformation from cell coordinate system along the column dimension to ground coordinate (or local coordinate system). nVars, order, nCoefficients, and all coefficients of the denominator of the row polynomial, where pType=1, nVars=0, order=0, and nCoefficients=1. The value of the single coefficient must be 1 for the current release.

4. Physical Data Storage (1): Schema

The multi-toasted table is used to store the data of PostGIS `PGRASTER` objects in the following schema.



- Two main kinds of tables will be setup to store the metadata and raster data seperately for `PGRASTER` object. One is called PGRaster table for the meta, and the other is call Raster data table for the blocked raster data.
- The PGRaster table should at least have a column in the type of `PGRASTER` which stored the necessary metadata information for a `PGRASTER` object. More than one column could be of the type `PGRASTER`. Such as, you can save both the SPOT and TM images for each state in USA. Or you can save SPOT images for three different years: 1980, 1990, and 2000.
- The Raster data table store the blocked data for image or raster dataset in a GIS application. For each row of raster data table, there can only be one `PGRASTER` object connected to it. But a single raster data table could host the blocked data for more than one `PGRASTER` object, while the data of a `PGRASTER` object could only be saved within the same raster data table. Due to the limitation of physical storage size of a single table, it is strongly recommended that one raster data table should only host the data for one `PGRASTER` object.
- There is a one-to-many mapping between the records in PGRaster table and Raster data table.
- There could also be some other tables, such as table for storing GCP points, table for histogram, the Value Attribute Tables (VAT), and some other additional tables connected to PGRaster data table by the same `PGRASTER_OBJECTID.` About the definition of such tables could be found in the following sections.

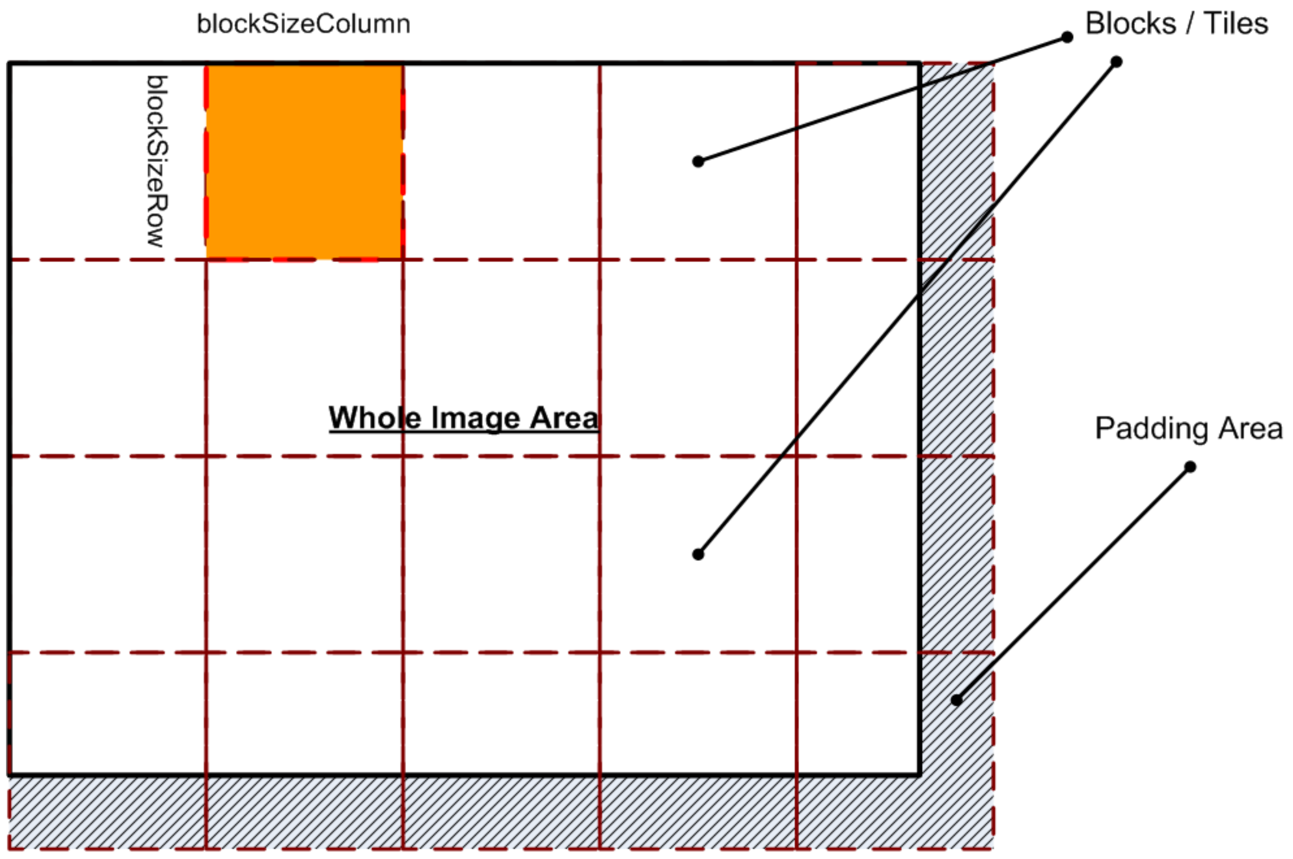
5. Physical Data Storage (2): Blocking, Pyramids Structure and Compression

Blocking storage techniques, pyramid structures and data compression are the three key approach to improve the performance of PostGIS PGRaster in data processing and visualization.

5.1.Blocking Technique

Blocking technique is using to improve the performance of data storage for PostGIS PGRaster. If the whole data is stored in a single record, there wouldn't be any points to implement such a PGRaster data model within a ORDBMS like PostgreSQL. Blocking techniques could be applied to improve the storing performance in three main aspects: asynchronous data processing, asynchronous data visualization and extended limitation of data size. Normally, a single file on modern 32 bit OS have the limitation of size to 4G. It means the largest data size within a ORDBMS record is about 4G.

As the name says, the blocking technique will divided the whole image into small blocks or tiles before storage. Then each block of data will be numbered and saved within a single record in raster data table. While need, the whole or subset of the image could be rebuilt using the number of blocks. When it is necessary, blocks of data could also by processed asynchronously block by block.



By default, the PostGIS PGRaster will carry out a algorithm to specify a proper block size for each input raster dataset which takes into consider the data type, cell depth, dimension, range and bands information. Mostly, the prefer block size are equals to (128, 128, B), which means each block will hold a data range of 128 pixels by 128 pixels and store data of all bands with this range in one single data block. User could also specify the block size in the storageParams while creating a new `PGRASTER` object by importing from a external image files. StorageParams is a text base data structure to express the users' requirement for data storage. More details about storageParams could be found in the later sections. In the stroageParams, you can set blocking flag to TRUE and set the blockSize to any one you like or just skip it to use the default size of each data block. For examples, in the storageParams, you can write the following words to specify your requirement on the blocksize.

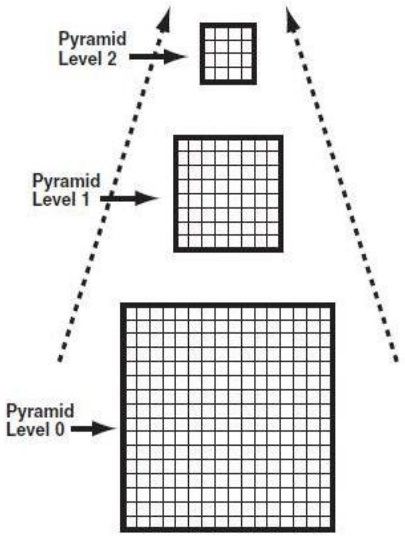
```
“storageParams='blocking=TRUE;blocksize=(256,256,B)';...”
```

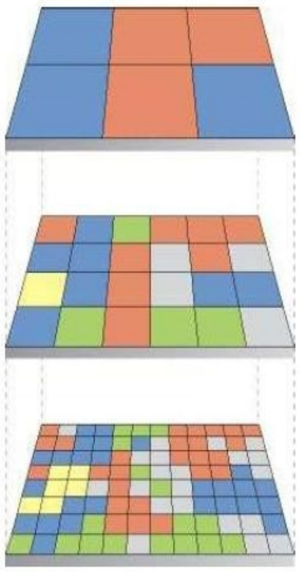
The decision on how to choose a proper block size for a certain input raster dataset is not so easy. Larger blockSize will cause less data records with heavy data read/write overhead for each record. Smaller blockSize also means more data records which could also slow down the performance. But anyway, there should be one that is well-tailored according both the raster dataset/image and the database's ability.

Usually, The dimension sizes (along row, column, and band dimensions) may not be evenly divided by their respective block sizes. Under such circumstance, two approaches could be applied. One is called data padding, which will fill the exceeded area of a block from the edge of image with NoDataValue, as shown in the last figure. Another approach uses three additional attributes together in the Raster data table to remember the actual size of each block along the row, column and band dimension. Currently, both approaches are supported in PostGIS PGRaster. By specifying the blockPadding attribute of `PGRASTER` object to TRUE to inform the usage of data padding for the blocks near the edges.

5.2.Pyramid Structure

Pyramid structure is a fantastic one which could highly improve the performance of visualization, especially for delivering the wide-range high-resolution geo-referenced images via network environment. The mechanism behind the pyramid could be illustrated using the following figure.





Pyramid creation. The spatial extent stays the same, but cell sizes change.

In fact, the pyramid structure will set up several copies of the original image in a coarser resolution by using a certain kind of resampling algorithm. As shown the above figure, after resampling, the number of image pixel will decrease badly along the pyramid level up. It will cost much smaller space of storage on disk(about 75 percents less when one pyramid level up). But the “duplicated” image will cover the same area region with less pixel but bigger spatial resolutions, as shown in the right figure.

In PostGIS PGRaster, user can specify whether to use pyramid structure or not in the storageParams as well as the `rasterPyramidEnable` attribute of `PGRASTER` object. But the rasterPyramidDepth can't be sepcified by end-user, because it will be filled automatically after the PGRaster finish building up the whole pyramid of images after the users set to take the pyramid structure.

If taken, the pyramid levels or pyramid depth could be calculated using the following algorithm:

- 1. Taking the whole image, build the first pyramid up and resize the image to ¼ of the original.
- 2. Check whether the resized image could be hold within a single block. If yes, stop building pyramid structure and count the pyramid level.
- 3. Otherwise, take the last resized pyramid level as the original one, do the same operation as No.2. Until the pyramid level could determine.

Typically, in GIS raster spatial analysis field, they are many algorithms of resampling available for building up the pyramid structures. These five are most popular:

- Nearest neighbor (RM_NN = 0)
- Bilinear interpolation using 4 neighboring cells (RM_BL = 1)
- Cubic convolution using 16 neighboring cells (RM_CUBIC = 2)
- Average 4: using 4 neighboring cells (RM_A4 = 3)
- Average16: using 16 neighboring cells (RM_A16 = 4)

Building up the raster pyramid structures for PostGIS PGRaster, it will take up quite a lot temporary disk space and computation resource. After built up, it will keep static and seldom change.

You can specified the parameters for building up pyramid structures in the stroageParams which will be accepted by many PostGIS PGRaster functions. By default, the pyramid structure will be enabled and the nearest neighbor resmapling method will be taken. You can write such words to the storage Params.

```
"storageParams='...;Pyramid=TRUE;RESAMPLE=RM_BL;...;'"
```

This text tell the PostGIS PGRaster to enable the pyramid structure using the bilinear interpolation resampling method.

5.3.Data Compression

Data compression or image compression/encoding technique is also important in PostGIS PGRaster. Some excellent image encoding algorithm such as JPEG2000 could greatly reduce the size of image with an acceptable image quality. This quality of compression could also be controlled by user. For some other raster dataset that require a lossless data compression, the LZW/LZ77 algorithm could also be applied. You can specify the image compression method and quality if needed to the stroageParams as well as the attributes of `PGRASTER` objects.

```
"storageParams='...;Compression=JPEG-F;Quality=80;...;'"
```

Currently, the following compression / image encoding method will be applied.

- CM_JPEG-B = 0
- CM_JPEG-F = 1
- CM_LZW/LZ77 = 2
- CM_RLE (Run Length Encoding) = 3
- CM_NONE = 4.

The optimization of image data which is merely for visualization purpose only, data of both the original image and pyramid structures will be stored directly in JPEG-F or JPEG-B format block by block. While used, each block of image data could be easily converted smaller pieces of JPEG images and displayed asynchronously in the special order (such as GiST-R tree. Please read more about the spatial indexing of PostGIS PGRaster data).

The end user have the power to choose whether to use the data compression / image encoding techniques. The following image from ESRI shows the image compression ratio and quality for different type of algorithms.

6. Input & Output Functions for `PGRASTER` object

As required by PostgreSQL, any new user-define object type should provide at least a text-based input and output function for it. There are also some need to create a PGRASTER by hand using the text as input in SQL language. In this section, both text-based and binary input/output function will be illustrated here. Specified image encoding schema will be applied during the implementation of such import/export functions.

6.1 TEXT Input & Output Functions

Text input and output functions for PGRASTER object in PostGIS/PostgreSQL will follow both a WKT and a XML style.

- The WKT/Flat Text Style (Default)

As a WKT/Flat Text style, the raster data and related meta will be provided in text of “name=value” items which are separated using “;”.

- The XML Style

6.2 Binary Input & Output Functions

(To Be Finished....)

7. Utility Functions for `PGRASTER` Objects

(Coming Soon....)

8. System Tables and Namespace

(Coming Soon....)

9. Indexing `PGRASTER` in PostGIS

(Coming Soon....)

10. Useful Tools: Import, Export and Viewer

(Coming Soon....)

11. Examples and Demo

(Coming Soon....)

12. Task and Schedule

(Coming Soon....)

13. Acknowledgment

(Coming Soon....)

14. Appendix A: Object Types and Functions

(Coming Soon....)

15. Appendix B: Table Structures and System Tables

(Coming Soon....)

16. Reference & Web-Links

- References

[1] PostGIS Documentation of Manuals, http://postgis.refractions.net/docs/

[2] PostGIS Online Wiki, http://www.postgis.org/support/wiki/

[3] PGCHIP - The GDAL PostGIS driver for raster data, http://simon.benjamin.free.fr/pgchip/

[4] Oracle® Spatial GeoRaster 10g Release 2 (10.2) Documentation, http://download-uk.oracle.com/docs/html/B14254_01/toc.htm

[5] Oracle® Spatial User's Guide and Reference 10g Release 2 (10.2), http://download-uk.oracle.com/docs/html/B14255_01/toc.htm

[6]

- Web Links

[1] PostgreSQL Website, http://www.postgresql.org

[2] PostGIS Website, http://www.postgis.org/

[3]