3D GIS DATA MODEL USING OPEN SOURCE SOFTWARE

A. Scianna, A. Ammoscato

CNR-ICAR, Viale delle Scienze edificio 11, 90128 Palermo, Italy – andrea.scianna@cnr.it,alessio.ammoscato@unipa.it

Commission IV, ICWG IV/8

KEY WORDS: Cartography, GIS, Technology, Interoperability, Spatial, Global-Environmental-Databases

ABSTRACT:

Today many kinds of applications requires data containing actual three-dimensional data; fields like urban and town planning and pollution studies need 3D data, both for visualization purpose, as well as carry out many spatial analysis. This research -*Management and use of distributed 3D data by open source Web-GIS software* - is part of the Italian "PRIN 2007"[*] research project, aimed to build urban and suburban 3D models, and to interact with them using open source software only.
Particularly free and open source software, used for the experimentation here shown, are *Blender* and *PostGIS*; the first one has been used to build and structure three-dimensional data, the second one for data allocation. These software interact using scripts, written in Python language.
Buildings have been modeled upon the GIANT3D model (Geographical Interoperable Advanced Numerical Topological 3-Dimensional Model) developed in the research "PRIN 2004", regarding "Evolved structure of numerical cartography for Gis and Web-GIS".
Python scripts, activated by Blender, allow to allocate data into a spatial database implemented through *PostgreSQL* and *PostGis*, that could be a remote database somewhere on the net; all geometrical and topological information, implemented in the 3D model, are so transferred in *PostGIS*. These information can be retrieved by *Blender* using other *Python* scripts, so *Blender* fully interacts with 3D data allocated in *PostGIS*. These data can be also accessed by many other clients, both directly using a database client, as using other protocols (like HTTP on the internet). Next step is to build an open source viewer, or a plugin for internet browsers, that allows client to visualize, explore and inquiry 3D model, retrieving data from database.

1. INTRODUCTION

1.1 The state of the art of 3D modeling using open source software

Need for three-dimensional features has been remarkably increased during last years in many computer related fields, like GIS, GPS navigation, games, ... Generally, the third dimension helps to understand immediately where the user is located, or to better analyze a situation or a model: it is essential in every advanced geographical information system or everywhere correct 3D position is important.
So 3D modeling became more and more important to structure any kind of information related with fields mentioned above, imposing software to get themselves better, refining their 3D features.
Among many 3D software, free 3D modeling software have been analyzed; main aim is to investigate their suitability to build various three dimensional model and, above all, if and how they interact with databases containing spatial data.
Examined software were:

- K-3d (www.k-3d.org);
- Wings3d (www.wings3d.com);
- Blender (www.blender.org).

Another useful 3D modeling software is Sketchup by Google; this software is powerful and very easy to use, and it can be extended using “Ruby” scripts; however this software hasn’t been used in this research, due to the fact this is not *free* and *open source* - it is only freely distributable in the basic version-.

K-3D is a free and easy-to-use 3D modeling and animation software. It has tools to build and edit NURBS, patches, curves and it can handle animations too.
It has a very user-friendly interface, easier to use in respect to other software; during objects' creations and editing user can add and modify further attributes besides default ones. Moreover it has a script tool, which is very useful to define customized functions or automate command sets or perform batch processing (this software follows the "*everything is a plugin*" philosophy).
Unfortunately it's on a development stage (version 1 is not yet available) and above all, compared to other software, it has a limited amount of import-export formats.

Wings3d is a 3D modeler, that allows to assign materials, vertex colors, UV coordinates and textures.
It can import and export models using following formats: Nendo (.ndo); 3D Studio (3ds); BZFlag (.bzw); Autodesk FBX (.fbx); Cartoon edges (.eps); Kerkythea (.xml); Lightwave (.lwo, .lxo); Wavefront (.obj); POV-Ray (.pov); Renderware (.rwx); StereoLithography (.stl); VRML 2.0 (.wrl); DirectX (.x).
Due to the fact that it is not enabled to make animations, and above all because it has less function than other software, we didn't sawn fit it to our aims.

[*] PRIN 2007: “Interoperabilità e gestione cooperativa di dati geografici dinamici multidimensionali e distribuiti con strumenti GIS liberi e Open Source” – Principal investigator Paolo Zatelli.

Blender is a powerful 3D modeling software, endowed with advanced functions and developed by the open source community; it is available on many platforms (like Microsoft Windows, Mac OS X, Linux, IRIX, Solaris, NetBSD, FreeBSD, OpenBSD).
It has many features like a wide range of 3D objects (polygon meshes, NURBS, Bezier and B-spline curves, metaballs, . . .), fast and powerful rendering and shading tools, animation environment and, last but not least, a set of tools for real-time 3D game creation, with full support for audio, collision detection, dynamics' simulation, dynamic constraints.
Its features can be remarkably extended using Python scripts, that allow to customize entire projects introducing new functions too.
At last, due to all this features, *Blender* software was chosen for our experimentation.

1.2 Blender software

As mentioned above, *Blender* is the most suitable software to work on three-dimensional geometries and primitives, both in general as particularly for games. It can define and handle several kinds of geometries, adding also customized properties, allowing to import-export in many different formats; another powerful feature is the chance to use Python scripts to do everything you need and that is not included among standard functions.
One of the main difficulties, using *Blender* to allocate geographical data, is represented by kinds of -CAD- primitives, that are different from which usually used in GIS. Some of primitives (Basic Objects) used in Blender are “Mesh”, Curve”, “Surface” and they are entirely reported below.

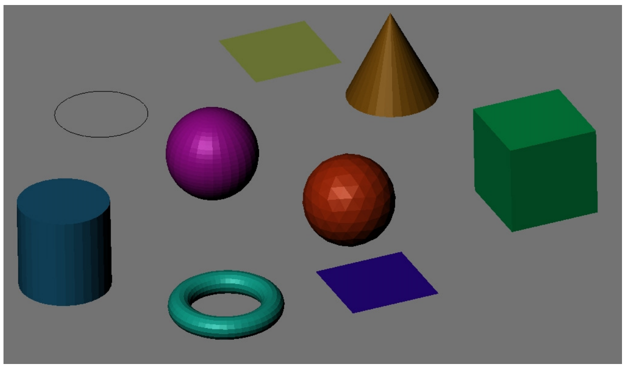
Figure 1. Primitives in Blender

Used meshes are:

- Plane: it contains four vertices, four sides and one face. It's an object that usually has two dimensions only, but it can become a 3D object moving one or more vertices.
- Cube: it has eight vertices, twelve sides, six faces: it's an actual 3D object.
- Software manages other primitives not interesting for purposes of this research, as circle, UV Sphere, Icosphere, Cylinder, Torus, Cone, Grid.

Curves and surfaces are like meshes, but they are defined using mathematical functions, instead of points. Blender uses Bézier curves and NURBS (Non Uniform Rational B-Splines). They are both defined using control vertices, interpolating (Bézier) or attracting (NURBS) curves.
Bézier curves are parametric curves defined by a polynomial, starting from control points; curve is graphically fully contained in its control points convex set, starting from which you may transform the curve.
Blender widely uses Bézier curves, defined with three points, the first to move and the latest to modify the shape of each curve.
NURBS are defined as rational functions; a NURBS is defined with many variables like its order, a group of control points and a knots vector; unlike other type of curves (like Bézier) NURBS are more suitable to exactly follow a contour or to adapt to a 3D shape. For example, a Bézier curve will not be a circle, while a NURBS circle will be exactly a circle.
Surfaces represent a general case of NURBS: to obtain a curve you make a one-dimensional interpolation, while to obtain a surface you make a two-dimensional interpolation (U,V): using a one-dimensional interpolation you obtain a curve as a simplified case of a surface. A two-dimensional control points grid defines the shape of these NURBS surfaces.
As you can see above, *Blender* doesn't use points and lines as primitives, probably due to the fact that these primitives are not useful in 3D modeling; anyway they can be schematically presented using a partially modified spline primitive, i.e. using a spline constituted by a single point - to obtain a point - or a couple of points – to obtain a line -: this trick has been used in this research as written below. Thank to this expedient it's possible to define points and lines, that are basic primitives in GIS applications.

2. GIS AND DATA BUILDING

2.1 3D primitives for data allocation

Among many applications requiring actual three-dimensional data, we can cite those regarding urban and town planning, pollution studies: spatial analysis is their main aim and is often their core too. Another example, among many others, is represented by the cadastres, where the information on the third dimension (i.e. many apartments in a skyscraper upon a single parcel) is becoming more and more important day by day.
As reported in [*Yanbing et al., 2006*], main 3D geographical models - useful for city modeling and visualization - use following primitives:

- *3DFS*: node, arc, face, edge;
- *n Tuple model*: 0-3 cell;
- *SSM*: node, planar face;
- *UDM*: node, triangle;
- *OO3D*: node, segment, triangle.

Other kind of models, i.e. terrain models like 3D-TIN, GRID, TEN, Octree (the three-dimensional analogous of quadtree), use further primitives like rectangle, tetrahedron, cube.
Against these remarks, currently spatial databases contain two-dimensional data only; in some more advanced cases, three-dimensional objects are modeled using two-dimensional primitives, using 3D coordinates in order to describe them (i.e. two-dimensional polygons can identify a parallelepiped bounding it). These solutions can lead to various malfunctioning, e.g.:

- objects assumed as 3D are not real 3D objects, and they are not considered as;
- 3D objects cannot be validated, and the geometrical functions can also make mistakes;
- sometimes data structure allocates data regarding a 3D object in several records, each containing a bi-dimensional object; this implies that database software cannot manage very well data, especially handling

large data sets; a 1:1 relationship is to be preferred, and this is possible only using a real three-dimensional primitive.

As written in (*C. Arens et al. - 2005*), 3D spatial objects can be modeled using various 3D primitives; among them:

- CAD and 3D computer graphics' objects: solids can be modeled by wireframe (points, line and curves), by surfaces (using surfaces with no thickness) or by solids (using Constructive Solid Geometry or Boundary Representation);
- Tetrahedron (*Stoter and Van Oosterom, 2002*): it consists of four triangles that form a closed object in 3D coordinate space; this is the simplest 3D primitive; using this primitive there is not a 1:1 relationship between the object and the object's representation in the database;
- Polyhedron (*Stoter and Van Oosterom, 2002*): it is like a polygon, but in 3D. It is constituted by several flat faces that enclose a volume, carrying out the 1:1 relationship between object and its database allocation.

2.2 GIANT3D model

During last years a three-dimensional model to structure data has been developed by our research group. Its name is GIANT3D (Geographical Interoperable Advanced Numerical Topological 3D), that summarizes that it's a geographical and topological model too.

In this model features of objects and relationships between objects have been defined through a geometric-topological model, according to the theory known as Boundary Representation (*Foley, 1995*). Generally, the geometric model and the topology are separately defined; in this model they are defined at the same time.

Real objects are represented by primitives of 0, 1, 2, 3 dimensions (node, edge, face, toposolid), each primitive is composed of higher order than primitives of the lower neighbour. A solid is bounded by at least 4 surfaces connected to each other, surfaces must be flat and, because it can distinguish inside from outside, must be directed using the "normal" value. Normal of a face is represented by three coordinates of a point: line passing through this point and centre of face is the normal direction. Each surface is bounded by an ordered set (outer ring) of segments (edge); empty spaces inside are delimited by one or more inner rings. Each segment should be straight, going from node to node. Only nodes are defined by their three-dimensional coordinates (x, y, z).

2.3 Blender and geodatabase interaction using python scripts

Generally, databases allow you to manage wide amount of data, efficiently managing it and regulating different accesses, with different users at the same time, each with its permissions.

These concepts prove true also in the case of a spatial database, where spatial data are allocated instead of alphanumeric data only. But usually 3D modeling software use file system to allocate data, even if they are very large.

To investigate on relationship between 3D software and database, firstly a broaden bibliographic research was made, especially looking for previous similar researches in specialized web sites, regarding databases and 3D modeling. From this research only two projects arise: one called "Blenderpeople" and a beta project - carried out by Nathan Letwory, Blender's developer - regarding 3D data allocation using *MySQL* database. Blenderpeople is a project, running on *Blender 2.42a* only, concerning contemporaneous animation of several three-dimensional elements: a typical example is constituted by a battle fought by two armies, where several elements (like soldiers and horses) move on the scenario seemingly independently.

Python is a high-level programming language - here used as a scripting language -, that supports multiple programming paradigms (primarily object oriented, imperative, and functional) and features a fully dynamic type system and automatic memory management, similar to Perl, Ruby, Scheme, and Tcl.

Some Python scripts are allocated within the "blenderpeople.blend" file: they allow you to allocate on the database all information regarding both the scenario as fighting elements. This project has a fundamental lack: *MySQL* doesn't natively support geometrical or geographical characteristics of objects.

Taking into account this lack, a new spatial data allocation project has been developed, using *PostGIS* (a geo-DBMS extension based on PostgreSQL) as database server and Blender as 3D modeling software.

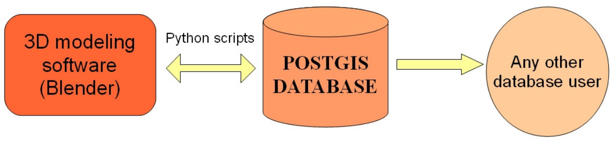

Figure 2. Relation between data and software in our project

In advance, "Psycopg" software has been installed; *Psycopg* is a *PostgreSQL* database adapter for the Python programming language, that allows Python scripts to interact with PostgreSQL databases.

Various python scripts have been made, with following aims:

- export Blender entities in *PostGIS* database;
- read geometrical entities from *PostGIS* database;
- draw previously read entities (point, lines, polygons) in *Blender*;

Due to the fact that *Blender* does not directly handle points and lines, points are drawn like a Bezier curve with start point only, while polylines are drawn like a Bezier curve having null curvature between vertices.

Geometrical entities are stored in database using functions to convert coordinates from WKT (Well Known Text) human readable to WKB (Well Known Binary) and viceversa.

Subsequent scripts handle 3-dimensional data using 2-dimensional primitives, especially due to the limitations of DBMS. To handle 3-dimensional data using 3-dimensional primitives represents a further development of this research work.

A script that allows to connect to *PostGIS* database and upload data from three-dimensional Blender objects has been made and partially reported in appendix A, as well as that one that read geographical information from a table (three-dimensional points or lines) on *PostGIS* database and, after verifying kind of data, draws points or a polyline, both using fictitious Bézier curves.

3. TOPOLOGICAL MODELS

3.1 Topological model using PostGIS and triangles

Moving from above considerations, a very simple topological model has been defined using *Blender* and *PostGIS*, partially according to GIANT3D model. Data are allocated using a "three

table only" schema, useful for modeling flat surfaces, representing roads or buildings.
Schema of this model is represented below:

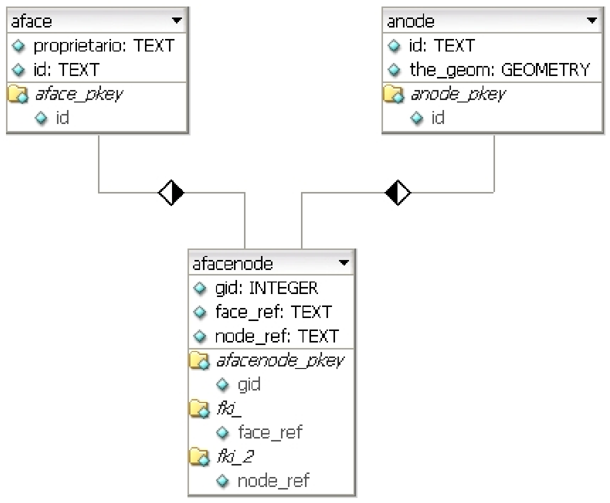

Figure 3. Simple schema using triangles

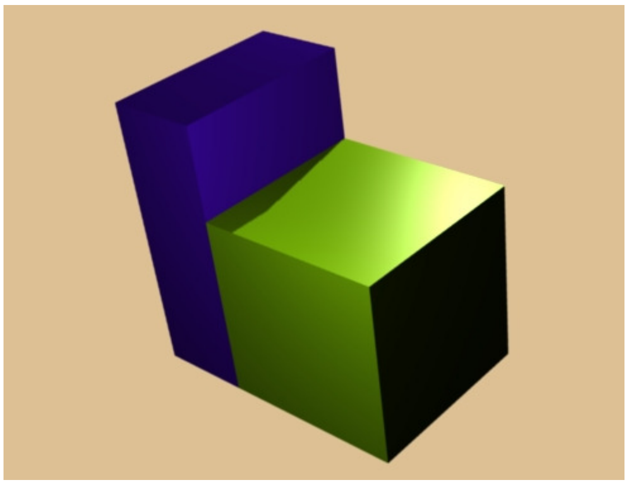

Figure 4. Two contiguous buildings

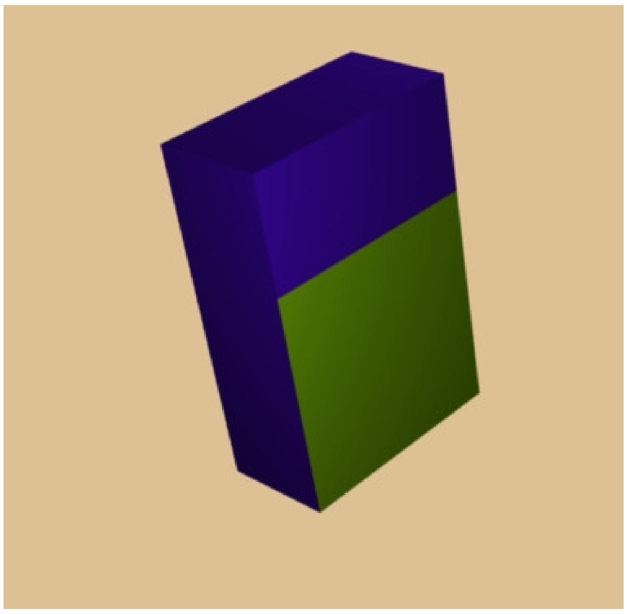

Figure 5. The highest building with the shared wall

An important feature of this way of structuring data is the use triangular faces, so there is no need for edges, due to the fact that a face defined with only three points is unequivocally identified.
In the following illustrations a simple case study is represented, with two buildings with a common part; each building is structured using triangles.
Buildings are modeled using *Blender*; after a triangular conversion has been made for every face, a Python script uploads all information into database. Firstly "anode" table and "aface" table are populated, therefore "afacenode" table is written; this order because there are two foreign key constraints on relationship table "afacenode", pointing to "aface" and "anode" tables.
Part of script that allows to connect to database and to write information according to this triangular schema is reported in appendix A.

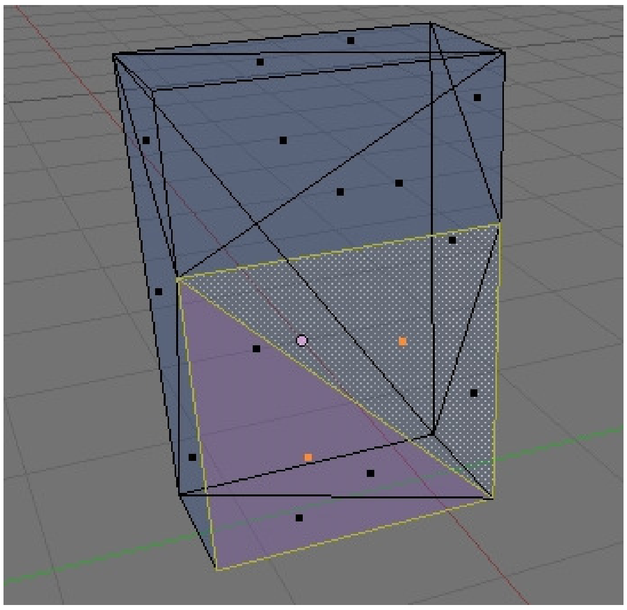

Figure 6. Inner structure of highest building

3.2 Topological model using PostGIS and quads

Model explained above is very simple, since it hasn't edges as primitive. To obtain a more complete model, suitable to better represent real world, like buildings, walls, roads, sidewalks, . . . edge primitive has been added. This model has a more complex structure, so it can handle relations between objects and faces, faces and edges and between edges and vertices; these are many-to-many relations and each can be represented by a table, with foreign key constraints pointing to object, face, edge and node tables.
Python scripts that allow interaction between Blender files and database's tables have some routines that prevent data duplication and ensure a strong data structure; this kind of controls is currently based on a geometrical comparison.
For example, to verify if a face exists a very simple criterion has been used: a script verifies if a face already exists searching the same area and centre (automatically calculated by *Blender*). If face already exists, script takes and uses its unambiguous identifier (*uuid*) to populate relation tables of database; similar procedure verifies existence of edges and nodes, with geometrical comparison, and it uses data already allocated to populate relationship's tables. So if an edge is shared by two or

more faces, script allocates it one time in one record only, but many records in face-edge table can point to it.
Schema of this model is represented below:

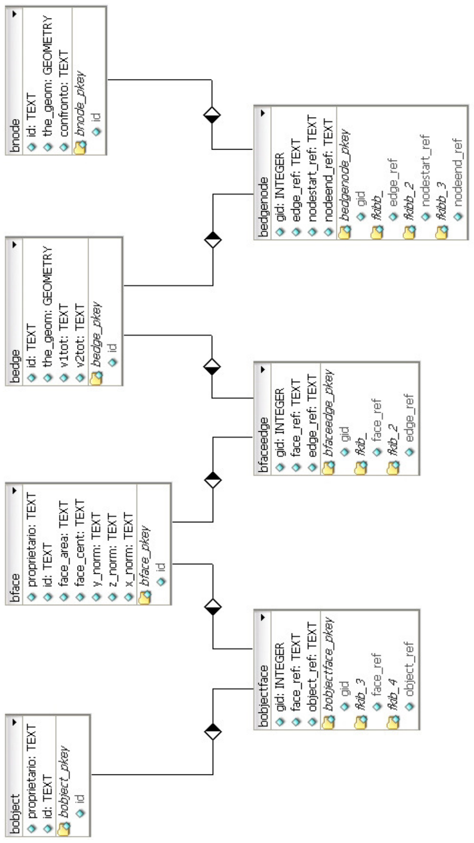

Figure 7. Schema using quads

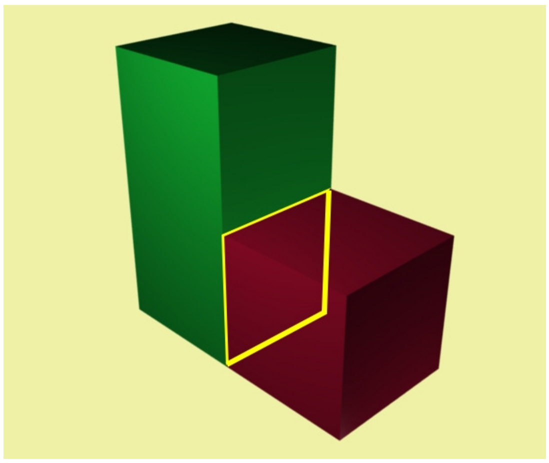

Figure 8. A common part

In figure above there is a more complex situation of a simple common part: face highlighted in yellow is shared by two parallelepipeds, that can represent two buildings, or two building units of a building. The common part represents both a full wall in the red building and a part of wall in the green one; to be correctly processed, these objects have to be drawn following a correct way of design and subsequent editing.
These procedures makes topologically correct data, without any data duplication and redundancy: as a consequence data can be correctly reconstructed, i.e. by a 3d modeling software, knowing this way of generating them.

Currently only fundamental Blender objects, like planes and cubes, have been used to structure model defining our specific objects; defining a new three-dimensional primitive, useful to straight define real objects, is seen as a further deepening of this matter.

4. CONCLUSIONS

Excellent results have been achieved using open source software to model and store geographical data. Use of *Blender* to build three-dimensional objects allow to fully take advantage of this powerful software. On the other hand, *PostGIS* database has many features and it's large-built enough to correctly allocate and manage large amount of data like geographical ones.
Python scripts behave as connector between these software, allowing a good bidirectional interaction. As future developments we can list:

- improving *Python* scripts, enhancing calculation speed by code optimization;
- introducing customized objects into *Blender* (like buildings or roads), to improve drawing speed and obtain a more realistic model too;
- introducing a customized "really 3d" primitive into *PostGIS* software, to allocate data in more realistic three-dimensional primitives; at present use of two-dimensional primitives in a three-dimensional way is less realistic and it holds back data processing.

References from Journals:
Arens, C., J. Stoter and P. van Oosterom, 2005. Modelling 3-D spatial objects in a geo-DBMS using a 3D primitive. *Computer & Geosciences*, Volume 31,2, pp. 165-177.

Khuan, C.T. and A. Abdul-Rahman, 2006. 0D feature in 3D planar polygon testing for 3D spatial analysis. *Geoinformation Science Journal*, Vol. 6, No. 1, pp 884-92.

Khuan, C.T., A. Abdul-Rahman and S. Zlatanova, 2008. New 3D data type and topological operations for geo-DBMS. *Urban and regional data management: UDMS annual 2007*, Taylor & Francis, pp. 211-222.

References from Books:
Foley, J., van Dam, A., Feiner, S. & Hughes, J., 1995: *Computer Graphics: Principles and Practice*. Addison Wesley, 2nd Ed.

Khuan, C.T., A. Abdul-Rahman and S. Zlatanova, 2008. 3D Solids and Their Management In DBMS. *Advances in 3D Geoinformation Systems*, Springer Berlin Heidelberg, pp. 279-311.

Yanbing, W., W. Lixin, S. Wenzhong and L. Xiaomeng, 2006. 3D Integral Modeling for City Surface & Subsurface. *Innovations in 3D Geo Information Systems*, Springer Berlin Heidelberg, pp. 95-105.

Zlatanova, S., 2006. 3D geometries in spatial DBMS. *Innovations in 3D Geo Information Systems*, Springer Berlin Heidelberg, pp. 1-14.

References from Other Literature:
Arens C., 2003. Maintaining reality; modelling 3D spatial objects in a geo-DBMS using 3D primitive, Master's Thesis, TU Delft.

Pu, S., 2005. Managing freeform curves and surfaces in a spatial DBMS, Msc Thesis, TU Delft.

References from websites:
Many authors, 2009. PostgreSQL manual. http://www.postgresql.org/docs.

Many authors, 2009. PostGIS manual. http://postgis.refractions.net/documentation/.

Many authors, 2009. Blender manual. http://www.blender.it.

Many authors, 2009. Blender documentation. http://www.blender.org/documentation.

Many authors, 2009. Python documentation. http://docs.python.org.

Many authors, 2009. Python and Blender documentation. http://en.wikibooks.org/wiki/Blender_3D:_Blending_Into_Python.

Many authors, 2009. Blender documentation. http://en.wikibooks.org/wiki/Blender_3D_Reference_Guide.

APPENDIX A

Parts of script that reads geographical information from a table (three-dimensional points or lines) on PostGIS database and draws points or a polyline.

```
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
#start connecting to database#
o = psycopg1.connect("dbname=nuovo user=postgres
password=postgres")
c = o.cursor()
#end connecting to database#
sce = Blender.Scene.GetCurrent()
c.execute("SELECT ASEWKT(the_geom) FROM puntizz") # selection
from a table
rows = c.fetchall()
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
vect3d_list=[]
####### vector class definition ############
class vect_3d(object):
def __init__(self,rigaxf,rigayf,rigazf):
self.rigaxf=rigaxf
self.rigayf=rigayf
self.rigazf=rigazf
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
############## Draw points ##############
def dispunto():
######### Create a new mesh
i=0
  while i<len(vect3d_list):
    print "i:",i
    poly=NMesh.GetRaw()
    v = NMesh.Vert
(vect3d_list[i].rigaxf,vect3d_list[i].rigayf,vect3d_list[i].rigazf)
    poly.verts.append(v)
    i+=1
######### create a new object using new mesh
    polyObj = NMesh.PutRaw(poly)
  Blender.Redraw()
dispunto()
########### end drawing points #############
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

Parts of script that connects to database and writes information using triangular face model.

```
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
for face in faces:
  ####### face writing procedure #######
  print "Face " + str(facenum)
  id = uuid.uuid1()
  c.execute("INSERT INTO aface(id) VALUES ('"+ str(id) +"')")
  o.commit()
  for vertex in face:
    confronto= str(int(round(vertex.co[0],3))) + " " +
str(int(round(vertex.co[1],3))) + " " + str(int(round(vertex.co[2],3)))
    print "--------- New vertex -----------"
    #search existing data in database #
    c.execute("SELECT ASEWKT(the_geom) FROM anode") #
selection
    rows = c.fetchall()
    numoggetti= len(rows)
    count = 0
    for j in range(len(rows)):
      print"j: ", j
      print "riga: ", rows[j][0]
      print "confronto: ", confronto
      count = string.find (str(rows[j][0]), str(confronto))
      pippo = str(rows[j][0])
      if confronto in pippo:
######### start writing procedure for existing data #########
        c.execute("SELECT (id) FROM anode") # selection from a table
        rows = c.fetchall()
        idnode_exist = rows[j][0]
        c.execute("INSERT INTO afacenode(face_ref,node_ref) VALUES
('"+ str(id) +"','"+ str(idnode_exist) +"')")
        o.commit()
######### end writing procedure for existing data #####
        print "\n"
        count = 0
        break
        else:
          count += 1
          j+=1
    if count >= 0:
### start writing procedure #####
      idv = uuid.uuid1()
####### write vertices ########
      c.execute("INSERT INTO anode(id,the_geom) VALUES ('"+
str(idv) +"',GeomFromEWKT('SRID=3004;POINT("+
str(round(vertex.co[0],3)) +" "+ str(round(vertex.co[1],3)) +" "+
str(round(vertex.co[2],3)) +")') )")
      o.commit()
####### write "afacenode" table ########
      c.execute("INSERT INTO afacenode(face_ref,node_ref) VALUES
('"+ str(id) +"','"+ str(idv) +"')")
      o.commit()
### end writing procedure ####
    vertnum += 1
  facenum +=1
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

Full versions of scripts are available at http://www.dirap.unipa.it/python_scripts.