Automatic Generation of Urban Virtual Environments

Tiavina Tantely, Nivolala
University of Antananarivo
Antananarivo, Madagascar
e-mail:
tiavina.tantely@gmail.com

Tahiry, Andriamarozakaniaina
University of Antananarivo
Antananarivo, Madagascar
e-mail: filamatra@gmail.com

Jean-Pierre, Jessel
Paul Sabatier University
Toulouse, France
e-mail:
Jean-Pierre.Jessel@irit.fr

ABSTRACT
This paper presents an approach to automatic city generation relying on the existing set of models. First, a hierarchical database containing the main elements of an urban environment is created, and then procedural techniques are applied in order to generate the environment with the data from the database, for the chosen city. This approach adds randomness while keeping control on the main features of the environment.

Keywords
City generation, 3D database, 3D visualization, virtual worlds, computer graphics, declarative modelling

1. INTRODUCTION
Urbanization is occurring rapidly, and since 2008, more than half of the world population lives in cities. Those cities inspired a number of resulting urban virtual environments, and predictably, many new applications began to use them as a platform to integrate numerous information on cities coming from various resources. These environments are used for visualisation and exploration of urban landscapes as well as to assist in various other tasks as they offer plentiful possibilities in the field of city, traffic or disaster management as well as the creation of virtual worlds.

Models of 3D cities, in an effort to get closer to the real ones, are becoming increasingly complex in terms of spatial and thematic structures. In order to store all this information, standardized data models, ensuring consistent and inter-operable data structure, have to be built. On the other hand, all the progress made in computer graphics and hardware enables 3D virtual worlds to be more and more complex visually speaking. As visual quality of video games improves with each new generation of graphics hardware, the user expectations increases proportionally. To manage the display of this quantity of data, a possible alternative to manually creating large amounts of models is to apply procedural techniques, along with the use of a few models. Indeed, a complex city containing many different models would take a long time to build manually, and to minimize the costs of world's creation, the generation of objects with a procedural approach, not only adds controllable randomness to the world but also gives varied results.

Our work is focused on creating a hierarchical database model that contains the basic elements of an urban environment, and from these data, our system will automatically generate a city respecting the architectural style in the database, which will keep control on the costs of creation and the general appearance of the cities.

2. RELATED WORKS
Past research has tackled the problem of automatic city generation, each one focusing on a particular aspect. Several standards exist to represent city data : the IFC standard [9] is a file format generally used in the building and construction industry to describe, exchange and share information. One of the most popular data models focusing on cities as a whole is CityGML [7]. The 3DCityDB project [14] aims to reuse the main features of CityGML, while simplifying the complex data model for a more compact database and improve the speed of queries, our model is also largely influenced by the latter. The QUASY model [2] has similarities with CityGML but it adds semantic extensions to the usual basic models.

Another interesting approach is to reconstruct city models from facade photographs [12], to take these facades and generate variations [1], or to use terrestrial scan data [4], laser scanning and digital ground plans [3].

The use of grammars, L-systems in particular, showed successful results in the modeling of plants. Grammars have been extended to the modeling of architecture since the introduction of the shape grammars [15], later, other novel shapes extended this initial grammar, introducing new concepts and operations, namely : the Split Grammar [16] CGA Shape Grammar [11].

Procedural methods were used for the CityGen system [10] in order to create cities interactively by direct manipulation of parameters of the generation algorithm via an interactive interface. Undiscovered Worlds [6] generate pseudo-infinite virtual worlds by vertical extrusion of floor plans, which is possible only by generating the visible part of the world at once.

3. OVERVIEW
We present a system that allows the user to feed the database using custom models for each chosen city and then generate a virtual world respecting this urban style. The creation of this system is done in two steps:

- The design and feeding of the database.
- Procedural generation of cities from this database.

The rest of this paper will be structured as follows: in the next section, we describe the methods used for the representation of the main features of cities and the

feeding of the database. Section 5 focuses on the random generation of cities from these key items. Section 6 presents the results of our system and the last section concludes the paper and provides some prospects of evolution.

4. THE DATABASE

4.1 Database design

The database is based on PostgreSQL and its spatial extension PostGIS that adds new geometric and geographic types to the database. It will represent the geometric and thematic aspects of a city. A geographical region will contain a set of cities, and a city is made up of buildings, streets, cars and other items that will be called urban objects, the latter are generic objects which may contain everything that is not a building.

The solid model of buildings is geometrically represented using Boundary Representation [8] or BRep. A solid is represented by its bounding surfaces, a surface by its edges, and the edges by vertices defined by 3D coordinates. In our case, a simplified model is used: all geometries will be stored as polygons and aggregations of polygons, as any surface-based 3D object contained in the environment may be modeled using these. All spatial objects are built from polygons compositions. Aggregations are done between solid and representative surfaces. We also introduce a table containing appearance data (textures and colors which are typical of the city in question) which are attached to geometric surfaces.

In addition to surface-based elements, we introduce the predefined objects for the cases where a geometrically complex element is instantiated multiple times for one or more buildings. They are 3D models that are first normalized and then stored as such in the database, they will be called each time they should be instantiated in the environment, and each reference will contain the corresponding transformation matrix.

To keep consistency between the entities, the geometric elements are linked with the thematic elements with the same level of hierarchy [13]. The building itself is composed of bounding faces that may contain openings. A building may have other facilities outside (fireplace, balcony ...). Respecting the spatial and semantic consistency, buildings are associated with geometric solids, bounding surfaces (walls, roof) and openings (doors, windows) with geometrical surfaces and predefined objects, respectively.

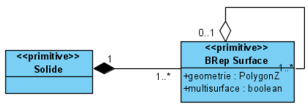

Figure 1. Simplified model of the database's geometries

4.2 Database feeding

The feeding of the database is done through model insertion via Blender and its Python API. The database currently will only contain 2D data, 3D objects are modeled using primitives in 2D with 3D coordinates. We will directly use the geometric polygon, as edges or isolated vertices are not representative in a city.

Preliminary operations: Before executing the script that will perform the transition to the database, the user must manually perform some operations in Blender. All meshes of the same urban object must be grouped. In the case of a building, this group should be assigned a custom property named 'type' with the value 'building', this property may be omitted for a generic urban object. In the database, bounding surfaces may contain openings, in order to respect this constraint, the openings (doors, windows) must have a parent-child relationship with the bounding surface (wall, roof) they are attached to. All these entities will also have a custom property the value of which will be their function ('wall', 'roof', 'door', 'window'). In case the user does not assign any type or function to the objects in the scene, they will be inserted into the database as generic objects. If an object in a building group has no custom property, the script will assume that this is a building facility and insert it into the corresponding table.

Inserting surface-based objects: The objects are first identified by theme. The solid forming these objects are inserted first. Afterwards, the script retrieves all the object's thematic components. For the building, meshes that do not have parents are collected first, meaning : roofs and walls, and for each object, an entry in the bounding surface table is inserted. The corresponding polygons and their vertices are then retrieved; with this list of vertices, a polygon entry will be inserted into the geometric surface table. Note that whenever we are dealing with a polygon, should it be concave, our script will break it down into a set of smaller and convex polygons, this will serve us later during the generation step. After that, appearance data is extracted from the above multi-surfaces and inserted into the corresponding tables.

Inserting predefined objects: Children of each bounding surface are collected and here comes into play the concept of predefined objects as the same window will often be reused for one or more buildings. Said window is stored only once as a 3D model in the predefined form table under a unique name. Each time it is used, a new table entry is created as a reference to the entry in the predefined form table that uses it, accompanied by a transformation matrix to store the translation, rotation and scale to apply this new instance. All these items are standardized before export.

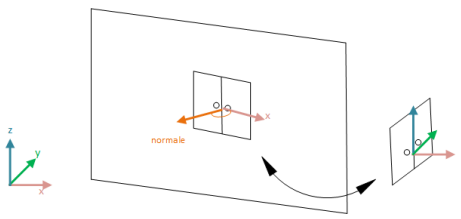

Figure 2. Preparing the predefined object before exporting

Their local center is placed at the center of gravity, their location is moved to the origin of the scene and their local pivot is oriented so that the front face of an opening is perpendicular to the x axis. To correctly orient this pivot, we'll refer to the position of the opening once properly placed against its closing surface. At that time, this opening is properly oriented along the x and y axes to make its front face perpendicular to the x axis, it should rotate around the z axis. The value of the rotation angle applied to correct the orientation is the angle formed between the x axis and the normal of the nearest polygon constituting the wall that is also the parent of

the said opening.

5. THE CITY GENERATION

The last step is the generation of a simplified city from the database models. This city consists of a street and buildings on both sides of the street. This part is carried out with the Unity3D engine. The terrain and the streets are placed first. After querying the base on a particular town, we will get a number of buildings, urban objects and cars.

5.1 Buildings

To each building are attached child elements : bounding surfaces which are a composition of convex polygons, and openings: predefined objects accompanied by their transformation matrices. To generate polygon-based elements, a simple triangulation is performed using the coordinates of the polygon vertices [5]. Next we determine the UV texture coordinates in order to project a 2D image correctly on the surface of a 3D model by calculating the coordinates of the triangles in the image used as the texture attached to the triangles of the 3D object. We can do this by valuing which axis of each polygon must be eliminated so that we can keep only two axes in 2D which are normalized between 0 and 1. As for the predefined objects, they are instantiated at the origin, then the rotation, translation and scale extracted from the transformation matrix are applied before they are attached to the building.

Adjustments on buildings: When a building is generated from a database, we calculate its bounding box to ensure the placement of buildings at random positions while checking for collisions. Next, a pivot is attached to that building. The pivot is positioned in the center of the bounding box, and the x axis is oriented towards a door, because often buildings are oriented in respect to a front door.

Once all database buildings are generated, the existing buildings are cloned and placed on a random, unoccupied position on the terrain. Once the clone is placed, we calculate the rotation to be applied to it. We choose the building rotation so that its door is facing the street by default (the x axis of its pivot, previously calculated, points to the street). Lastly, we try to adjust the scale of the resulting building, if necessary, an item that will serve as a reference for comparison to correct the building scale should be chosen. The value of the vertical height of the door was chosen because, out of all the elements of a building, this value is less subject to variation in their style (they must, at least, have a minimum height). However, we allow a margin of more or less 20 % compared to the standard height. The standard height is calculated relative to the only element that was present before any generation: the street and its width. Finally, the new scale applied to the building will have a difference in scale between the standard height of the door and its current height.

Thematic elements: For each building clone placed on the terrain, its thematic elements are selected, and instead other ones are placed selected in the database that belong to the same city and are of the same type. In order to do this, thematic components of the building are individually selected. We start by retrieving bounding surfaces which are isolated by type ('wall', 'roof'), and for them, random appearance data is retrieved in the database for the same type of element and with the selected city. These new appearances are applied to the corresponding surfaces.

We then retrieve the transformation matrix of the building predefined objects by the type ('door' and 'window'), and the predefined objects of the same type are retrieved from the database for the same city and placed instead of the previous predefined object using the matrix saved above. The dimensions of the previous opening are temporarily stored to prevent any inconsistencies and applied to the new opening. The standardization of their orientation before export enables the reuse and exchange of their rotations. For openings and especially windows, for each bounding surface of the original building, it is randomly chosen whether or not windows will be attached to it on the condition that at least one bounding surface contains these openings.

5.2 Urban objects and vehicles

Urban objects are generated like buildings, except that they are not decomposed into thematic elements, and are simply composed of polygons and/or predefined items the generation of which has been covered above. They are also generated after the buildings and cover a larger area of the terrain than the latter.

The vehicles are similar to urban objects by their design and extraction from the database. Once all the basic vehicle models are selected, they are randomly generated along both sides of the street, facing opposite directions. Once generated they are animated to move towards the end of the street, and once this goal is achieved, they disappear and are regenerated at the beginning of the street, and so on.

6. RESULTS

Typical buildings of three test cities, each with some common features for their respective buildings and a well-characterized urban style, were inserted into the database : a typical colorful city of Madagascar, another American city with skyscrapers with flat roofs and identical windows and a Norway village with wooden walls, stave and tiled roofs, and wooden windows and doors. The terrain and the street were placed first and buildings and urban objects were generated on areas unoccupied by the street.

6.1 Output

Table 1. Geometric elements from the database

Region	Cities	Buildings	MultiSurfaces	Predened elements	Convex Polygons
Madagascar	1	4	10	52	67
America	1	5	5	195	42
Norway	1	6	20	59	120

Table 2. Thematic elements from the database

Region	Buildings	Urban objects	Vehicles	Bounding surfaces	Openings
Madagascar	4	1	3	25	50
America	5	0	2	25	195
Norway	6	2	2	43	53