IX Symposium on Virtual and Augmented Reality - SVR 2007, p.318-320. Petrópolis, RJ, Maio 2007.

SOLUM: A Library for the Creation of Dynamic 3D Worlds

Felipe F. Quintella
Tecgraf / Computer Science Dept. / PUC-RIO
R. M. S. Vicente, 225 - Gávea
Rio de Janeiro, Brazil
felipe@quintella.com

Alberto Raposo
Tecgraf / Computer Science Dept. / PUC-RIO
R. M. S. Vicente, 225 - Gávea
Rio de Janeiro, Brazil
abraposo@tecgraf.puc-rio.br

ABSTRACT
This paper provides the description of concepts and implementation of the SOLUM Library. The purpose of SOLUM is to create complex dynamic 3D worlds based on a few parameters and definitions, removing detailed and complicated issues on this process. The idea behind SOLUM is to make the whole process of creating a complex 3D enviroment as clear and easy to undestand as possible, liberating application designers and programmers for other tasks.

Keywords
3D Worlds, World Creation, 3D Generation, Dynamic Worlds

1. INTRODUCTION
The creation of large 3D worlds, such as those used in games and virtual communities, can be a difficult and extenuating job. Doing this job by hand usually takes seveal hours of a dedicated team and the costs easily get very high [2]. In order to make this job simpler, some techniques are usually applied. The most common is the reuse of previously-made models. However, this has a major drawback that is the feeling of *'I already saw this before.'*, since you really don't have an identity for the created objects, such as the buildings of a town, for example. There are some random implementation for cities/buildings [5] and terrains [3], but they don't consider the surroundings and can create some strange things like buildings on places they should not be.

The purpose of this work is to solve these problems by creating a library (SOLUM) that works based on a set of parameters defined by the users and then uses a layer model to create the world. The parameters defined by the users are very basic, such as the size of the terrain, the elevations degree, the distribution of communities throughout the virtual world, the resources distribution, among others.

2. USER INTERFACE
One of the modules of SOLUM is the graphical interface, responsible for the configuration of the whole 3D environment. This interface creates the basic configuration file that will power the library, shows a preview of the results, and enables alteration. The project was implemented with C++ for Windows, making use of some support libraries. For the 3D visualization the OGRE (Open source GRaphics Engine) [1] engine was used.

A screenshot of the configuration GUI is shown in Figure 1. This figure illustrates the dialog for the terrain generation. In this dialog, the user indicates the files defining a height map and the textures for the terrain to be generated. Despite this possibility of previously defined input, the interface also provides the random generation of a height map and a texture.

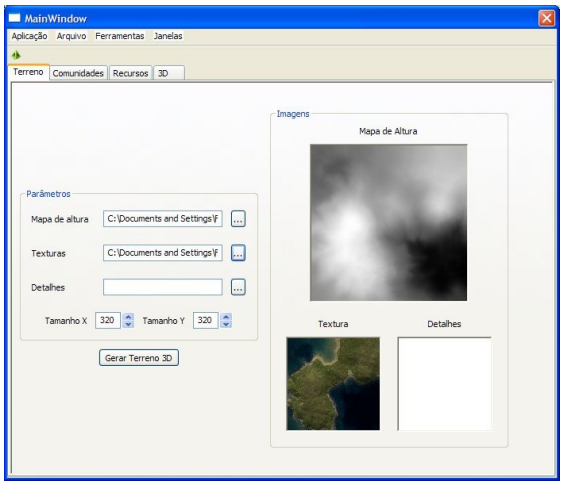

Figure 1: The front look of the GUI system.

In the *communities* tab of the configuration GUI, the user defines the list of communities of the virtual world, as well as their centers and their influence area in the map representing the terrain. The user also defines the possible styles for each community. In the *resources* tab, the user defines all natural resources available by means of a resource map. The user is also required to define the possible materials and textures for each available resource and style, as well as the buildings' floor plans for each style. Finally, in the *3D* tag, the user may pre-visualize the result of the current configurations

and generate the virtual world.

3. ALGORITHMS

The algorithms presentation is split into three layers, namely, *Terrain Generation*, *City Generation*, and *Building Generation*. These layers get information from the configuration file and the lower layers also get information from the higher ones The diagram in Figure 2 explains this.

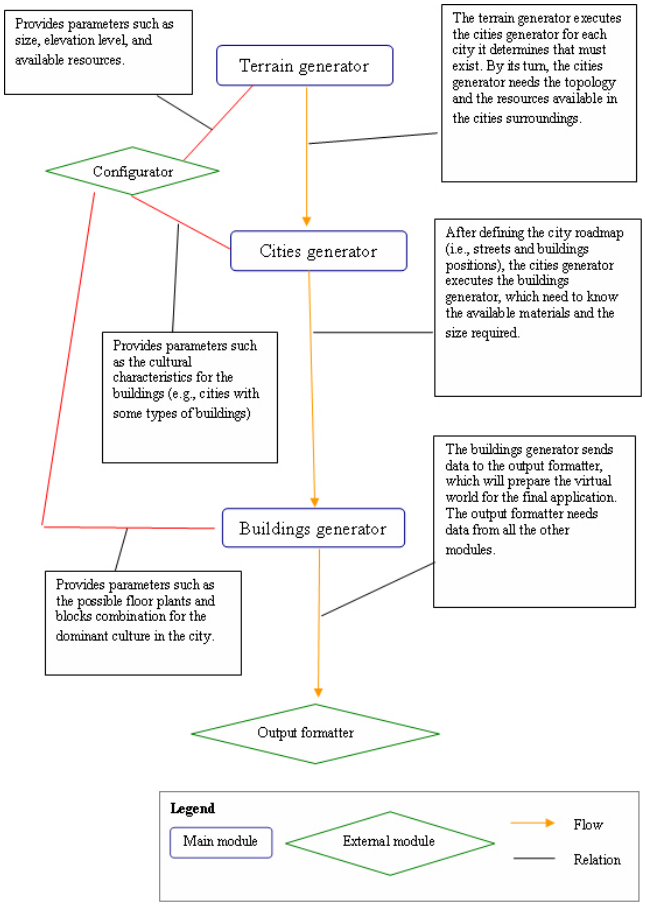

Figure 2: Interaction between the modules

3.1 Terrain Generation

The process of creating a terrain is split into two parts: the creation of the vertex mesh and the creation of the 2D texture for the terrain. To generate the vertex mesh, the hill algorithm was used [4]. The second part of the terrain generation algorithm is the texture generation. This is achieved by creating an image of the same size of the terrain vertex mesh (each vertex will have a pixel for it) and changing the color of the pixel according to the terrain height.

In order to avoid the creation of a very smooth unrealistic texture, we define five height areas separated by 150 points each. Every time a texture is to be generated in a different area, we add or remove 50 in the value of the color for each area. With this approach, we can determine, for example, that the mountain tops above a certain height will be white due to snow, and the deep valleys won't have green. To finalize, on top of the main texture, the user may also define a detailed texture during the world configuration, what provides more realism to the terrain.

3.2 City Generation

The city generation algorithm considers several parameters defined by the user during the world configuration. The location of a city is determined taking into consideration a community map defined by the user during the world configuration (Figure 3). A community may be conceived as a group of cities, such as a country. The library will search in the map for the place with the major concentration of a community. This place is then used as a parameter for the definition of the largest city of a community and the size of the other cities of this community in relation to the largest one. The major concentration point of a community is determined by *i)* the analisys of the initial points of the community, provided by the user during the configuration, *ii)* the influence power of the community (also determined by the user), which will define the area on which the community will expand, and *iii)* the resources available on the terrain, which also affect the community expansion.

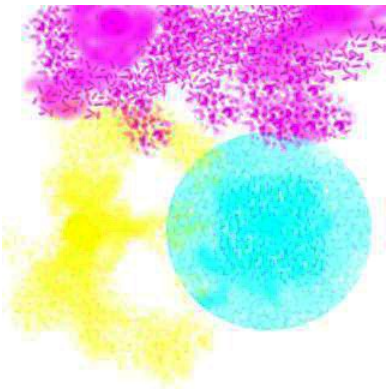

Figure 3: An example of community map. Each color represents one community.

The most influential place of a community will be the place for its capital city. Other cities will be distributed troughout the map taking into consideration the area for this community, a random value and the terrain (cities are more prone to be constructed in plane areas). Each city, depending on its size, has an influence circle that prohibits the construction of other cities in this area. Community by community, the library populates the terain with cities for all communities defined in configuration.

A city might be on the influence area of more than one community. Once the style is determined by the community, a city might end up having a mixed style. The style of a city defines the construction blocks available for the buildings' construction (this information is passed to the next module – the cities generator).

The size of the city is determined basically by two factors: the total points of the main community for that city and the available flat area for its construction in the region defined. Once the size is defined, the construction area is defined. Starting at the city center, the algorithm searches for

adjacent polygons with inclination smaller than 45 degrees (plane region) to determine the city's construction area.

Starting at the center of the city and considering its size, we can go to the configuration file, access the resource map and obtain the resources available. A resource map deals with three colors, representing the main construction resources: wood, iron and stone. Although resources provide guidance for a better city construction, the style of the city prevails in the definition of the construction blocks of the buildings. For example, if a style has only wood construction blocks, the city will have many wood blocks, despite having or not this resource available.

According to the styles defined for the city, we get a list of the main buildings that must exist in the city, and their approximate sizes. Then, we start finding places for these buildings, beginning at the city center. In order to find the places of the buildings in the city we need first to place the main building and then randomly choose the location of other main buildings. This random choice is, however, limited by some rules, such as: a main building must be a random radius away from the other buildings, and must be within the city area.

To define the streets of a city, we connect each building to the main building with a street, and then trace secondary streets at 90 degrees from those main streets. The street tracing continues until we find an obstacle, the end of the city, or another street. Each street must have at least a minimum distance from the previous one and we add a small random factor in order to avoid an extremely regular city.

Figure 4 illustrates the streets of a small city. P1, P2, P3, P4 and P5 are the main buildings, and the green streets are the main streets connecting them. It is important to clarify that the system also considers the terrain topology, trying to connect the main buildings keeping the streets at the same elevation level. The algorithm tries to balance the shortest way against deviations to keep the street plane. Returning to the figure, the orange streets are the secondary ones, perpendicular to the main streets. Again, deviations can be made to follow the terrain topology. The blue streets connect the secondary ones, avoiding "no way out" streets.

Having the streets as guide, the system is able to place smaller buildings. To place those buildings, the system takes each secondary street and walks through it looking for a free space that is big enough for the building. Buildings that we couldn't find a place for will have a street traced just for it on a open space around the city.

3.3 Building Generation

In order to construct the buildings and have access to its wireframe and strucutres, we used the concept of builing blocks. Building blocks are pre-defined 3D structures, which are given to the library during the configuration. Each building block has a style, a resource and a function. To use the building blocks we need a structure we call blue print, which determines the building blocks functions needed to construct a building.

Once we have an adequate blue print for the size and function of the building, we need to get the blocks needed. This is done by sorting by funcion, style and resource the blocks available and then randomizing the result (the style defines the kind of the block to be used, and the resource defines the texture applied to it). This way, having a reasonable number of blocks, buildings should not look the same. Finally, we must have some additional restrictions, for example, the main door of a building must be in the direction of the street, and if we choose a door building block, we need to use the same for all doors. Additional details such as roofs and floor pavings are chosen among those available.

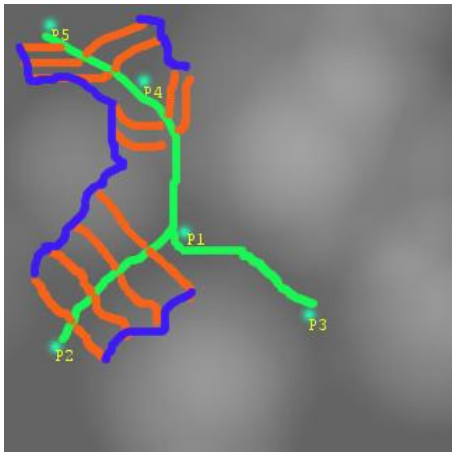

Figure 4: Streets map for a city.

4. CONCLUSION

This paper presented SOLUM, a library for the construction of large virtual worlds composed of terrain, communities and cities. The goal is to develop a basis for a framework that provides the construction of large virtual environments without having to model thousands of 3D objects or to make extensive reuse of them. This construction is based on a relatively small number of parameters defined by the user.

SOLUM is currently adequate for the creation of "aleatory" 3D worlds, such as those used in games. It is our plan to work in the direction of using the tool to create 3D models of real cities. The first step in creating those "real" 3D worlds was implemented for the terrain generator, which can not only generate a random terrain, but also a terrain based on a height map.

5. REFERENCES

[1] *OGRE 3D – Open source graphics engine*, 2006. http://www.ogre3d.org/.

[2] R. Bartle. *Designing Virtual Worlds*. New Riders Games, Indianapolis, Indiana, USA, 2004.

[3] D. Marshall, D. Delaney, S. C. McLoone, and T. Ward. Representing random terrain on resource limited devices. In *CGAIDE 2004 Int. Conf. Computer Games: Artificial Intelligence, Design and Education*, 2004.

[4] B. Nystrom. *Terrain Generation Tutorial: Introduction*, 2002. http://www.robot-frog.com/3d/hills/.

[5] Y. I. H. Parish and P. Müller. Procedural modeling of cities. In *SIGGRAPH '01*, pages 301–308, 2001.