Proposal of a tool for pseudo-infinite 3D virtual world generation

Fernando Bevilacqua
Cesar Pozzer
Universidade Federal de Santa Maria

Abstract

We present a proposal of a tool for pseudo-infinite 3D virtual world generation. The main idea is to split up the problem of generating a virtual world in three main steps: the first one is to generate the paths, the second one is to generate the terrain and the last one is to generate the world elements, like grass over the fields, trees, rocks and plants, for example. The world is divided in cells and only the ones that are visible by the user are kept in memory. To reach this goal, many techniques will be used, like height maps, real-time cities generation, terrain texturing and path-planning.

Keywords:: MMO, Terrain Generation, Virtual World, Path-planning

Author's Contact:

{fernando,pozzer}@inf.ufsm.br

1 Introduction

A game is composed of many elements like sounds, AI, physics and scenario. In multiplayer games, specially in MMO (Massively Multiplayer On-line games), the scenario has an important role, because the player spends a lot of time observing the world around his character.

The virtual world must be interesting, otherwise the game experience can become boring and frustrating. In addition, it's highly recommended a detailed scenario supported by a good history, so the player will be invited to explore all the fields, mountains and lands of the virtual world. One example of such things are the MMO World of Warcraft [Blizzard Entertainment 2007], where the player can explore a huge virtual world, organized in at least two different continents. Over the virtual terrains of the game, the players can view and interact with forests, mountains, fields, farms, deserts, and more, all these elements binded to a complex history of magic and epic fights.

The creation of such a complex virtual worlds is a very important task in the development of a MMO. The use of tools to generate the virtual landscape can be useful to developers and game designers, which can spend more time planning world details and history than modeling mountains or fields of the game, for example. Concerning about this subject, this paper describes a new proposal for an automatized virtual world generation tool, which aims to produce big and complex virtual worlds, but still caring about the game logic, such as path-planning and distribution of key elements, like gold mines in a strategy game.

2 Related work

This section presents four related works about creation of virtual worlds. The first three are focused on generation of virtual scenarios, while the last one is centered in making those generated scenarios look more realistic.

2.1 Height maps to generate terrains

A common technique for terrain generation is height maps, which are bidimensional arrays with each dimension representing the X and the Y axis, and the pair (i,j) in this array represents the point's height in the Z axis [Ribble 2001]. The main idea behind this form of terrain generation is to introduce controlled noise into a flat surface, which will produce terrains like the ones found in real world.

One method of height maps generation is called fault formation [Ribble 2001]. In this method, at every step, a line is traced in a random position over the map, and then the height of all points in one side of the line is increased. Another example of height maps are those generated through midpoint displacement, which can be used so simulate the effects of tectonic plates in the earth surface [Ribble 2001]. The method splits up the surface into small segments, then rise or lower the height map by a random amount in each segment (using the midpoint). After several steps, the terrain contains height variations over the surface, as shown in Figure 2.1.

Figure 2.1 Midpoint displacement method in a 2D view [Ribble 2001]

Another method of height maps generation is called particle deposition [Ribble 2001]. This method simulates a lava flow using a simple particle system, where the elements are dropped by specific sources located in random positions. The sources drop a bunch of particles during a random time, what can be controlled to produce higher or lower heights over the map. When the particle is dropped, it "falls" into a point in the height map, then it tries to reach a lower level (e.g. a neighbor point with lower height). The movement is made in all direction in a random way. If the particle reaches a point with no low height neighbors, it ends the movement. Figure 2.1 shows a terrain generated by particle deposition method.

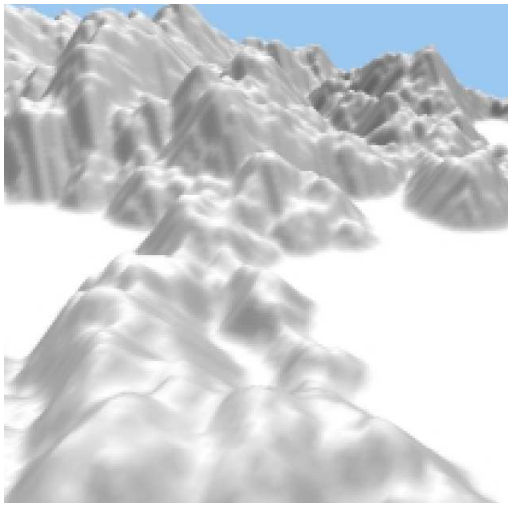


Figure 2.1 Terrain generated by the particle deposition method [Ribble 2001]

2.2 Real-time generated city

Another related work is the real-time generated city proposed by [Greuter et al. 2005]. In that work, the authors present a system able to generate pseudo-infinite cities that can be interactively explored in a first person view. All elements in the virtual city (like streets and buildings) are generated in real-time, as they are encountered by the user. The geometric form of each element is determined by its position, what makes possible to the user see the same buildings again in a specific place he walked before, no matter how far he walked around since then.

To reduce the amount of resources needed, the system keeps in memory just the elements that the user is able to see. As the user walks, the elements far away from the user view are released from the memory; the inverse process is performed to new elements in

the user's view, which are loaded into the memory as they appear to the user. This process can ensure a rational and acceptable memory use, even with a pseudo-infinite city with huge boundaries.

The management of the elements within the camera's view is called *view frustum filling*, according to the authors. In that approach, the world is splitted up in many square cells on a 2D grid, each of them with geometric elements, such as buildings and/or streets. The cells are arranged in square loops around the cameras position. To be part of the user view, the cell must be located within a 120° viewing angle and a distance of *loops X cellsize*. Figure 2.2 shows the visible cells in the users view.

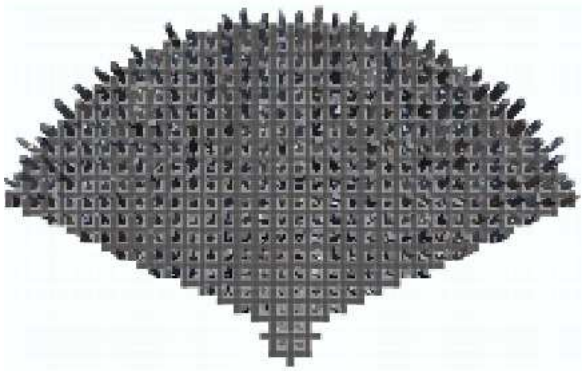
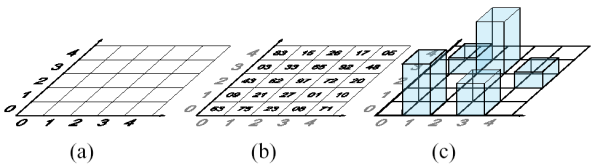


Figure 2.2 Visible cells in the users view [Greuter et al. 2005]

The form and appearance of each building in the world are determined by a 32 bits pseudo-random number generator. Each cell has its own 32 bits hash, generated by hashing the cells coordinates `X` and `Z` with a global seed. The mathematical formula is:

```
seed = hash(x XOR hash(z XOR citySeed))
```

where `hash()` is the 32 bit Mix Function proposed by Thomas Wang [Wang 2000]. All elements in an arbitrary cell are generated by a random processes seeded by the cells hash. In that way, a cell will have always the same elements, no matter how far the user walked around. The Figure 2.2 illustrates the process.



(a) (b) (c)

Figure 2.2 a) 2D grid (b) hashed seeds (c) generated buildings [Greuter et al. 2005]

2.3 AdVantage Terrain Library

The AdVantage Terrain Library (ATL) [Strugar 2007] is a tool designed to generate large virtual terrains, intended to be used in massively multiplayer online games, flight simulators and similar applications. The ATL was written in C++ and is distributed as .dll or .lib file, so it can be easily integrated into C++ applications.

The library is able to generate terrains from huge height maps, using huge textures (64000x64000 pixels or bigger) as well. As stated by the ATL author, some features of ATL are: fast rendering techniques, collision detection support, multithreaded data streaming/decompression and platform independent.

The ATL Win32 SDK is free of any charge for commercial or non-commercial purposes. Figure 2.3 shows a terrain generated by ATL from a height map of 24577x15233 pixels.

3 Mapping textures

Mapping textures on virtual terrains is important to make the landscape interesting to the end-user. As highlighted by Sinvhal [2005], terrain texturing in computer games, for example, doesn't have to

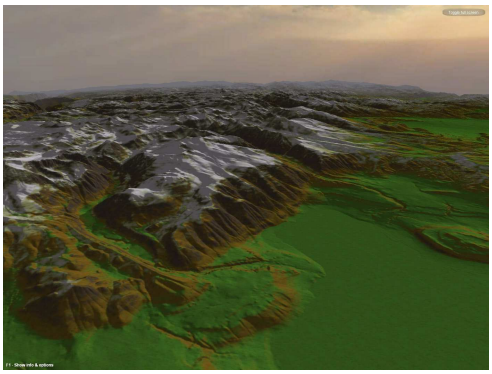


Figure 2.3 Terrain generated by AdVantage Terrain [Strugar 2007]

imitate the real world perfectly, it only have to be "believable" to the user. A non-realistic textured terrain can be more interesting then a realistic one, which will result in a better game experience.

In the work of [Sinvhal 2005], cellular automata are used in the automatic generation of textures for large surfaces. As defined by [Weisstein 2007], a cellular automaton is a collection of "colored" cells on a grid of specified shape that evolves through a number of discrete time steps according to a set of rules based on the states of neighboring cells. The work of Sinvhal [2005] presents a research about elements that make a computer generated texturing for a terrains seems to be realistic. Some of those elements are:

1. Natural terrains have strong characteristics, like the large area of sand in beaches;
2. Landscapes have random elements that make them seems to be natural;
3. Characteristics can be influenced by height, like the snow in higher elevations in a mountain.

To reach those elements in the terrain texturing process, the author uses a cellular automaton with a set of specific characteristics; the initial population in the cellular automata evolves over many steps to simulate a natural environment. To avoid a texture pattern, randomness is used. The features produced by height (like snow in mountains) are made through a probabilistic distribution of texture tiles for each state of the cellular automaton, according to the terrain height.

Figure 3 shows a textured terrain resulted from cellular automata proposed by [Sinvhal 2005]. Height elevation areas have different texturing pattern then lower height areas.

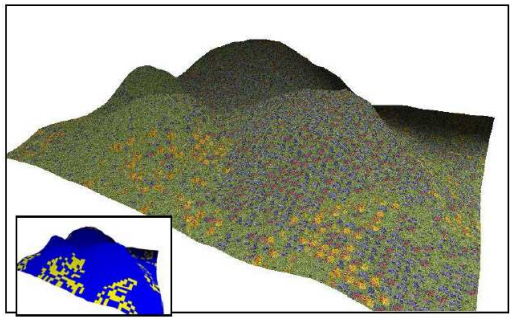


Figure 3 Textured terrain resulted from cellular automata texturing process [Sinvhal 2005]

4 Proposal of a tool for virtual world generation

As explained before, the task of generating a complex virtual world does not rely on terrain creation only, but also on other elements. Generate a random terrain with height variation is not enough to make it useful to be used in a game, for example. In a game designer view, the player hopes to see a world filled with paths, cities, secret

places, and so on, elements that can keep the player's attention for a long time. In that way, the generation of a complex virtual world requires a method able to create, in addition to terrains, a number of extra elements that together can produce a rich environment.

This proposal is about a tool able to generate virtual worlds, using for that most of the techniques explained in this paper and also new ones, like path-planning. The main idea of our proposal is to split up the problem of generating a virtual world in three main steps: the first one is to generate the paths, the second one is to generate the terrain and the last one is to generate the world elements, like grass over the fields, trees, rocks and plants, for example. As far as we have researched, no other tool was designed to create such a complex virtual worlds using terrain generation (with height variation), addition of cities and path-planning to ensure a way to reach every place in the world.

This approach of splitting up the world generation in three main steps is a way we have found to solve some problems related to terrain generation. One of them is about paths and height variations: we have decided to generate the paths first because it can simplify the task of generating height variations in the world surface. If we generate the height variations first (mountains and so on), then we have to scan the map looking for possible paths, but there are no guarantees we could find adequate paths (we could find no paths in the worst case). Generating the paths in the first step make us able to know where a big mountain can be placed, for example; we are also able to know if a mountain is blocking a path, so we can change the mountain height in that point, so it will no longer block the path already defined.

To reach this goal, the virtual world will be divided in cells, like the approach made by [Greuter et al. 2005]. In the first step, the tool will calculate the cells the user (e.g. player) is able to see, using a view frustum approach; for each of those visible cells, the tool will generate a set of graph nodes and then use path-planning to connect them to create the paths. Each cell will have five control nodes, which will define the path configuration for that cell. One of those nodes is called *origin node* and it will be used as the beginning of all paths in the cell. The other four nodes are called *binding nodes* and each of them will be located near a specific side of the cell. A small group of the generated nodes, called *boundary points*, will be placed in the cell boundary. The Figure 4.1 shows three arbitrary cells and their origin nodes (o), binding nodes (i) and the boundary nodes (u).

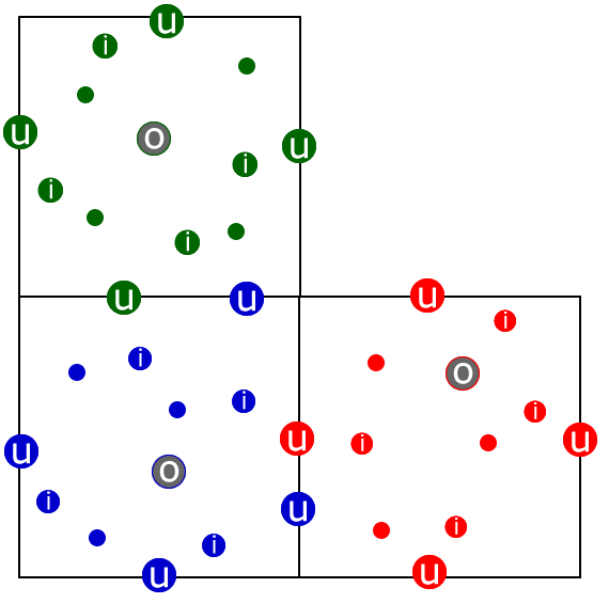


Figure 4.1 Three arbitrary cells and their origin nodes (o), binding nodes (i), boundary nodes (u) and the generated paths

After the nodes generation, the visible cells will present a set of graph nodes, but no paths. At this point, the tool will start connecting the nodes to create paths, so the user can walk over the visible cells. This process will be performed using the binding nodes: for each side of the cell the user is located (called *focus cell*), the tool will search for neighbor boundary nodes; the binding nodes of the focus cell will be connected to the searched neighbor boundary nodes, resulting in a set of new paths. In the end, the focus cell will have a set of own paths (generated from the origin node) and also a set of shared paths (originated from its own binding nodes and the neighbor boundary nodes). Figure 4.2 shows the paths originated by the connection between the binding nodes of the focus cell with the neighbor's boundary nodes (dotted lines) and the paths generated by the connection between the inner nodes of each cell (non-dotted lines).

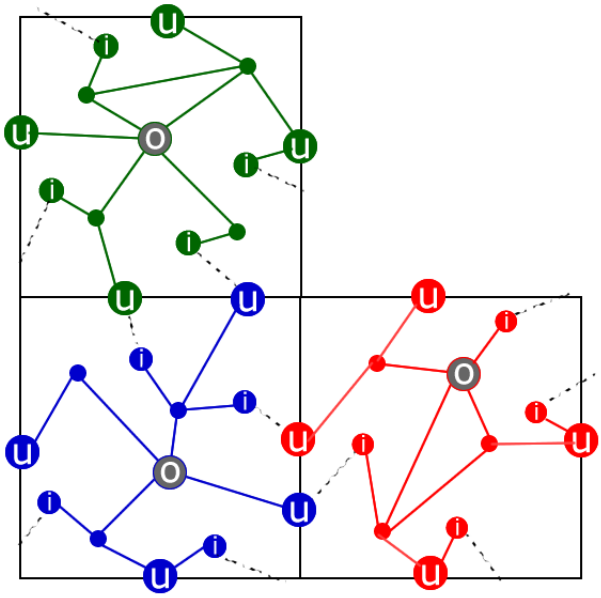


Figure 4.2 Three arbitrary cells and the generated paths used to connect them

The use of binding and boundary nodes are required because we do not want a recursive process to determine the graph nodes (and paths) of the visible cells. If those special nodes are not used, every cell must know where the neighbor nodes are, because all nodes have to be connected to create an uninterrupted path. Our idea of binding and boundary nodes tries to avoid recursive information requests between the cells. An example of recursive information request is when the focus cell asks cell `A` to inform its paths: `A` has to ask the same information to its neighbor cells, because it must find out where the neighbor nodes are to be able to generate its own paths. When `A` asks information, all asked cells also have to request information from their neighbor cells to be able to generate their own paths, and so on, in a recursive process, until a cell with no dependencies is found (a cell able to generate its own paths without asking any other cell).

The user will be able to explore all the places in the virtual world using the graph route or not. The paths resulted by the graph nodes will ensure a way to reach all places in the world, but the user will still be free to walk in places with no paths. The only restriction for that is the terrain height: the user cannot across a big mountain, for example.

The second main step is to generate the terrain. Using a combination of techniques described by [Ribble 2001], the tool will generate mountains and other height variation over the surface of each cell.

The last main step is to generate the world elements (like trees and cities) and to texturize the generated content in all the steps. All elements of this step will be added to each cell through specific function calls, like `addTrees()` and `addCity()`. All of those functions will work as a combination of random actions (seeded by the cell's hash) and statistical calculation, like the texturing process

proposed by [Sinvhal 2005]. Using the city generation as an example, the function `addCity()` will use all information available in the cell and its neighbors to determine the *city score* for the cell. The city score is largely calculated by the cell's height average and by the cell's hash. Another element used in the score calculation is the average city score of neighbor cells. If the cell fits the city score required, then a city will be added to that location. As a result, if a cell has a very high city score, then probably all of its neighbors will have such high score too. If a neighbor cell has a bad city score, then it will decrease the average to the city score of its neighbors, so they will have a smaller probability to become cities. If the neighbors of the neighbors have a bad city score too, then the average city score will be decreased again, what will result in no more city cells around the first cell with high city score.

5 Conclusion and future work

A proposal of a tool to generate virtual worlds with height variation, path-planning and texturing through cellular automata has been presented. To support that tool, height variation techniques were presented, as well as related work about virtual world generation. The main idea of this tool is to split up the problem of generating a virtual world in three main steps: the first one is to generate the paths, the second one is to generate the terrain and the last one is to generate the world elements.

The creation of a virtual world like the one used in World of Warcraft is a very hard task to accomplish. In addition to height variation, all surfaces and terrains need textures to seem interesting to the player. Extra elements are also required (e.g. trees, rocks) and a geographic features must be present (like snow in higher elevations of a mountain). A tool to generate virtual worlds in an automated way, with all those related elements, can help in game development.

All explained steps in this proposal are our first ideas to solve the problem of generating a complex virtual world. Our next steps will be modeling all C++ classes required for the tool, starting with the classes related to the main steps one and two: generate paths and height variation. After that, we will start planning and coding the classes related to the main step three that is about generation of world elements, like texturing, cities, rocks, trees, and so on.

References

BLIZZARD ENTERTAINMENT, 2007. World of warcraft. Available at: <http://www.blizzard.com>.

GREUTER, S., PARKER, J., STEWART, N., AND LEACH, G., 2005. Realtime procedural generation of 'pseudo infinite' cities.

RIBBLE, M. 2001. Using various techniques to generate height maps used in terrain generation. Tech. rep.

SINVHAL, S. 2005. *Mapping Textures on 3D Terrains*. Master's thesis, Texas A&M University, Texas.

STRUGAR, F., 2007. Advantage terrain. Available at: <http://www.advantageterrain.com/>.

WANG, T. 2000. Integer hash function. Tech. rep. Available at: <http://www.concentric.net/ Ttwang/tech/inthash.htm>.

WEISSTEIN, E. W., 2007. Cellular automaton. Available at: <http://mathworld.wolfram.com/CellularAutomaton.html>.