ARENA - Dynamic Run-time Map Generation for Multiplayer Shooters

Bhojan Anand and Wong Hong Wei

School of Computing, National University of Singapore
banand@comp.nus.edu.sg, wei@comp.nus.edu.sg

Abstract. In this paper, we present simple and novel method to procedurally generate the game maps for multiplayer shooter games faster (in order of seconds) without compromising the expected features of a good multiplayer shooter environment.

Keywords: games, procedural content generation, game map generation

1 Introduction

1.1 Procedural Content Generation

Procedural Content Generation (PCG) is the use of algorithmic means to create content [17] [19] dynamically during run-time. Instead of trekking the same grounds which gets stale with time, PCG promises a more novel experience every playthrough. PCG was utilised in games as early as 1978. A notable example is *Rogue* [12] [1] which spawns a new genre known as Roguelikes. Core features include randomly generated levels, item locations and so on. PCG's influence extends to racing games like *Gran Turismo 5*, which procedurally generates its tracks [13]. First Person Shooter (FPS) *Borderlands* series procedurally generates weapons [5].

1.2 Multiplayer Shooter

We believe that an exception to PCG lies with environments/maps for multiplayer shooters such as Battlefield [4], which remains one of the most popular genres to date [18].

The reader should note the difference between a multiplayer and a standard shooter, which often follows an approximately linear path. The competitive nature of multiplayer shooters brings additional challenge of ensuring fairness through the positioning of strategic points. Moreover, players typically have more freedom to move around the map. It is for these reasons the designers often take a different approach to craft multiplayer maps.

We limit our scope to the game mode **Capture and Hold**, popularised in games like Killzone [6] . Every player belongs to one of 2 teams. Littered around the map are *flags* that are captured by placing players in close proximity. A captured flag (can be recaptured) lowers the *team points* for the opposing team continuously. A team point of 0 signals the victory of the opposing team. To win, teams have to actively defend and pursue flags.

In this paper, we present a novel method that can generate dynamic maps almost as soon as the player press the 'Play' button, while ensuring the quality. The method is simple, practically feasible and we have implemented and evaluated it with a test game. In the rest of the paper, we first discuss the motivations for this work in Section 2 and then related works in Section 3. In Section 4 we describe the design goals of map generation method. Section 5 explains the design of the map generator. Implementation and evaluations are described in Section 6. Finally we conclude the paper with the summary at Section 7.

2 Motivation

In this work, we are attempting to automatically create playable, balanced (fairness) and interesting maps for multiplayer shooters, with a novel approach built using Search-based PCG [17]. While PCG is used by some multiplayer shooter game designers, who would procedurally generate maps and then manually tweak them to ship with the final product, our goal is to remove the human intervention for manual tweaking completely. This means that the generation should be completed within a span of seconds, or else the patience of the player could wear thin. If we are successful, the development time and cost needed to create maps for similar games could be drastically reduced. The result would be increased longevity that stems from the near limitless amount of maps for players to play in.

3 Related Work

Güttler et al. [7] identified some basic spatial properties of multiplayer FPS games and proposes several heuristics for better level design. In addition, the insights provided by several industry leaders of leading game companies on design of a good multiplayer game ([14], [9] and [8]) are incorporated in formulating our design goals. Search-based PCG (SBPCG), an approach to PCG, was introduced by *Togelius et al.* [17]. We will be employing a similar approach in our solution. *Togelius et al.* also managed to procedurally generate tracks for a racing game [15] and maps for strategy game Starcraft [16]. In both cases, SBPCG was used with a simulation based fitness function. *Kerssemakers et al.* [11] introduced a procedural PCG generator to generalise the creation of PCG to games again with the use of a simulation-based fitness function. However, the use of simulation-based fitness function is not suitable for our goals, due to the time it takes to create a map is long and it is not suitable for practical implementations due to the strict real time requirements imposed by the games and game players. Work done by *Cardamone et al.* [3] to evolve interesting maps for a FPS leveraging on SBPCG is a great starting point for our research. However, a great amount of work have to be done to ensure that the map can be generated in a span of seconds. Moreover, the maps that are generated are seemingly low on navigability and aesthetics, which are basic features of any good multiplayer game. In contrast, navigability and aesthetics are part of our design goals.

4 Map Design Goals

We describe what we want to achieve by identifying elements of interesting maps (of multiplayer shooters) so that they can be incorporated into our design. **1) Fast Generation** - We wish to generate the maps in real-time. In other words, the final map has to be

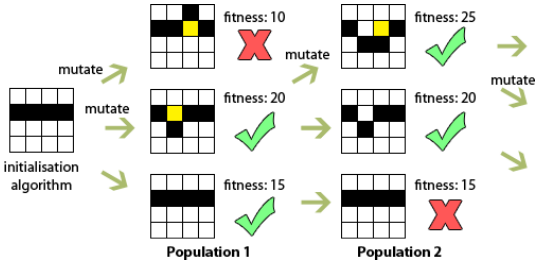

Fig. 1: Illustration of a SBPCG process. Only accepted maps are used in the next population.

generated within a span of seconds to minimise player frustration. **2) Collision Points** - *Güttler et al.* [7] defines collision points as areas that see the most clashes and where most tactical choices are made. Tactical choices begin with preparations (route to take and so on), and ends with a confrontation. The designer should be aware of them to give more opportunity for tactical choices [8]. In contrast, a map with no clear or too many such points are likely to see players stumping onto opponents unexpectedly. **3) Flow** - The designers in [9] emphasise on *flow*, an "invisible flow (that is) continually impelling the player onwards". As this is too abstract, we deconstruct it into measurable components (Navigability and Pacing). Navigability - *players should be able to recognize where they are and where they should go*. Pacing - *confrontation should last enough duration to be fun. It should be accompanied by some respite, but not to the extent of inducing boredom [7]. Also, the map should not have disruptive dead ends*. Even though they may not fully encompass it, we have observed that they provide reasonably good flow and serve as good starting point for future research. **4) Fairness** - Each team should have same chance of victory [7], which is related to flags. A team with flags closer to its spawn point has a higher chance of capturing them. **5) Aesthetics** - The design must have the potential to meet aesthetics demand of players. There is a rich set of taxonomy that collectively define aesthetic [10]. For instance, a map cannot constitute entirely of blocks. Instead, it should contain trees, vehicles and so on to create more natural challenges. In addition, having diverse items improve navigability, as players can use them to get their bearings.

5 Map Generator Design

In this section we describe our algorithm to generate interesting maps for multiplayer shooters, which makes use of the popular **Search-Based Procedural Content Generation** (SBPCG) [17] method. We employ *generate-and-test approach* as illustrated in Figure 1.

We first present how a map ***blueprint*** is created by our initialisation algorithm in Section 5.1. Then, we detail how it evolves with a fitness function to produce the final map blueprint and how it is mapped to the actual game environment in Section 5.2.

5.1 Initialisation Algorithm

Our initialisation algorithm consists of several phases. We first populate the blueprint piece by piece before determining where to place strategic points and computing its fitness.

PHASE I: Populating Game Tiles. Each game tile comprises of four smaller **cells**, which belong to either an *indoor* area (inside a building), *outdoor* area or *inaccessible*

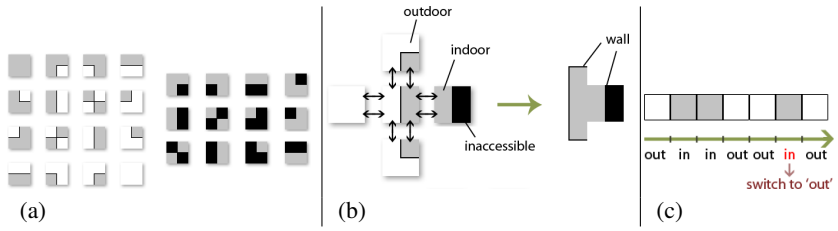

Fig. 2: a. Types of Game Tiles (not complete), b. Rule of Adjacent Game Tiles, c. Removing artifacts horizontally

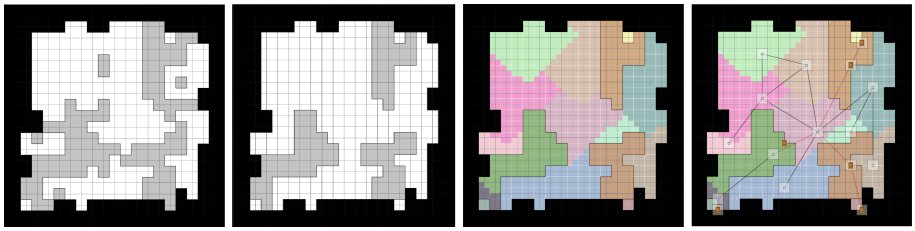

Fig. 3: Initialisation Algorithm Phase I-IV. Each colour represents a region, and the brown icon is a door.

area as shown in Figure 2a. A game tile can be placed in a location adjacent to another game tile only if the neighbouring cells match (Figure 2b). We call this adjacency requirement as *Rule of Adjacent Game Tiles*.

The map uses a grid layout, with each 'square' occupied by a game tile. The grid is first enclosed by a layer of fully inaccessible game tiles. This ensures players cannot move outside the map. Next, game tiles of random types are placed in unoccupied 'squares', constrained by the rule of adjacency (Figure 2b). This is repeated until it is fully filled. An example of the generated map can be seen in Figure 3.

PHASE II: Cleaning Up. We notice several artifacts like overly small buildings or protruding parts of buildings' remains. We scan the map blueprint both vertically and horizontally, removing single cells which are surrounded by cells of different types as depicted in Figure 2c. This is necessary to give the map a 'cleaner' look with more regularly shaped buildings which would otherwise hurt its navigability.

PHASE III: Identifying Regions. To analyse the map, the algorithm requires an understanding of its layout. We first identify **regions** of the map, which are either *indoor* or *outdoor*. Region detection is done by applying a *Flood Fill* algorithm to the cells, stopping when it reaches its maximum size or when no cells are left. This is repeated until all accessible cells belong to a region. The maximum size is proportional to the map's size and can be tweaked. Note that cells of different types cannot share the same region; Indoor region contains only indoor cells and outdoor region contains only outdoor cells.

PHASE IV: Connecting Regions. By identifying regions, we can construct an undirected *graph* with each *node* located approximately at the region's centre. Nodes are connected by an *edge* if players can move directly from one to the other without passing through a third one. At this stage, no edges exist between indoor and outdoor regions. For every indoor region, edges are created to connect to an neighbouring outdoor region. This adds a *door* between them and improves the connectivity of the map. The same

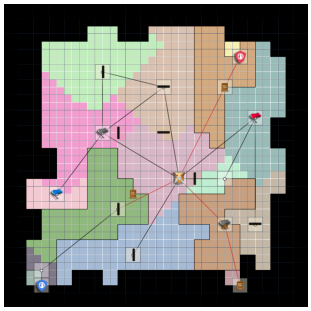

Fig. 4: Map blueprint after Phase V. The shields are the spawn points. The flag icons show the position of the flags. The shield and sword icon near the centre represents a collision point. The thicker lines are covers.

Fig. 5: Visual Appearance of the Game Environment

applies to *all* indoor regions of bigger buildings with multiple regions. It is therefore natural to have multiple doors at different points within it. Otherwise, moving deeper into a large building always result in a dead end. Most parts of the generated map is connected after this phase. However, there are exceptions where regions are fully surrounded by inaccessible cells.

PHASE V: Positioning of Strategic Points. Spawn Points, position of Flags, Collision Points and Covers are strategic elements which are depicted in Figure 4. Two **Spawn Points** are required for 'Capture and Hold' games, one for each team. They are positioned by identifying the two nodes of the graph that are the furthest away from each other geographically. Four **Flags** (*team flags*) are planned for the map. Each team is assigned a flag (captured) at the beginning of the play. They are positioned at the nodes closest to the teams' spawn points. The two remaining flags are neutral (not captured). A straight line is first 'drawn' between spawn points. Each side of the line will contain a neutral flag. They bound to a node with the minimum difference in travelling distance (using the *Dijkstra algorithm*) from both spawn points to preserve fairness. **Collision Points** are identified by the *degree* of the node. A high degree node most likely belongs to a region that players are prone to meet as many different routes will lead to it. We observed that a degree of *five* and above makes a good condition for a collision point. **Covers** are placed approximately at the meeting point of regions that have an edge to the collision points. This helps with promoting tactical choices as it provides more options to players who are more likely to meet at collision points. Covers are also placed at nodes which have not been assigned anything. Since these nodes are usually at region centres, the covers act as good places to hide if a firefight is to break out at that region. Without it, there may be too many open areas.

5.2 Evolution

Fitness Computation A fitness is assigned for the evolutionary algorithm to tell how good the map is. There are 3 approaches [17]. *Interactive Fitness Function* grades it based on interaction with a player. As this is physically impossible, this approach is not considered. *Simulation-Based Fitness Function* uses AI (Artificial Intelligence) agents to play through a portion of the game for evaluation. Many research works ([15], [11] and [3]) used simulation-based fitness functions. However, its weakness lies in the time required to compute it.

Therefore, we will be employing *Direct Fitness Function*, where a content is judged by retrieving a list of features. In our solution, the fitness is computed by simply summing up the values for all of the following features which are derived based on the design goals presented at the beginning of this Section. 1) **Connectivity** - Returns 1 if the graph is connected (as detected in Phase IV). Otherwise, returns 0. All regions are reachable in a connected map. 2) **Forced Collision Points** - Returns 1 if there are one or two collision points. Returns 0 if there are zero or more than two collision points (no clear collision points). Ideal number of collision points depends on the map size. 3)**Flag Fairness** - Difference in the distance travelled to own team flag. This is measured by finding the travelling distance from the spawn points to the corresponding team flag. The returned value is normalised to 0 to 1, with 0 being maximum distance apart and 1 being approximately the same distance apart. 4) **Overall Flag Fairness** - Similar to Flag Fairness, but returns the difference in distance travelled to all flags from the spawn points instead.

Evolution and Mapping The first population consists of three map blueprints generated with the initialisation algorithm. They are then mutated three times each, producing nine more blueprints. Each mutation removes part of the map and repopulate it with the initialisation algorithm. With that, the population is complete with twelve map blueprints. Three of the best maps (based on their fitness values) are picked and mutated three times to form the second population. This continues until enough populations are processed. The final map blueprint obtained after the evolution is then used to create the actual environment. Doing so is a direct mapping of each abstract game tile (in blueprint) to one of the many variations of concrete and matching game tiles seen by the player.

6 Evaluation

6.1 Evaluation Methodology

We developed a FPS game to implement our solution. Video demonstration of our algorithm and playable version of our game are available at our project homepage [2]. We evaluated the effectiveness of the evolutionary algorithm on producing good maps and its compliance with the design goals.

6.2 Effectiveness of Evolution

We created *3 different maps* using our solution. We run the evolution algorithm for 55 populations (approximately 10 seconds in a Intel Core i7 laptop) each time to generate the map. The maximum average summed fitness is 4.0, as the fitness for each of the 4 features is normalised from 0.0 to 1.0. The *average summed fitness* of the processed population against the *number of populations processed* so far is plotted in Figure 6. The positive gradient shows that the evolutionary methods does improve the quality of the map as more population are generated and mutated. It also shows that the fitness stabilises after 26 populations (which takes about 5 seconds).

The fitness for each feature (with unnormalised values) of generated maps are shown in Table 1. Based on the fitness function that we have defined, the maps are all connected (with *Connectivity* of value 1.0), meaning that no regions are blocked from the rest. The algorithm is also effective in constraining the number of collision points (with

Forced Collision Points of value 1.0, indicating that there are either 1 or 2 collision points.). For both *Flag Fairness* and *Overall Flag Fairness*, we show the unnormalised values. These values indicates the absolute difference in moving from the spawn points to own team flags for Flag Fairness, and the difference in moving to all flags for Overall Flag Fairness. A value of 1.0 means their distance are exactly one cell apart. Given our result, the values are very low, with the highest being 1.03510, which is barely one cell apart. To give a clearer perspective, one cell will take only approximately a second to travel. Henceforth, we can conclude that the flags are placed in positions that are largely fair for both teams.

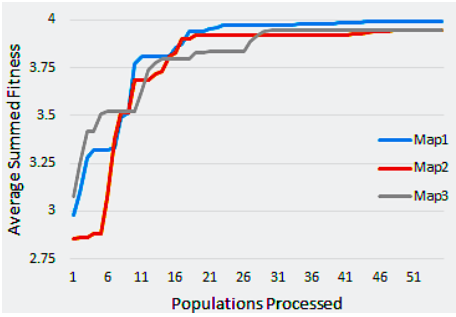

Fig. 6: Change in fitness when evolving

	Map 1	Map 2	Map 3
Connectivity	1.0	1.0	1.0
Forced Collision Points	1.0	1.0	1.0
Flag Fairness (unnormalised)	0.3246	1.0351	0.4806
Overall Flag Fairness (unnormalised)	0.0031	0.3103	0.5568

Table 1: Fitness values of generated maps (after 26 populations)

6.3 Meeting Design Goals

A user study was conducted to test if the map generated using our method meets the design goals. The game was ported to run in a browser environment. The URL of the game with a starting page containing the introduction to the game, game rules and mechanics were sent to all participants. The users were allowed to play the game multiple times before taking the survey. A brand new map is generated with every playthrough. Table 2 shows the demography. The survey had 6 questions with Likert scale of 1 to 5 for each question. The questions: Q1) How fast is the loading process? [1 being unacceptably slow and 5 being very fast] Q2) Did the combats took place all over the map or in few key locations? [1 being only sparsely located and 5 being focused on a few areas] Q3) Was it easy to navigate your way to your opponents and flags? [1 indicating very easy and 5 being very hard] Q4) Did you run into many dead ends? [1 indicating none and 5 indicating many] Q5) Please rate the pacing of the game. [1 being too fast/slow paced and 5 being very well-paced] Q6) Did the placement of the flags give fair chances for both teams? [1 being very unfair and 5 being very fair]

The results shown in Figure 7 are discussed below. **1) Fast Generation** - As discussed above it took about 10 seconds (10.1s, 9.9s and 11.5s respectively for Map1,Map2 and Map3) to generate the maps with 55 populations while 26 are sufficient. We feel that this loading time is reasonable for a commercial game. The mean user score of 3.19 with a standard deviation of 1.08 for Q1 implies that the users are generally acceptable of the time taken to generate a map. In contrast to algorithms in previous works [3], [15], [16] which takes hours, our algorithm is relatively successful in real-time generation. Previous works typically take a highly random approach in initialising the candidate,

Gender	Female (5), Male (48)
Proficiency Level in Games	Never Played (1), Novice (10), Average (26), Expert (16)
Frequency of Playing Games	Never Played (2), Few Times a Month (14), Few Times a Week (16), Almost Every Day (21)
Played any FPS game before	Not sure about the game type (2), No (5), Yes (46)

Table 2: User Study - Demography

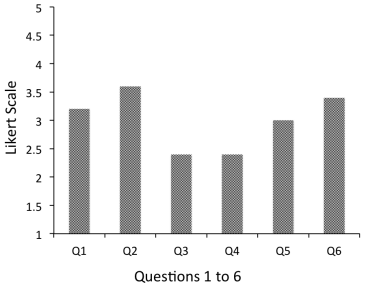

Fig. 7: User Study Results

and relies too heavily on evolution. The key difference lies in our relatively complicated initialisation process, which takes additional steps to improve fitness of base maps. **2) Collision Points** - The sample mean is 3.60 with standard deviation of 1.25 for Q2 implies the design of collision points are effective as we have successfully limited the points of confrontation. The covers surrounding it provides ample opportunities for tactical choices. **3) Flow** - From the results of Q3, Q4 and Q5, we can say that our maps have fairly good navigability and pacing. Even with measures to prevent dead ends, they remain in some areas that are hard to detect. The buildings on the left and right are large but do not have doors other than the ones that leads to the centre. Players may find themselves trying to find flags deeper into them only to discover a dead end. This can be improved by tweaking the maximum region size. Pacing is susceptible to many confounding factors, such as the speed of movement, rate of fire, speed of reloading and so on, and is therefore not exclusively dependent on the map design. **4) Fairness** - We measure the time to navigate from both spawn points to all four flags by playing through the map. It takes 58.88s to move from the blue base and 60:21s to move from the other base. The small difference between these values implies high fairness. The sample mean of 3.36, standard deviation of 0.98 for Q6 implies that the placement of flags are largely fair for our users. **5) Aesthetics** - The look of the game depends on the artist. In general, However, what we can observe from the map is that the buildings have decent shapes and the objects such as trees and wagons are placed naturally as shown in Figure 5.

7 Conclusion

We first presented our findings of what constitutes a good game environment for multiplayer shooters, before designing and implementing an algorithm for the procedural generation of a map that can satisfy all the criteria. We then evaluated the maps by measuring the effectiveness of the evolution to produce better maps and to check whether the result satisfies what we have set out to do. Our evaluations show that we can generate game map procedurally for multiplayer shooters in less than ten seconds without compromising the common design requirements (flow, positioning of collision points, fairness and aesthetics) of commercial multiplayer shooters. Hence, in contrast to previous works, our method is practically feasible for run-time dynamic map generation and can be immediately used by game industry.

References

1. Rogue 1984 - the dos game, the history, the science. `http://science-fiction.fch.ir/rogue/doc/Rogue_1984-The_DOS_Game-The_History-The_Science.html` (Mar re-

trieved 2014)
2. Anand, B., Wei, W.H.: Arena- procedural map and music generation for multiplayer shooters (project page). http://www.comp.nus.edu.sg/~bhojan/arena
3. Cardamone, L., Yannakakis, G.N., Togelius, J., Lanzi, P.L.: Evolving interesting maps for a first person shooter. In: Proceedings of the 2011 International Conference on Applications of Evolutionary Computation - Volume Part I. pp. 63–72. EvoApplications'11, Springer-Verlag, Berlin, Heidelberg (2011)
4. DICE: Battlefield 4 multiplayer. http://www.battlefield.com/battlefield-4/gameplay/multiplayer (Jan retrieved 2014)
5. Gearbox: Borderlands 2 information. http://www.borderlands2.com/us/#info (Jan retrieved 2014)
6. Guerrilla: Killzone home. http://www.killzone.com/en_GB/killzone.html (Jan retrieved 2014)
7. Güttler, C., Johansson, T.D.: Spatial principles of level-design in multi-player first-person shooters. In: Proceedings of the 2Nd Workshop on Network and System Support for Games. pp. 158–170. NetGames '03, ACM, New York, NY, USA (2003)
8. Hill, S.: Game design theory: Multiplayer level design. http://www.alteredgamer.com/game-development/60255-game-design-theory-multiplayer-level-design/ (Jan retrieved 2014)
9. Holloway, J.: Deathmatch map design: The architecture of flow. http://www.gamasutra.com/view/feature/195069/deathmatch_map_design_the_.php (Jan retrieved 2014)
10. Hunicke, R., LeBlanc, M., Zubek, R.: MDA: A formal approach to game design and game research. In: Proceedings of the AAAI-04 Workshop on Challenges in Game AI. pp. 1–5 (Jul 2004)
11. Kerssemakers, M., Tuxen, J., Togelius, J., Yannakakis, G.: A procedural procedural level generator generator. In: Computational Intelligence and Games (CIG), 2012 IEEE Conference on. pp. 335–341 (2012)
12. Loguidice, B., Barton, M.: Vintage Games: An Insider Look at the History of Grand Theft Auto, Super Mario, and the Most Influential Games of All Time. Focal Press, 1 edn. (Feb 2009)
13. PolyphonyDigital: Gran turismo 5 course maker. http://www.gran-turismo.com/us/products/gt5/coursemaker/ (Jan retrieved 2014)
14. Scimeca, D.: How to build the best multiplayer fps maps. http://www.g4tv.com/thefeed/blog/post/719216/how-to-build-the-best-multiplayer-fps-maps-part-one (Jan retrieved 2014)
15. Togelius, J., De Nardi, R., Lucas, S.: Towards automatic personalised content creation for racing games. In: Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on. pp. 252–259 (2007)
16. Togelius, J., Preuss, M., Beume, N., Wessing, S., Hagelback, J., Yannakakis, G.: Multiobjective exploration of the starcraft map space. In: Computational Intelligence and Games (CIG), 2010 IEEE Symposium on. pp. 265–272 (2010)
17. Togelius, J., Yannakakis, G.N., Stanley, K.O., Browne, C.: Search-based procedural content generation. In: Proceedings of the 2010 International Conference on Applications of Evolutionary Computation - Volume Part I. pp. 141–150. EvoApplicatons'10, Springer-Verlag, Berlin, Heidelberg (2010)
18. VGChartz: Global yearly chart 2013. http://www.vgchartz.com/yearly/2013/Global/ (Jan retrieved 2014)
19. Watson, B., Muller, P., Wonka, P., Sexton, C., Veryovka, O., Fuller, A.: Procedural urban modeling in practice. Computer Graphics and Applications, IEEE 28(3), 18–26 (May 2008)