Computers & Graphics 44 (2014) 33–44

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Stateless generation of distributed virtual worlds

Jiri Danihelka*, Lukas Kencl , Jiri Zara

Czech Technical University in Prague, Faculty of Electrical Engineering, Czech Republic

ARTICLE INFO

Article history:
Received 11 February 2014
Received in revised form
11 July 2014
Accepted 11 July 2014
Available online 30 July 2014

Keywords:
Graph algorithms
Path problems
Distributed graphics
Computational geometry
Urban modeling

ABSTRACT

We present novel techniques for implementing possibly infinite on-demand generated 3D virtual worlds in distributed environments. Our approach can be useful in two scenarios: 1. A multiuser virtual world with mobile clients with sufficient CPU and GPU power but with limited network speed. This reflects current mobile phones, tablets and laptops in areas without a high-speed mobile connection or Wi-Fi connectivity. 2. Virtual world on-demand generation in a cloud environment that would be useful for scalable massive multiplayer games. If multiple independent generators create areas that are overlapping, our method ensures that the intersection of the areas will contain the same geometry for all of them. For this reason, we call our method *stateless generation*.

1. Introduction

Much attention is currently focused on multi-user virtual environments hosted in the cloud for both gaming and non-gaming purposes [1–3]. With the arrival of massive multiplayer online games, game creators have had to deal with limited server capacity in terms of world size or number of players [4], but virtual-world services must be scalable [5]. Current cloud computing technologies are able to provide additional resources on-demand, but virtual-world systems are rarely able to generate on-demand game content (to save memory for world parts not needed at the moment). Another problem is the limited network connectivity of mobile clients in cellular networks, which is often too slow for downloading the content generated on the server, thus having a highly negative impact on the emerging and ubiquitous mobile gaming.

Our method is innovative in that it eliminates the need to synchronize the static content that was procedurally generated on multiple devices. This will allow virtual-world servers to dedicate additional machines from the cloud environment to parallel content generation, or even to generate content on client devices. Because the content is generated on-demand, the virtual world can be considered theoretically infinite.

We refer to our method as *stateless generation* because it allows for locally generating only the content of the world that is relevant to the viewing frustum of the clients, and this content is generated independently, without knowledge of the states of the other generators.

As one of the most difficult virtual environments to generate, our efforts focus on generating an urban landscape (see Fig. 1). City environments are generally complex because they are structured and are both detailed and enormous. Procedural generation is a convenient tool for saving storage space and/or Internet bandwidth. When in the view frustum, buildings can be created on-demand from their lots (land parcels) using generating grammars.

Moreover, our method is general enough to be applicable to other types of structured landscapes (e.g., countryside, caves, labyrinths). Structured landscapes are generally more difficult to generate than unstructured landscapes (e.g., forests).

We provide a general purpose guideline for scalable algorithms generating virtual worlds that is applicable to most types of landscapes. We formalize the requirements and constraints such algorithms must fulfill. We then provide a novel algorithm for generating an infinite and scalable city-street layout, including a novel sub-algorithm for generating streets in a constrained environment, which could also be useful in traditional approaches to procedural city generation [6].

2. Related work

The most advanced approach for procedural building generation was published by Müller et al. [7] in 2006, improving upon the previous method by Wonka et al. [8] in 2003. The lot and street geometry can also be generated procedurally. The first such algorithm for finite cities was published by Parish and Müller [9].

In 2003, Geuter et al. [10,11] presented an algorithm for the on-demand generation of infinite cities in a regular rectangular grid.

* Corresponding author.
E-mail addresses: danihjir@fel.cvut.cz (J. Danihelka), kencl@fel.cvut.cz (L. Kencl), zara@fel.cvut.cz (J. Zara).

http://dx.doi.org/10.1016/j.cag.2014.07.002
0097-8493/

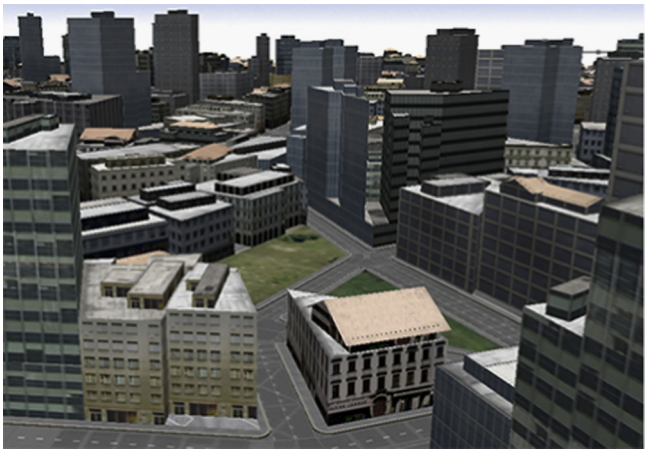

Fig. 1. A stateless infinite city generated by our method.

Fig. 2. Previous approach in infinite-city rendering published by Greuter et al. [10,11] showing street level view. Note the regular rectangular shape of the street network.

In their approach, the street network has to be aligned with the main axis, and all building lots must have the same square shape and size (see Fig. 2). The visible buildings are determined and procedurally generated according to the viewing frustum. Each building lot is assigned an integer number according to its coordinates using a hash function. This number is used as another seed for the pseudo-random building generation of that building lot. Some of these ideas are applied and extended in our approach.

A method for real-time generation of detailed procedural cities from GIS data was published by Cullen and O'Sullivan [12]. Their system uses a client–server approach, allowing multiple clients to generate any part of the city without requiring the full data-set. It creates the building geometry on-demand from the provided lot database and, in contrast to our work, does not address street and lot generation. Vanegas et al. [13] presented an interactive method for procedural generation of city parcels. They generate spatial configurations of parcels similar to real-world cities and support consistent lot locations relative to their containing blocks. Their approach generates parcels highly similar to those observed in real-word cities, but it mainly focuses on parcel layout and does not address all the phases of the city-generation process, unlike our method.

Aliaga et al. [14] presented a system for synthesizing urban landscapes by example. They proposed a random walk algorithm for obtaining parameters from existing cities that are later used in the generation process. Their system was somewhat capable of on-demand generation, but it was neither intended nor suitable for distributed environments because it required knowledge of all previously generated geometries for each future step.

On-demand world generation is highly related to texture synthesis algorithms. The main difference between these two approaches is the use of the generated results and whether the algorithm generates vector or raster output. Algorithms for texture synthesis usually use Voronoi diagrams [15] of randomly distributed points. One of the pioneering works in this area was published by Worley [16], who uses a function that complements Perlin fractal noise to produce textured surfaces resembling flagstone-like tiled areas, an organic crusty skin, crumpled paper, ice, rock, mountain ranges, and craters. Our algorithms are inspired by his function to determine the *n*th-closest points that affect the structure of the texture at the currently generated area. We aim for a similar goal, but we use Delaunay triangulation instead. We also use techniques based on Voronoi diagrams to divide areas that are affected by different geometrical elements.

Liang et al. [17] presented another algorithm for synthesizing textures from an input sample in real-time, but they use a significantly different approach than ours and their results are not isotropic, which is important for stateless generation. Texture synthesis can also be used to generate street patterns, [18,19] but these works use different approaches that are difficult to adjust for the purposes of infinite cities. Lefebvre and Hoppe [20] presented an algorithm for parallel on-demand texture synthesis based on a neighborhood matching approach. Their scheme defines an infinite, deterministic, aperiodic texture from which rectangular views can be computed in real-time on a GPU. Another advance in infinite texture generation was made by Cohen et al. [21]. They utilized a small set of Wang Tiles to tile a plane non-periodically. Using a proper tile set, the texture can be extended on-demand. In 2007, Merrell [22] presented an algorithm for generating 3D buildings and cities from a set of 3D tiles. Merrell's later work [23] was focused on continuous city model synthesis. These techniques are, however, limited to structures aligned with the main axes.

To generate a realistic world structure, we must first analyze examples to acquire characteristics that are later used in procedural modeling. Important progress in inverse procedural modeling was made by Stava et al. [24]. They create parametric context-free L-systems that represent an input 2D model. Their approach is based on vector shape recognition, clustering in the transformation spaces and detecting structures as L-system rules. Elements and structures can be edited by changing the L-system parameters.

Our approach follows up on the above works, combining their benefits and removing some of their limitations. Unlike [10,11,22,23], our building lots can have various sizes and shapes, streets can be arbitrarily oriented, and the street network is not periodic. Unlike [14], our approach adds capabilities for distributed environments as well as the ability to generate only content related to the view frustum of the client.

3. Stateless generation approach

We have laid down the following general requirements for our world generator:

1. The generated world is infinite and is not periodic.
2. Clients and/or servers are able to generate the static part of the world on-demand; they do not have to download it from a single (common) place. They should download only one random generator seed (or hash function) for the whole world.
3. Generation of the world can start from any point. The client will generate only those parts that are relevant (e.g., visible) to it.
4. The generation process is deterministic (usually achieved using pseudo-random generators). The results are always identical, regardless of the starting point or the area relevant to the client.

If a generator fulfills the above requirements, we call it a *stateless generator*. This is because its results do not depend on the results (or inner states) of other generators working in parallel