Mugla Journal of Science and Technology

A GRID-BASED MULTI-ZONE BURGESS APPROACH FOR FAST PROCEDURAL CITY GENERATION FROM SCRATCH

Buğra Yener ŞAHİNOĞLU*, Engineering Directorate, BITES Defence and Aerospace Tech. Inc., Ankara, bugra.sahinoglu@gmail.com
(https://orcid.org/0000-0001-9967-0781)
Ufuk ÇELİKCAN*, Department of Computer Engineering, Hacettepe University, Ankara, ufuk.celikcan@gmail.com
(https://orcid.org/0000-0001-6421-185X)

Received: 31.08.2021, Accepted: 20.04.2022 Research Article
*Corresponding author DOI: 10.22531/muglajsci.988965

Abstract

In this work, we present a novel methodology for procedural city generation from scratch, with the main goal of producing realistically structured cities with as little user input as possible. The methodology offers a thorough solution including procedural zone generation, procedural road network generation, procedural parcellation and procedural building generation. As we adopt a Burgess development model, the generated cities are complete with various zones of urban and suburban districts and public structures such as parks, schools, hospitals and police stations. We demonstrate the practicality of the proposed methodology via an application featured with a simple easy-to-use interface. The advantages of the proposed methodology, such as fast generation time and low resource requirements, are demonstrated in comparison to a similar commercial city generation engine.

Keywords: Procedural generation, city generation, road network generation, parceling, building generation, simulation

SIFIRDAN HIZLI PROSEDÜREL ŞEHİR ÜRETİMİ İÇİN IZGARA YAPILI ÇOK BÖLGELİ BURGESS YAKLAŞIMI

Özet

Bu çalışmada, mümkün olduğu kadar az kullanıcı girdisi ile gerçekçi biçimde yapılandırılmış şehirler üretme ana hedefi ile sıfırdan prosedürel şehir üretimi için yeni bir metodoloji sunuyoruz. Önerdiğimiz bu metodoloji, prosedürel bölge oluşturma, prosedürel yol ağı oluşturma, prosedürel parselasyon ve prosedürel bina oluşturma dahil olmak üzere kapsamlı bir şehir üretim çözümü sunmaktadır. Bu çözümde Burgess gelişim modelini benimserken, oluşturulan şehirler, çeşitli kentsel ve banliyö bölgeleri ve parklar, okullar, hastaneler ve polis karakolları gibi kamu yapıları ile tamamlanmaktadır. Metodolojinin uygulanabilirliğini, basit, kullanımı kolay bir arayüze sahip bir uygulama ile göstermekteyiz. Sonuçlar, benzer bir ticari şehir üretim motoruna kıyasla, metodolojinin hızlı üretim süresi ve düşük kaynak gereksinimleri gibi avantajlarını ortaya koymaktadır.

Anahtar Kelimeler: Prosedürel üretim, şehir üretimi, yol ağı üretimi, parselleme, bina üretimi, simülasyon

Cite

Sahinoglu, B.Y., Celikcan, U., (2022). "A Grid-Based Multi-Zone Burgess Approach for Fast Procedural City Generation from Scratch", Mugla Journal of Science and Technology, 8(1), 8-18

1. Introduction

The genesis of the term procedural generation in video games dates back to late 70's. In computer science, procedural generation is an approach for generating data with algorithms rather than manual creation. In computer graphics, it is also called random generation and is commonly used to create textures and 3D models. It becomes particularly beneficial in video games, where it can be used for automatically creating massive amounts of game contents. Advantages of procedural generation include smaller file sizes, larger amounts of content and randomness for less predictable gameplay.

There has been a growing interest in creating realistic and large synthetic cities on demand as they add realism and complexity to games and simulations, where cities can be used as open worlds to be explored and interacted with. Such cities even greatly influence the game flow as they give players a sense of freedom in terms of places to explore, while allowing the developers to create their games without losing control of the gameplay. In simulations, cities provide complex and realistic training environments. Using virtual cities for military or civil trainings such as bomb disposal, hostage rescue, intervention in internal conflicts, firefighting, flood victims rescue, earthquake protection, search and rescue missions etc. can be more effective than conducting these trainings on sparse terrains. Biljecki et al. [25] demonstrated 29 different use-cases of virtual cities in more than 100 different applications, illustrating the urgency of diverse 3D virtual cities.

In this vein, there have been numerous studies that aimed to generate synthetic replicas of real cities in whole or in part. Şenol and Kaya [22] generated a 3D model of Çiftlikköy Campus of Harran University in CityEngine using only data that is freely available on internet. Likewise, Şenol et al. [23] used a geodesign method to plan urban transformation areas and generated the re-designed Eyyubiye district with CityEngine. Ernst et al. [24] also used a geodesign method to create a new master plan for Harran University campus. Büyüksalih et al. [26] generated a 3D replica of Istanbul with Unity based on CityGML schema version 2.0.

Building cities from scratch manually is a very costly and time-consuming task that requires lengthy efforts by large teams. With procedural city generation, a different user experience can be achieved each time by creating different cities for large-scale simulations and games. Furthermore, it can be used for producing large and varied datasets to be used in scientific research. So much so that the biggest impact of procedural city generation is likely to be on social simulations and urban testbeds [1]. The cities produced by procedural methods will contribute to the work in these fields by increasing the number and diversity of the complex environments that can be used in synthetic data requirements.

In the previous studies on procedural city generation, the cities are not produced in a layered structure, which is observed in many large city layouts worldwide. This type of concentric development in layers called zones through the years is known as the Burgess model. The lack of zones gives the cities created with the previous work a rather artificial look.

With an aim to offer an alternative that is suitable for the needs of the state-of-the-art games and simulations, we introduce a new procedural city generation framework consisting of several algorithms that we developed for creating large-scale virtual cities. In a grid-based multi-zone Burgess approach, the proposed framework can create cities that extends from a city center convoluted with high-risers to suburbs populated with family homes, all connected with a dense road network, within seconds. Another significant contribution of our methodology is that it features a holistic approach to procedural city generation as it presents a complete solution encompassing generation of the whole city layout including zoning, blocking and parcellation; generation of primary and secondary road networks; generation and placement of public structures such as parks, schools, police stations and sports fields; and finally, generation and placement of all other buildings populating the city. Furthermore, we propose novel procedural generation algorithms virtually in all of these stages in order to provide a diversified result that is unique at every run.

The rest of this paper is organized as follows. First, we review the literature on procedural city generation in Section 2. Then, we present our methodology in Section 3. Section 4 gives the details of our user application and demonstrates the practicality of our methodology by presenting the performance results in comparison to a similar commercial city generation engine. Finally, Section 5 concludes the paper.

2. Related Work

Synthetic city generation is a multifaceted issue composed of multiple subproblems such as road network generation, layouting, parceling and building generation. Hence, it requires a different solution for each of these subproblems. A review of the previous work shows that many of the prior studies handled only a subset of these problems.

The previous studies focusing on road networks have used various methodologies to procedurally generate road networks by different means such as L-Systems, templates, graphs, and even analyzing real cities. Zee et al. [2] used L-system as a mathematical formal grammar. L-system was first devised by the biologist Aristid Lindenmayer to model the growth of plants. The system works in a rewriting process, which starts with an axiom or an initial state that is rewritten using a set of rules, i.e., a grammar. The rewriting process repeats recursively until the iterations are completed by creating a string that defines a complex object [3]. Kelly and McCabe [4] have examined city generation in three stages and the first two were about road network generation. They initially generated a primary road network using undirected planar graphs that are implemented as adjacency lists, then they generated the secondary road network with the help of the L-System algorithm. Whelan et al. [5] suggested the formation of road networks in two stages, as well. In the first stage, the user selects points on a 3D terrain and their algorithm generates a road network based on these selections. In the second stage where they use the L-System algorithm, they begin at the borders of the city cells, which are sections of a terrain enclosed by primary roads, form neighborhoods and proceed inwards in a parallel fashion. In Sun et al.'s work [6], road networks are generated based on templates. Voronoi diagrams are used for population-based templates and L-System was used for their raster and radial templates. In Lechner et al.'s work [7], tertiary road networks are generated using an agent-based method. Two agents are created for this task. An extender agent roams the terrain until it finds a space lacking connection to the road network and then it builds a road from that point to the network. The other one, a connector agent, wanders over the existing road network and builds road segments between unconnected patches of the road. Hartman et al. [8] proposed to synthesize road networks by making use of generative adversarial networks (GANs). Their system is made up of two main steps. In the first step, a raster image is created from a road network patch from real world and in the second step the trained GAN model is used to synthesize road network variations from images containing uniformly sampled noise.

Parceling, on the other hand, is mostly done using subdivision algorithms in the examined work. Yang et al. [9] generates parcels by using two splitting algorithms. These are called template-based splitting and streamline-based splitting. The user divides the region, where the city is to be located, with one or more streamlines from end to end. After this, the region is subdivided with lines that are parallel to these streamlines. The areas between streamlines are further subdivided with templates that they provide. Since their templates have similarly sized parcels and few in number, generated city parcels are mostly isometric and this is particularly noticeable in neighboring parcels. Vanegas et al. [10] proposed to separate city blocks into parcels using oriented bounding box (OBB) -based subdivision, which is a binary space partitioning method. Kelly et al. [4] used the lot subdivision method, which recursively subdivides each region until a target lot size is reached. Each division is realized with a line perpendicular to the longest side. Even though it is suitable where the input data is generally regular and block shaped, the lot subdivision algorithm becomes erratic when lots are angular and irregular as in a typical suburban road network. As a remedy for this case, the division is first prioritized along sides with road access. Lots without road access are not considered suitable for building development and discarded or labeled as green spaces.

Building generation has been handled using different approaches in previous studies. Some of them were interested in generating the shape of the building procedurally, while others generated just the facade of the buildings. In Seifert et al.'s work [11], all buildings in the same block are kept alike by using similar parameters. Their buildings are generated from a baseline, taken as parallel to the road level. The baseline is extruded to create a polygon; the polygon is extruded to create a shape for a new building; and finally building generation ends with roof generation. The main focus of Greuter et al.'s study [12] was on generating office buildings using a method that merges various primitive shapes into a floor plan and extrudes these plans to random heights. Wonka et al. [13] used a split grammar to procedurally generate facades of city buildings. A building facade is represented by this grammar as a non-terminal shape and subdivided until reaching grammatically terminal shapes. When the split grammar algorithm completes the subdivision of the facade, terminal shapes are replaced with windows, doors, walls etc. In Seifert et al.'s work [11], generated buildings in the same area are of similar sizes. This creates a look with a coherent silhouette for the city. They further stated that their method can be used in urban planning in existing cities. Müller et al. [14] introduced CGA, a shape grammar for procedural architecture modeling on a large scale by refinement of shapes via expanding a basic vocabulary iteratively. Schwarz and Müller [15] extended this as a grammar language called CGA++ with two main features. The first one grants shapes first-class citizenship that helps individual shapes to be uniquely identified. With this feature, operations were able to take shapes as arguments, enabling Boolean operations. The second feature provides a dynamic grouping mechanism and synchronization facility by a linguistic device of events to enable coordination across a group of shapes.

In general, studies have handled city layouting task by using road networks, since most of them generate cities modeled by rectangular-grids. Only a few have used other specific methods to generate different types of city layouts. Zee et al. [2] generated zones with manually analyzing the map of the original city. In Yang et al.'s method [9], city layout is generated by producing parcels with splitting algorithms. In Groenewegen et al.'s work [16], city limits are generated based on the real cities in Western Europe. They generate candidate locations for districts based on a random distribution and the best locations are chosen for placement of the districts with respect to the positions of the previously placed districts, terrain type, area within the city, distance from rivers and distance from highways. For example, the industry-heavy districts have a high attraction towards water-adjacent locations. Finally, they divided those districts by generating a Voronoi diagram. Bustard et al. introduced the PatchCity method [17], where a texture synthesis approach is used for generating city layouts. They use one or more vector street maps as inputs for this image-based texture synthesis.

There have also been studies that handled multiple aspects of city generation together. Kim et al. [18] created such a system in which they first parse the input query image to extract a city component vector which pass through the trained GAN model to obtain terrain and height maps. They also construct a convolutional neural network (CNN) model that takes the same image to create a city property vector. After that, they synthesize a city model based on the created terrain and height maps by applying parameters collected from the city property vector. Steinberger et al. [19] introduced a method that provides city layout and building generation conjointly by extending parallel generation of architecture. The method generates city layouts using visibility pruning through multiple phases. In the first phase, building hulls that are assumed to be conservative bounding volumes are generated for view frustum pruning. If all vertices of the generated hull are outside of the same side of the view frustum, the building can be ignored, i.e., will not be put through further processing. In the second phase called building specification, a geometric description of the building is constructed for occlusion pruning. Arbitrary compositions of shapes are determined for this phase by associating each one with a corresponding building and designating an occluder type, which can be opaque, enclosing, or hidden. If an opaque or enclosing shape is determined to be visible, the building is constructed. On the other hand, the entire building is skipped when no part of the building specification is visible. Finally, geometry is generated for all buildings that pass visibility pruning. In this phase,

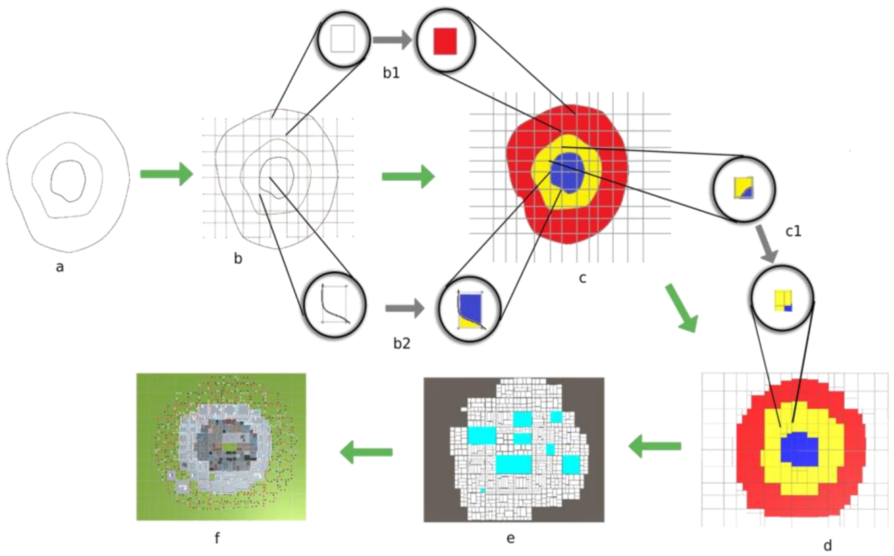


Figure 1. Method overview: (a) Defining tentative zone borders. (b) Forming clusters. (b1, b2) Interim zone assignment: (b1) If a cluster belongs to a single zone, that zone value is assigned directly; (b2) if a cluster lies over multiple zones, zone values are assigned using Algorithm 1. (c) Result of interim zone assignment. Generating city blocks: (c1) handling clusters with multiple zones. (d) Resulting city blocks with finalized zone assignments. (e) View of the parcels formed after subdividing city blocks. Blue areas are reserved for public structures, and therefore are not subdivided. (f) Final result with completed road network, buildings and public structures.

they propose to use automatically generated surrogate terminals for procedural level of detail. Parish and Müller [20] also explored city generation in an expanded approach with sub-systems for road generation, lots division and building generation. According to their research, those were found to be the slowest changing elements of the cities. They proposed a method called Extended L-Systems to generate the roads. They port the setting and the modification of parameters in the L-system modules to external functions. To generate parcels, they subdivide the blocks that are formed during road generation into smaller pieces by using a recursive algorithm which divides longest edges until the parcels meet or go under the thresholds specified by the user. They use L-Systems also to generate buildings. Furthermore, they designed a tool to create facade textures for buildings semi-automatically using a simple functional composition of layers.

3. Method

The survey of the prior work on the matter shows that in previous studies cities are not produced in layers, or zones, which are observed in layouts of many real-world cities as a result of historical development either naturally or by design. The lack of zones in the generated cities gives them an uncanny appearance. To bridge this gap and offer an alternative that is suitable for the needs of the state-of-the-art games and simulations, we propose a grid-based multi-zone Burgess approach to facilitate fast procedural generation of synthetic 3D cities from scratch. To this end, we developed a framework consisting of an array of algorithms utilized in tandem. For some of these algorithms, we improve upon the previously established techniques such as Catmull-Rom splines, OBB-based subdivision, zone-based subdivision, random walk algorithm, and Perlin-noise algorithm. An overview of our methodology is given in Figure 1 and its stages are detailed in the following.

3.1. Generating Smooth and Non-Deterministic Borders for City Zones

In our approach, cities are generated with distinct zones. The name “zone” in this context denominates a region where similar architectural structures exist. Naturally, a city grows outwards from an established city center, forming various zones. This type of development is called concentric zone model, also known as the Burgess model [21]. In time, the region around the old city center becomes a heavily commercial zone (the downtown) and the outskirts of the developed city becomes the affluent residential suburban zone (the uptown). The ring in between these two is called the midtown, home to the city’s middle-income population.

To generate the zone borders, a closed form Catmull-Rom spline consisting of Hermite curves is created to form the outer boundary of each zone. The number of curves n can be set arbitrarily as a parameter of the algorithm. For n curves, n control points are needed. These control points are created on directed line segments emanating from the designated center point. The successive line segments have equal angular separation from each other by $360/n$ degrees. For the sake of a structure that facilitates faster computation, the center is simply chosen as the origin.

In our methodology, the only set of inputs requested from the user is the ranges of radial distances of outer zone boundaries from the city center. That makes a total of 6 integer values as input, since our approach generates

cities with 3 zones. For the zone i, where $i \in \{1, 2, 3\}$, the inputs $z_{i_{min}}$ and $z_{i_{max}}$ define the minimum and maximum radial distances of that zone, respectively.
To use as the control points c_j of a spline curve, where $j \in \{1, 2, ..., n\}$, the algorithm generates a random number between $z_{i_{min}}$ and $z_{i_{max}}$ for each of the n directed line segments. Then, any two consecutive control points c_j and c_{j+1} become the start and end points of the curve j between them and any point on this curve can be calculated parametrically by varying the designated parameter in the unit interval [0, 1], where 0 and 1 correspond to c_j and c_{j+1}, respectively. Thus, the algorithm stores the outer boundary of each zone in a separate array for each one of the n curves. No computation is required for the inner boundaries, as the outer boundary of one zone is the inner boundary of the next zone.

3.2. Forming Clusters

Next, the whole city grid is divided into clusters by random segmentation of horizontal and vertical lines, which serve as the preliminary road network to become the primary road network later (see Section 3.4 below). For this, the vertical and horizontal lines are selected randomly in column and row indices of the grid matrix. Consequently, neighboring vertical and horizontal line pairs define a cluster that is the axis aligned rectangular region bounded by these lines on each side.

3.3. Interim Zone Assignment to Grid Cells

In our approach, we use a grid-based structure to generate the interim zone structure. In this way, a grid cell is the most elementary structural unit in our methodology. Consequently, sides of a grid cell over the virtual terrain are taken as a unit length, which corresponds to a physical length of 4 meters in real-world.
It is necessary to identify the grid cells that lie on the three zone boundaries beforehand. For this, elements of the stored spline arrays are converted to two-dimensional integer coordinates. Then, to find out which cells belong to which zones, we carry out an algorithm that first checks the edges of the processed cluster for zone boundaries. If none of the cells on the edges of the cluster is a zone boundary, this shows that the cluster sits entirely in a single type of zone. Then, the algorithm detects which zone that is and assigns all cells within the cluster to the detected zone. On the other hand, in case the algorithm happens upon a zone boundary on the edges, then it processes the cells within for further inspection again starting from the edges. Further details of the procedure are given in Algorithm1.

3.4. Generating the Primary Road Network

In this step, our algorithm expands the preliminary network generated during the formation of clusters into the primary road network. The amount of expansion per road is carried out in a randomized way to create primary roads in varying number of lanes between two and six. In order to discard the excessive polygons where two roads meet, our method generates crossroads by

Algorithm 1: INTERIM ZONE ASSIGNMENT

```
ProcessClusterCellsForZoneAssignment(cluster):
    Check cluster edges for zone boundaries;
    if zone boundary is not detected then    /* indicates that the processed cluster lies within a single zone */
        zoneValue ← BoundaryControl(the first cell in the cluster closest to the center);
        /* zoneValue: 1 for the downtown (the central zone), 2 for the midtown (the middle zone), 3 for the uptown/suburbs (the outer
        zone), 4 for the remaining non-zoned areas */
        assign zoneValue to all cells in the cluster;
    else   /* zone boundary is detected >> the processed cluster covers portions from multiple zones */
        foreach cell sitting on the cluster edges do
            if processed cell has not been marked as checked then
                zoneValue ← BoundaryControl(processed cell);
                assign zoneValue to the processed cell;
        foreach cell in the remainder of the cluster do
            if processed cell has not been marked as checked then
                zoneValue ← BoundaryControl(processed cell);
                for i = 1 to 4 do
                    assign zoneValue to all cells moving from the processed cell until reachedBounds[i] in the
                    designated axis-aligned direction and mark visited cells along the path as checked;

BoundaryControl(cell):
    for i = 1 to 4 do  /* starting from the input cell, go through the unchecked cells in each of the four orthogonal directions along
    the horizontal and vertical axes */
        move until either a zone boundary, grid boundary, the processed cluster's boundary or a checked cell is reached
        and mark visited cells along the path as checked;
        get zoneValue of the last processed cell when the move stops and store it in reachedBounds[i];
    return the largest zoneValue observed;
```

splitting roads from the overlapping part and deleting the extra layer of overlap.
The basic building block of our primary road mesh consists of six triangles. The first and the last two of these make up the rectangles which are textured with zebra crossings (the leftmost and the rightmost portions in Figure 2a). The remaining two in the middle form the main rectangular part of the road block with the lane texture. Width of a lane is set to a half unit (i.e., 2 meters in real-world). Texturing a road of a given width is done by repeating the road texture for every half unit across that road. When the road mesh is extended along a line to create a primary road, the road texture on it expands between two crossroads such that only the main part with the lane is repeated along the line, while the parts with zebra crossings are kept as is. Then, the mesh with the extended texture is repeated side-by-side as many times as the number of lanes assigned for that road necessitates (Fig. 2a). Finally, the method places traffic lights on all four corners of each crossroad (Fig. 2b), excluding the ones with roads emerging on only three directions.

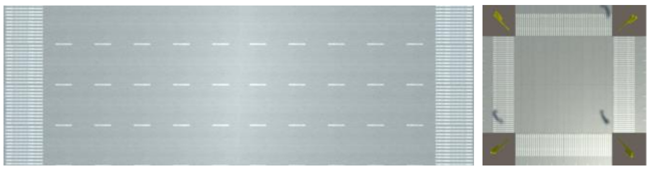


Figure 2. (a) An example showing the basic road mesh expanded into a primary road. (b) A crossroad.

3.5. Generating City Blocks

Since different zones allow for different sets of buildings within, the algorithm computes the parcels on which the buildings are placed on a given block according to the zone it is assigned to. However, as some clusters may contain grid cells of more than a single zone, a procedure is devised to divide a cluster that lies over multiple zones into blocks where each block belongs to a single zone. That is, for such a multi-zone cluster, the algorithm first detects the innermost zone within it. Then, it also determines the vertical and horizontal limits of the innermost zone within the cluster. Due to the nature of our zone border generation algorithm, the innermost zone in this cluster is connected to at least one edge. Leveraging this aspect and using the gathered information about the innermost zone, the cluster is broken into rectangular pieces by vertical and horizontal dividers with respect to the limits of the innermost zone (Fig. 3). This process results in a minimum of 2 and a maximum of 6 pieces. After the division, the algorithm goes through the side lengths of the resulting pieces. If either side of a piece is smaller than the threshold set for the corresponding axis, that piece is marked as irregular. Thresholds are dictated by the minimum values set for the buildings to be placed in that zone. Each irregular piece is merged with a neighboring regular piece along

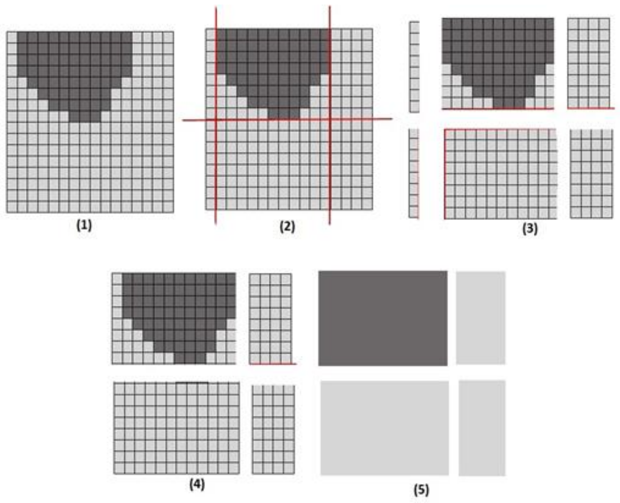


Figure 3. An example showing the steps of converting a cluster encompassing cells of two city zones (1) to a city block that is assigned to a single zone (5). This is an example of the extreme case where the block is divided into 6 pieces (3).

the axis of irregularity. This partitioning process is carried out recursively for each multi-zone cluster.
Roads with four lanes are added along the division lines between the resulting blocks. These newly added roads serve as an extension of the primary road network. This last step erases the cells now covered by a lane from both blocks on either side of a division line.

3.6. Placing Public Structures

Public structures such as parks, schools, hospitals, fire stations, sports facilities and police stations are universally essential in all large-city layouts. Since the areas for these structures are to cover multiple parcels, our method will reserve the blocks for these areas before the division of blocks into parcels.
The details of public structure placement procedure are given in Algorithm 2. The number of each type of public structure and the zones to place them are predefined and can be changed as desired. The algorithm selects a random block from each zone to place these structures. Chosen blocks can be used with multiple purposes. Such that, if the area of the randomly chosen block is larger than the base area of the assigned public structure, then the public structured is placed by making its entrance roadside and as many sports fields as possible are placed in the remaining part of the block.
The algorithm also procedurally generates a large municipal park in the central zone. This large park may or may not include a monument depending on its size. Either way, wooded areas covering the lion share of the four corners of the park, walking areas and benches alongside them are generated and placed procedurally.

3.7. Generating Parcels

For dividing blocks into parcels, we adopt a novel approach that builds upon the OBB-based subdivision method [10]. In OBB-based subdivision, an area, which will be recursively divided into smaller parts, is surrounded with an oriented bounding box. For

Algorithm 2: PLACING PUBLIC STRUCTURES

```
PlacePublicStructures():
    GenerateLargeMunicipalPark();
    foreach structure in publicStructureList do
        foreach zone do
            while do
                get random block within the zone;
                if the base area of the public structure fits the block then
                    block.type = publicStructure.type;  /* "police station", "hospital", "school", "small park", or "large municipal park" */
                    add block to publicBlockList;
                    break loop;
    foreach block in publicBlockList do
        generate a 5-sided park surface mesh using the park surface materials; /* do not need the bottom face which is not visible */
        create pavements surrounding the edges of the the block with predefined width and materials;
        add the generated park surface mesh at the vacancy within the pavements;
        if block.type is "small park" then
            add the pre-made park in the middle of the park block by replacing the surface mesh as needed;
        else
            place public structure within the block randomly by replacing the surface mesh as needed and also by making its entrance roadside;
            if (block area - base area of the public structure) > sportsfield base area then
                place randomly chosen sportsfield areas within the vacancies; /* such as football field, basketball field, tennis court, or running track */

GenerateLargeMunicipalPark():
    get the block closest to the city center;
    if block.size > monument.size*5 then ;   /* if the block is large enough for both monument and park */
        place monument in the middle of the block;
        /* calculating the dimensions of the wooded areas that occupy the four corners of the park */
    if monument exists then
        lengths of the wooded areas = (block.length – monument.length)/2-2;
        widths of the wooded areas = (block.width – monument.width)/2-2;
    else
        lengths of the wooded areas = (block.length – 6)/2;
        widths of the wooded areas = (block.width – 6)/2;
    place walking areas and trees accordingly;
    place benches on both sides of the walking areas;
```

parceling, the area is divided into two rectangular pieces by a line parallel to the shorter edge of the bounding box. This creates deterministic results. Thereby, for a given area, OBB-based subdivision method always generates the same end results.

In our method, the first major change is that the dividing line will not always be parallel to the shorter edge of the bounding box. Instead, it is carried out in a loop, cycling between vertical and horizontal divisions. And secondly, the lengths of the divided edge pieces are determined randomly, provided that they fall within the predetermined minimum and maximum limits. The whole set of rules for our subdivision approach is as follows.

- If the length of the edge to be divided is larger than twice the minimum limit and also larger than the maximum limit, division is carried out.
- If the length of the edge to be divided is larger than twice the minimum limit, but not larger

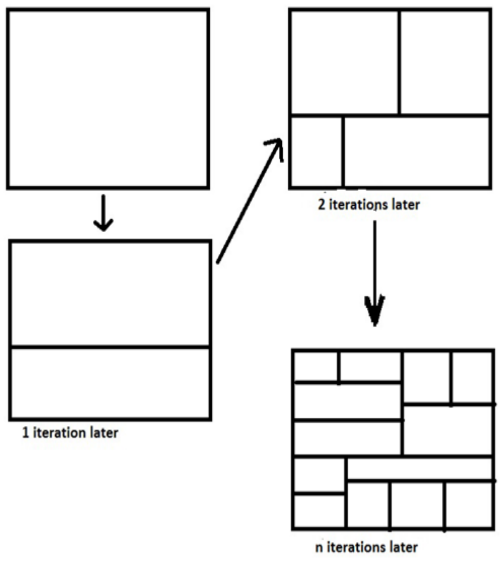


Figure 4. A sample parcellation process of a city block.

than the maximum limit, division takes place with a 0.5 probability.

- If the length of the edge to be divided is smaller than or equal to twice the minimum limit, division is not carried out.
- If at least one of the edges to be divided has no parallel connection to road, division is not carried out.
- The length of the divided parts cannot be smaller than the minimum limit.

With these rules, our method produces non-deterministic parcellation. A sample run of the algorithm is illustrated in Figure 4.

3.8. Generating the Secondary Road Network

To generate the secondary roads of the city's road network, we propose a graph-based random walk algorithm. Secondary roads with a width of two lanes are generated between the nodes of the graph that an agent travels randomly using the algorithm. The graph is built using parcels in a block as follows.

- Corners of parcels in a given block constitute the nodes of that graph.
- Two or more overlapping corners are registered as the same node.
- If two nodes lie on the same edge without another node in between, those two become neighboring nodes and are connected in the graph.
- However, if the connection between two neighboring nodes overlaps with an edge of the block, the connection between the two nodes is discarded, i.e., their neighborhood is broken.

An agent initiates a random walk on the graph starting on a node that is on an edge of the processed block. Such nodes are significant for our approach as they are connected to the primary road network. The agent continues the walk using the node neighborhood connections, never using the same connection more than once. Every node it visits is stored in a list and eventually, when the run inside the block is complete, this list gives the secondary road network inside that block. If a node visited by the agent is not on an edge, the node is pushed on a stack. When the agent happens on another node that is on an edge (i.e., indicating the agent reached another edge of the block) before the walk is complete, the agent jumps back to the last node added on the stack and continues its random walk. In case that all neighbors of this node have already been exhausted (i.e., visited), this node is removed from the stack. The random walk continues until the stack is emptied or the number of the generated secondary road pieces exceeds the predefined limit. After the secondary roads are generated, the remaining areas within parcels are the available spaces where (non-public) buildings are to be placed to populate the city zones.

3.9. Generating and Placing Buildings

To populate the city with buildings, there are two alternatives in our methodology. For the vacant parcels within the central zone, buildings are generated procedurally. For the other zones, 3D models from the library of available buildings are picked and placed randomly.

The blocks in the central zone (the so-called downtown of the city) need to be more tightly populated with buildings in comparison to outer zones as it is crucial to use the land much more efficiently in the city center. Hence, the algorithm generates buildings for the central zone to fit the exact sizes of the parcels they occupy. Height of each building is computed using a two-dimensional normalized Perlin-noise algorithm. A random value generated by the Perlin's algorithm is multiplied by a and then the result is added to b to give the height. While a and b are set to 70 and 30, respectively, by default, either can be changed as desired. Shapes of the buildings are randomly varied based on their height. Visual appearances of the buildings are also varied by applying textures randomly.

For the remaining two zones, buildings for a given block are chosen randomly with respect to which zone the block belongs to. Each of these two zones is given a separate building library. That is, while midtown-style buildings are placed in the second zone, single-family homes with spacious gardens are used in the third zone, which has the suburban districts of the city. The buildings are chosen to match the size of the parcels. It is also ensured that the front facade of each building is placed so that the parcel faces the roadside.

4. User Application, Results and Comparison

We developed a procedural city generator application in order to demonstrate the practicality of our methodology. The application is built using Unity graphics engine. A fully-featured WebGL version of the application is made available online[1].

The application has a rather easy-to-use and minimal interface. In the main menu (Fig. 5a), the user is only asked to enter the desired radius ranges for the three zones as input. The total number of buildings in the city is automatically determined by the zone sizes that result according to the user input. Then, the user can generate the city right away using a single button. After the generation completes in a few seconds, the user may either view the generated city from a top-down view or roam around in it using the free-fly camera mode (Fig. 5b). They may also observe the city in different layers (zone boundaries, blocks, parcels, as well as the 3D city) or in a combination of multiple layers.

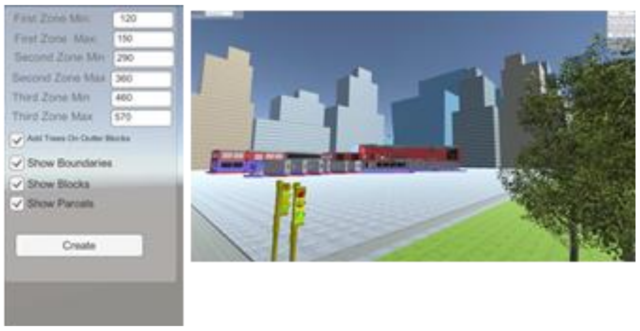
We tested our application in comparison to CScape, a commercial procedural city generation engine. CScape is a publicly available city generator that is also based on

[1] https://bugrayenersahinoglu.github.io/ProceduralCityGenerator/

Table 1. Comparison of our application with CScape.

Tool	Building Count	Time (second)	Reserved Memory (GB)	Used Memory (GB)
CScape	840	61.05	1.76	1.43
City Generator	841	16.8	1.25	0.98

the Unity graphics engine. Thus, we were able to make a fair benchmark test with it. CScape also uses a grid-structure for parcellation but only generates Manhattan-like cities which correspond to the central zone produced with our approach. A visual comparison of the cities generated by both engines is available in Figure 6. While the generated buildings by CScape look more photorealistic, CScape only generates business district type city centers and the layouts of the generated cities are always similar to each other. Our application, on the other hand, is able to generate cities as a whole, with more diverse compositions, including public structures.
The performance results of the comparison are given in Table I. It is seen that our application is able to generate a city that is similarly complex in considerably less time using significantly less memory.
Both city generators were tested in Unity's editor mode, since CScape runs only in this mode. A notebook computer with 8GB RAM, 2.70GHz Intel i7-5700HQ processor, and a 2GB NVidia 960m graphics card was used for the tests. It was seen that a city of 841 buildings was generated by our application in 16.8 seconds using 0.98GB memory, while CScape generated a city of 840 buildings in 61.05 seconds using 1.43GB memory.
The executable version of our application is even faster, generating a city of 836 buildings in 5.81 seconds. At runtime, frame refresh rate of our application varies between 5 and 20 FPS with the executable, and between 2 and 17 FPS in the Unity editor mode.

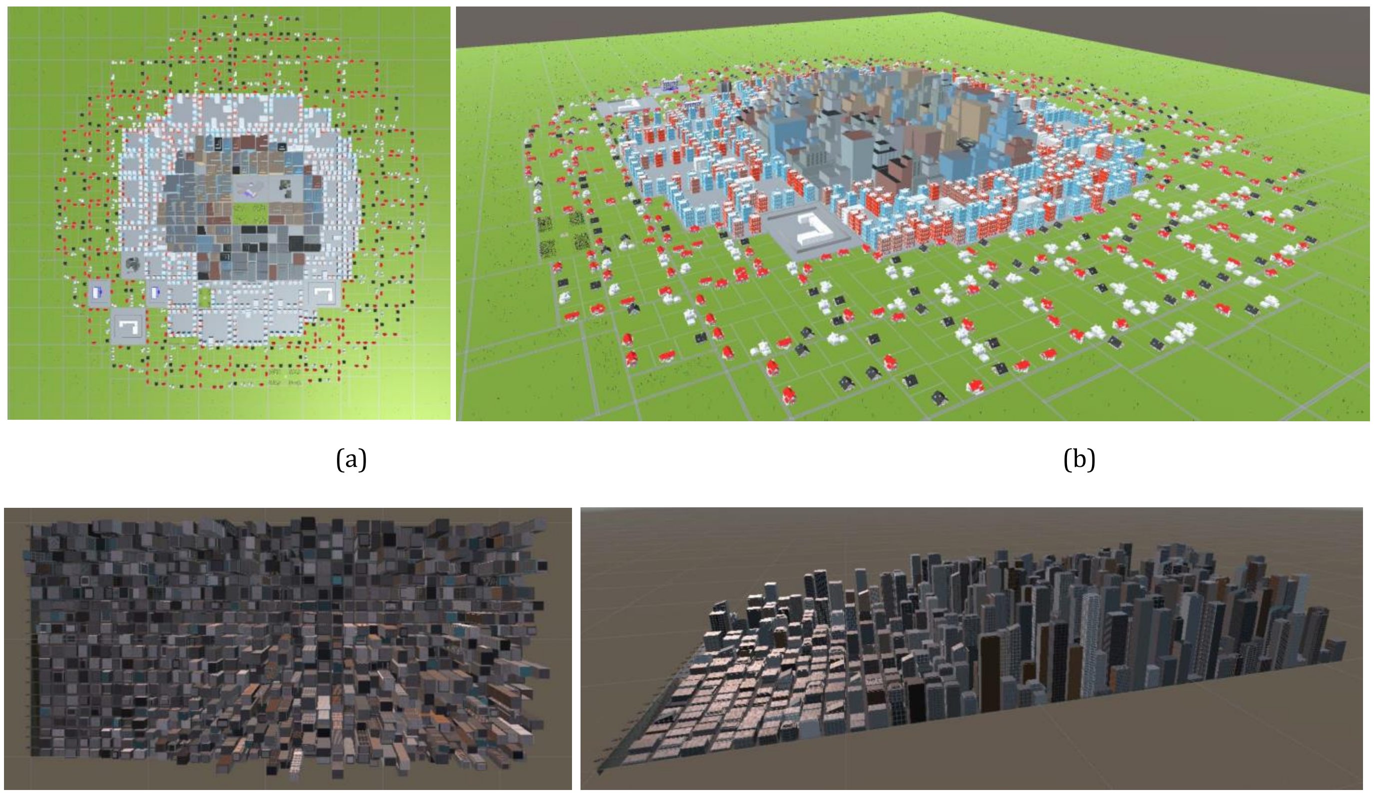

First Zone Min 120
First Zone Max 150
Second Zone Min 290
Second Zone Max 360
Third Zone Min 460
Third Zone Max 570
✓ Add Trees On Outer Blocks
✓ Show Boundaries
✓ Show Blocks
✓ Show Parcels

(a) (b)

Figure 5. (a) The main user interface of our application. (b) A sample screenshot of a city procedurally generated by our application with a view of the city's urban center

5. Conclusion and Future Work

In this work, we presented a novel set of methods for procedural large city generation from scratch. The proposed methodology offers a holistic grid-based approach complete with solutions for the whole city layout and road networks, public structures as well as private buildings. To address the problem of uniformly created cities in the previous work, our approach adopts a Burgess model which is manifested in a concentric development layered in zones and therefore grants a more natural overall look.



(a) (b)

(c) (d)

Figure 6. (a) Top-down view and (b) perspective view of the city generated by our application. (c) Top-down view and (d) perspective view of the city generated by CScape.

We demonstrated the practicality of the offered methodology in a framework which is made publicly available as a web application. We tested the presented framework with respect to a commercial alternative and showed its diversity as well as its rapidity. The results exhibit the applicability of our methodology towards the needs of the-state-of-the-art games and simulations which call for fast generation and varied outcomes that look fairly realistic. Despite its contributions and improvements to the existing body of work, the presented methodology has several limitations. First of all, our method, like many other procedural city generation methods, assumes a completely flat terrain across the land to be build the city on. However, that is rarely the case in reality. Therefore, changes to the proposed method are necessary to make it generalized enough for applicability to terrains that are more rugged, coastal terrains and terrains with waterways such as rivers or canals. Another shortcoming is that our method builds a city on an exact grid structure while very few real cities adhere to this. Finally, further arrangements can be easily added to the proposed methodology in order to generate cities consisting of zone structures with more than one center, as observed in many metropolitan cities.

6. References

[1] Kim, J. S., Kavak, H. and Crooks, A., "Procedural city generation beyond game development", *SIGSPATIAL Special*, *10*(2), pp. 34-41, 2018.

[2] van der Zee, A. and de Vries, B., "Modeling of RL-Cities", *30th International Conference on Education and Research in Computer Aided Architectural Design in Europe (eCAADe 2012),* eCAADe and CVUT, Faculty of Architecture, pp. 375-380, 2012.

[3] Prusinkiewicz, P. and Lindenmayer, A., *The algorithmic beauty of plants*. Springer Science & Business Media, 2012.

[4] Kelly, G. and McCabe, H., "Citygen: An interactive system for procedural city generation", *Fifth International Conference on Game Design and Technology*, pp. 8-16, 2007.

[5] Whelan, G., Kelly, G. and McCabe, H., "Roll your own city", Proceedings of the 3rd international conference on Digital Interactive Media in Entertainment and Arts, pp. 534-535, 2008.

[6] Sun, J., Yu, X., Baciu, G. and Green, M., "Template-based generation of road networks for virtual city modeling", *Proceedings of the ACM symposium on Virtual reality software and technology*, pp. 33-40, 2002.

[7] Lechner, T., Watson, B. and Wilensky, U., "Procedural city modeling", *1st Midwestern Graphics Conference,* 2003.

[8] Hartmann, S., Weinmann, M., Wessel, R. and Klein, R., "Streetgan: Towards road network synthesis with generative adversarial networks", *25th International Conference on Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, pp. 133-142, 2017.

[9] Yang, Y. L., Wang, J., Vouga, E. and Wonka, P., "Urban pattern: Layout design by hierarchical domain splitting", *ACM Transactions on Graphics (TOG)*, *32*(6), pp. 1-12, 2013.

[10] Vanegas, C. A., Kelly, T., Weber, B., Halatsch, J., Aliaga, D. G. and Müller, P., "Procedural generation of parcels in urban modeling", *Computer graphics forum*, Oxford, UK: Blackwell Publishing Ltd, Vol. 31, No. 2pt3, pp. 681-690, 2012.

[11] Seifert, N., Mühlhaus, M. and Petzold, F., "A parametric 3d city model: basis for decision support in inner-city development", *Proceedings of the 16th International Conference on Computing in Civil and Building Engineering*, pp. 1285-1292, 2016.

[12] Greuter, S., Parker, J., Stewart, N. and Leach, G., "Real-time procedural generation of pseudo infinite cities", Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia, pp. 87-ff, 2003.

[13] Wonka, P., Wimmer, M., Sillion, F. and Ribarsky, W., "Instant architecture", *ACM Transactions on Graphics (TOG)*, *22*(3), pp. 669-677, 2003.

[14] Müller, P., Wonka, P., Haegler, S., Ulmer, A. and Van Gool, L., "Procedural modeling of buildings", *ACM SIGGRAPH 2006 Papers*, pp. 614-623, 2006.

[15] Schwarz, M. and Müller, P., "Advanced procedural modeling of architecture", *ACM Transactions on Graphics (TOG)*, *34*(4), pp. 1-12, 2015.

[16] Groenewegen, S. A., Smelik, R. M., de Kraker, K. J. and Bidarra, R., "Procedural city layout generation based on urban land use models", *Short Paper Proceedings of Eurographics 2009*, 2009.

[17] Bustard, J. D. and de Valmency, L. P., "PatchCity: Procedural City Generation using Texture Synthesis", *IRISH MACHINE VISION & IMAGE PROCESSING Conference proceedings 2015*, 2015.

[18] Kim, S., Kim, D. and Choi, S., "CityCraft: 3D virtual city creation from a single image", *The Visual Computer*, *36*(5), pp. 911-924, 2020.

[19] Steinberger, M., Kenzel, M., Kainz, B., Wonka, P. and Schmalstieg, D., "On-the-fly generation and rendering of infinite cities on the GPU", *Computer graphics forum,* Vol. 33, No. 2, pp. 105-114, 2014.

[20] Parish, Y. I. and Müller, P., "Procedural modeling of cities", Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 301-308, 2001.

[21] Kearsley, G. W., "Teaching urban geography: The Burgess model", *New Zealand Journal of Geography*, 75(1), pp.10-13, 1983.

[22] Şenol, H. İ. and Kaya, Y., "İnternet Tabanlı Veri Kullanımıyla Yerleşim Alanlarının Modellenmesi: Çiftlikköy Kampüsü Örneği", *Türkiye Fotogrametri Dergisi*, *1*(1), pp.11-16, 2019.

[23] Şenol, H. İ., Ernst, F. B., & Akdağ, S., "Kentsel Dönüşüm Alanlarının Geotasarım Yöntemi ile Planlanması: Eyyübiye Örneği", *Harran Üniversitesi Mühendislik Dergisi*, *3*(3), pp. 63-69, 2018.

[24] Ernst, F., Erdoğan, S., Yılmaz, M., Ulukavak, M., Şenol, H. İ., Memduhoğlu, A., & Çullu, M. A., "Geodesign For Urban Planning – The Example Of Harran University's Campus Masterplan", *International Journal of Environmental Trends (IJENT)*, *3*(1), pp. 17-30, 2019.
[25] Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., & Çöltekin, A., "Applications of 3D city models: State of the art review", *ISPRS International Journal of Geo-Information*, *4*(4), pp. 2842-2889, 2015.
[26] Buyuksalih, I., Bayburt, S., Buyuksalih, G., Baskaraca, A. P., Karim, H., & Rahman, A. A., "3D Modelling and Visualization Based on The Unity Game Engine-Advantages and Challenges", *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, *4*, 2017