*Journal of Universal Computer Science, vol. 22, no. 6 (2016), 836-855*
*submitted: 30/11/15, accepted: 28/5/16, appeared: 1/6/16 © J.UCS*

# Parameterized and Dynamic Generation of an Infinite Virtual Terrain with Various Biomes using Extended Voronoi Diagram

**Kazimierz Choroś**
(Wrocław University of Technology, Wrocław, Poland
kazimierz.choros@pwr.edu.pl)

**Jacek Topolski**
(Wrocław University of Technology, Wrocław, Poland
jacek.topolski@pwr.edu.pl)

**Abstract:** The paper describes and extensively evaluates a new method for the parameterized and intelligent generation of an infinite environment including various biomes in a virtual 3D space defined by the user. The biomes might be generated with different set of textures and by using different formulas to form shape of the landscape. Despite different shapes they still blend smoothly between each other. To achieve this goal Gaussian blur and Voronoi diagram algorithms are used. To enable additional parameterization for biomes placement the standard Voronoi algorithm is extended by including cell change along an axis and dispersion of Voronoi cells. Applying latitude into terrain type produces more realistic results. An entire terrain is generated on the CPU using a separate thread to eliminate stuttering during calculations and then the data is sent to the GPU in order to draw it. It forces reducing amount of data as much as possible, because data must be sent through the graphical bus. The tests have been performed by setting up a sample terrain and performing basic actions on this terrain like moving or rotating to gather frame times. The results showed that although the method demands much memory it is efficient and suitable for the real-time processing.

**Keywords:** Virtual 3D Spaces, Virtual Biomes, Infinite Terrain Generation, Real-Time Rendering, Extended Voronoi Diagram, Gaussian Blur, Terrain Patch Generation, Virtual Terrain Parameterization
**Categories:** H.5.1, I.2.10, I.3.7, J.7

## 1 Introduction

Computer visualizations, virtual reality systems, computer animations, computer games, simulators, dynamic representations and visualizations of industrial processes, as well as many other software applications and systems demand the generation of virtual and as much as possible realistic terrains in three-dimensional spaces (Shiau, 07], [Mat, 09], [Reed, 10], [Ruzinoor, 12], [Yannakakis, 15]. The bigger and in addition more complex terrains for virtual sightseeing are now expected. They should also include various biomes. Biomes are defined as contiguous areas with similar climatic conditions. Such geographical communities of plants, animals, and soil organisms (flora and fauna) are often referred to as ecosystems.

Three-dimensional space (3D space) is a geometric model where a point is described as an ordered triple containing coordinates which the point is placed on. Their coordinates are represented by real numbers. Every object is composed of many such points called vertices forming their shapes, which are later overlaid with miscellaneous colours and texture coordinates. 3D space can represent real world very intuitively. Therefore, this model is generally chosen to visualise photorealistic and interactive spaces. Before flattening the 3D space onto the screen we need to transform all vertices to the projection space which stores the whole space in a cube. To display the defined vertices we have to go through the world, view, and projection spaces which are basically a consequence of multiplication of the transformation matrices. We need a world-view-projection matrix which transforms the model vertices from the model space to the coordinates on the 2D screen.

Realistic terrain visualizations generated by rendering of the 3D graphics involve enormous calculations [Akenine-Moller, 02], [Choroś, 08]. However, a real-time rendering has still some limitation, especially in computer games, simulators, or other interactive visualizations where current devices impose on saving resources for also other things like animations, artificial intelligence, or game logic.

The modelling and simulation in 3D virtual and augmented reality systems become very significant in the visualisations of industrial processes. Simulations provide visual information on the behaviour and performance of existing systems as well as of enterprises planed for the future. 3D dynamic representation of industrial processes are produced in a virtual reality, where the real and virtual elements have direct and synchronized connections with each other, extending the features of the real components with the ability and services of these new virtual elements. Virtual system provides faster planning, easier system integration, and more reliable operation and control [Kopácsi, 13].

Terrain is usually represented as a manually or procedurally generated heightmap [Cassol, 11]. Some approaches use hybrid techniques.

In an infinite virtual world the terrain is boundless. But in the majority of different implemented virtual worlds the terrain is limited for example by impassable borders like a valley bounded with mountains or an island bounded with water. The real infinity is difficult to achieve because of technological restrictions. Therefore, the infinity of virtual worlds is usually simulated using different sophisticated techniques of management of the terrain parts. For example the virtual world can be looped by using an abstraction of torus model called wraparound [Wolf, 01], because it's impossible to reach the end of a torus as it basically doesn't exist. The torus is usually identified with a planet which can be encircled. Current hardware with the hardware tesselation allows producing very complex planetary scenes using wraparound model. Such a technique is called a planet rendering and it maps a cube onto a sphere. The main problem of planet rendering is to maintain the real size of a planet because it requires to preserve large distances and many details. However, the application of fractals for the generation of details may create a very realistic landscape [Cozzi, 14]. Similarly, the wraparound approach can be used but around a cylinder, which means we have boundaries only on the top and bottom of the map.

The infinity can be also simply simulated by generating the subsequent parts of the terrain just before approaching the borderland of the already generated landscape. Such a solution has been implemented in the Minecraft game [Minecraft, 14]. It generates infinite terrain without repetitions using procedural algorithms. In that case the problem of begin-end edge welding does not occur as due to the noise coherence it is easy to connect already generated parts with the new ones.

In this paper we are going to choose the approach based on a full landscape infinity (without wraparound or planet rendering), moreover taking into account the latitudes of terrain parts and their types. Because it is not possible to create a terrain manually or by using some existing maps, so we have to rely on the procedural generation techniques. The goal of this paper is to analyse a method of infinite terrain generation including various biomes and to enable additional parameterization for biomes placement taking into account terrain type and latitude. The results of exhaustive tests performed with this new method demonstrating its advantages and efficiency will be presented and discussed.

The paper is organized as follows. The next section describes some related work in the area of the generation of infinite terrains in 3D virtual worlds. The main idea of the Voronoi diagrams is outlined in the third section. The method of the dynamic generation of an infinite terrain is presented in the fourth section. In the fifth section the extended Voronoi diagram is presented including its parameterization by cell change along an axis and dispersion of Voronoi cells. The sixth section presents the results of extensive tests of quality and efficiency of this new method of parameterized and dynamic generation of an infinite virtual terrain with various biomes. The final conclusions are discussed in the last seventh section.

## 2 Related Work

There are many papers on the generation of virtual terrains but unfortunately only a few solutions have been proposed for the generation of infinite terrains. Most frequently it was enough to create a finite landscape with the fixed size, although very large [Lin, 09]. Such a terrain is easier to maintain and process, because we get complete information on terrain elevations that may be used to logically distribute terrain features and special objects, whereas infinite terrain implies generating its content partially, so it limits our knowledge about landscape [Luna, 06].

The problem of the infinity of 3D terrains has been thoroughly discussed in the Dollins dissertation [Dollins, 02]. In this dissertation hash-based algorithm was used to manage the adequate parts of the generated terrain. The terrain parts were procedurally generated on-the-fly without necessity to save them in a file, so every time the visitor came back to already visited place the adequate part of the terrain was generated once more. These parts were regular grids with no adaptation. The only additional improvement was applied for terrain parts more distant from the camera consisting of diluting the very distant grids.

Unfortunately, the terrain generated using this method is rather monotonous because its every part is generated by the same procedural algorithm and the whole terrain is covered by the same texture. However, the colours are changing depending on the height of a vertex, finally the colours are blended with the texture to produce pixel colour. So, the main disadvantages of this method are a low diversity of the landscape shape as well as uniform and unvarying colours. Such a solution seems to be sufficient for some applications like flight simulators where the details of generated landscapes may be not very important. But the expectations in many games or virtual visualizations are much higher and the texturing of landscapes in such applications should be much more refined and realistic.

Greuter et al. [Greuter, 03] proposed an approach to procedurally generate pseudo-infinite virtual cities in real-time. These virtual cities contained highly diverse and complex buildings generated as required, on the fly as they are encountered by the user. The building generation parameters were set using pseudo-random number generator, seeded with an integer derived from the building's position. The city could expand to an extent that would require a lifetime to explore, and was called by the Authors as pseudo-infinite.

In [Schneider, 06] the authors have proposed a new method for real-time editing, synthesis, and rendering of infinite landscapes exhibiting a wide range of geological structures. This solution is based on a concept of projected grids which is frequently used for sea or ocean rendering. This technique creates a surface expanded up to the horizon using a grid with fixed number of vertices.

The concept of projected grid is quite intuitive and means that we define a grid covering the whole screen which is then projected (overlaid) onto terrain. The grid vertices are conformed to the terrain shape preserving roughly the same triangles size in screen space giving free LOD and creating far distance view (up to the horizon). This scheme may be also used on a spherical surfaces, e.g. for planet rendering. The technique requires an additional frustum called projector frustum which specifies the extended range of view to eliminate artifacts, but at the same time it makes the algorithm a little more complex. One of the main drawbacks is that we need to know the maximum vertex height of a terrain. Using that information the algorithm knows how to set up the projector frustum properly, otherwise we would see defects on the screen edges caused by not generated geometry. Projected grid is in fact computed on the GPU, but also in this approach it is difficult for the generated terrain to create various areas with different shapes and textures.

Procedural approaches seem to be the most attractive for the generation of infinite terrain [Raffe, 12]. A procedural generation means that the content is created algorithmically rather than manually. Procedural methods are employed in generation not only of landscape but also vegetation, buildings, fire or water and they give the opportunities to create very diversified virtual worlds of different scales. Whereas manual methods are time-consuming proportionally to the size of the landscape.

Procedural generation techniques differ in the visual aspect and computational complexity, as well as in application areas. Many of these procedural techniques, mainly applied in computer games, are described and discussed in the surveys presented in [Smelik, 09], [Carli, 11], [Hendrikx, 13]. Also the number of implemented systems with landscape generation is rapidly growing. Most of them like [Livny, 09] or [Losasso, 04] use GPU for generating a terrain, because such a solution does not require transmitting a lot of data from CPU to GPU. The techniques mainly performed on GPU are much more efficient and easier. In our method vertex buffers are not used because dynamic buffers demand a lot of memory. However, despite its advantages we are going to use the CPU for generating terrain final data as current CPUs are fast enough to generate the data for landscape in runtime and by using few manners it is possible to reduce data as much that it is small enough to send it to the GPU when crossing terrain quads. Furthermore, there is still a lack of mechanisms that handle different biomes in a flexible way, for which it is much harder to create logic on the GPU side.

Realistic biomes incorporate correct plant placement which many approaches has been already proposed for. Terrain parameters [Hammes, 01] can be used to simulate a realistic placement of plants. Another solution has been proposed in [Deussen, 98] where a special technique has been used for distribution texture to generate black and white dots in conjunction with Voronoi diagram for relaxation of the dots. The black dots represented plants, but the plant type must have been determined from additional parameters. Another texture with the plant details was used, but at this step it seems that it would be better to use the terrain parameters instead of a texture. Both of the mentioned solutions have the disadvantage that they don't involve creating separate special biomes which may be desirable in some games or visualizations. On the other hand they provide a great addition for plants placement used locally in already created biomes. The challenge in procedural approaches is to place the objects in the way that they fit the ground and environment well. Objects can be put on the ground by mixing the ground textures with the object textures. However, it might not be suited for all kind of objects due to the primitive merging with the ground using simple gradient.

Recently a new method (unfortunately the original paper describing this method [Liu, 11] is written in Chinese) of an infinite terrain generation has been applied in the unmanned aerial vehicle real-time flight simulation and training system [Wei, 13]. In this new method various ecological phenomena of different altitude zones can be simulated in the rendering process. For example in the mentioned plane simulator such effects as rain, snow, and explosions were generated, and furthermore, realistic sound effects were provided of different frequencies adequate for the engine speeds.

And finally a very recent solution [Parberry, 14] proposes to generate geotypical infinite terrain based on elevation statistics acquired from widely available sources such as the United States Geographical Service. To generate procedural terrain that shares height characteristics with real terrain the Perlin noise method [Perlin, 02] with elevation data has been applied.

The Perlin noise method can be executed using the parallel algorithm which combined the characteristics of the original algorithm with parallel processing [Li, 15]. The Authors have shown that such a parallel approach is very efficient for terrains of large sizes satisfying requirements for generating massive terrains.

Whereas in this paper, our method of the parameterized and dynamic generation of an infinite terrain with various biomes in a virtual 3D space will be analyzed. The method uses the extended Voronoi diagram as well as the Gaussian blur and takes into account the latitude. The goal of the tests was to confirm that the proposed extended version of the Voronoi diagram can be applied in a real-time process.

## 3 Voronoi Diagram

Voronoi diagrams are used to divide the virtual 3D space into regions called simply Voronoi cells. Each of these cells contains a point called seed, site, or generator. To create cell distances are calculated using one of the distance functions. The cell is created in such a way that each point in the cell has the lowest distance to the seed than to any other.

Let T be a non-empty set from Euclidean space with a distance function d. Let P be a set of all seeds and x, y are the indices of the seeds. The equation (1) produces a set of points creating the Voronoi cell for the given x-seed

$$T_x = \{t \in T \mid d(t, P_x) \leq d(t, P_y), \forall y \neq x\} \tag{1}$$

There are many distance functions like Karlsruhe metric, Chebyshev distance, Minkowski distance, etc. which can be applied to create the Voronoi diagram. The most frequently used is the Euclidean distance function. It determines a straight line drawn on the halfway of the segment connecting two neighbour points and perpendicular to this segment. The straight line runs until crossing another straight line created from another two neighbouring points.

The pseudocode of the Algorithm 1 produces the original Voronoi diagram. For the given position (in our case we use 2D space) we search for the cell for which the seed point is the closest to the given position. Therefore, we have to iterate over the neighbouring seeds to calculate the distances. The seeds are in random positions, although they are determined, which basically means that for the same given position the algorithm uses the same seed positions. The function *valueNoise2D* generates random and determined value for the given parameters in the range of [-1,1], whereas *calculateDistance* is the function for calculating the distance. The *generated2DNoise* is a function returning final noise value in the range of [0,1]. Using the Euclidean distance it is not necessary to use the original equation as we only need to compare distances, thus we can optimize it using the *Squared Euclidean distance* which doesn't apply the final square root to the result.

**Algorithm 1**: Voronoi diagram algorithm

```
input data: x, y
RANGE = 2
for j := ⌊ y ⌋ – RANGE to ⌊ y ⌋ + RANGE do
    for i := ⌊ x ⌋ – RANGE to ⌊ x ⌋ + RANGE do
        xSeedPos = xCur + valueNoise2D( xCur, yCur, SEED_FOR_RAND )
        ySeedPos = yCur + valueNoise2D( xCur, yCur, SEED_FOR_RAND + 1 )
        xDist = xSeedPos – x
        yDist = ySeedPos – y
        distance = calculateDistance(xDist, yDist)
        if distance < minimumDistance then
            minimumDistance = distance
            xCandidate = xSeedPos
            yCandidate = ySeedPos
        end if
    end for
end for
return generated2DNoise(⌊ xCandidate ⌋ , ⌊ yCandidate ⌋ )
```

Among numerous applications of the Voronoi diagram there are such as applications in mining for estimating the reserves of minerals, in climatology for calculating the rainfall for an area as well as in ecology, architecture, or even machine learning. The practical applications of the Voronoi diagram are described for example in [Okabe, 09]. In our method [Choroś, 15] of infinite terrain generation the Voronoi diagram is used to simulate spreading area types over the landscape.

There are few algorithms for producing the Voronoi diagram like Fortune's algorithm [Berg, 08], Lloyd's algorithm (this one produces evenly-spaced cells, that's why it is sometimes called a relaxation), or algorithms producing the Delanuay triangulation like Bowyer-Watson algorithm or S-hull [Sinclair, 14]. Due to the duality between Voronoi diagram and Delanuay triangulation it is possible to construct the diagram from the triangulation in linear time. Because we need to control the diagram in a specific way to apply some parameters for virtual terrain with various biomes an extended version of the Voronoi diagram has been proposed [Choroś, 15]. In the next section we will recall only the main features of this extended version of the Voronoi diagram. Then the parameterization of dynamic generation of an infinite virtual terrain with various biomes using the extended Voronoi diagram and the results of extensive tests of quality and efficiency will be presented.

## 4 Infinite Terrain Generation

In the proposed approach and analyzed in this paper the CPU is used for terrain generation. This solution has been chosen because nowadays computer equipment is very effective, mainly the CPUs installed in computers are sufficiently fast to generate the adequate data for virtual terrain generation in runtime. Furthermore, the generated data are intelligently reduced to such extent that they can be continuously sent to the GPU when traversing the virtual areas [Choroś, 15]. Another reason is that at present we do not have any mechanism to easily handle different biomes the more that it is difficult to propose an adequate logic for the GPUs.

It is not easy to maintain the whole mesh at the same level of detail (LOD), so, the terrain is partitioned into quads of the same size. Then, quads have some parameters, for example a parameter used in LOD effects or a parameter applied for removing cracks (T-vertices) on the terrain. This parameter determines the position of the quad, i.e. it determines whether the quad is on the left, on the right, or in the corner, etc. It leads to the generation of the grid with modified vertices on edges and which fit to another quad with different quality level. The Figure 1 presents a matrix preserving real positions of quads and illustrating how the quads are organized.

Vertices are characterized by three elements: position, normal vector, and weights of areas. The weights of areas are stored in an array containing values determining the proportions of every type of area in the vertex. It is a kind of transition between such areas as a desert, a grassy area, or rocky area. For example at some point the weight of a sandy area is 80% and a grass area has the weight equal to 20%, so it means that this area is much more similar to a desert than to a grassland. Moreover, vertices heights are also used to calculate a maximum and minimum vertex height and then to create a cube used for the frustum culling. All these data are stored in a file containing whole terrain ensuring the opportunity to load the generated quads from a file in runtime instead of generating them once more.

The generation of an infinite terrain does not demand the same highest quality of every part of the terrain. The layers applied in this method increase a flexibility and possibility to investigate the best configuration for quality and level number. The set

L of layers is defined as L={A, B, C, ...}, where A, B, C, ... ϵ **Z** and the number correlates to the number of layers for the given quality. It means that the A is number of layers for the highest quality level, B is the number of layers for medium quality level, etc.

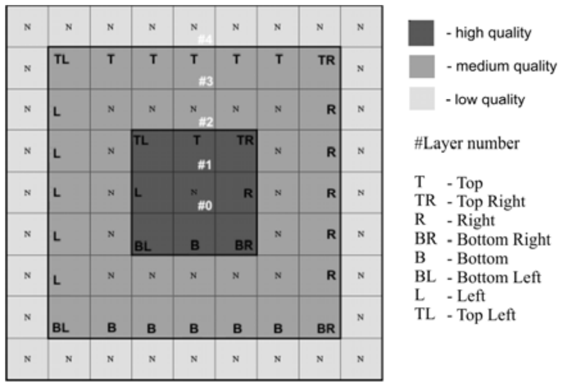


*Figure 1: Terrain quads organization for five layers L={2,2,1}*

At a given moment of the generation process we are in the certain quad of the terrain, seen as a central quad. When the position of camera changes the quads visible around should be updated and their quality should also be changed. To reduce the frequency of these updates and in consequence to reduce the amount of data sent to the GPU the quads will be updated when we move to another quad. Crossing a quad at the top edge demands loading new quads in the first row of the matrix but also freeing memory from the quads at the last row. We should also update the quality and position parameters of other quads in the matrix. A generic solution is possible to be used to update the quads with such variability using the L array because of the similarity of the squares.

The algorithm of determining position parameters marks the quads that are not on the edges with another quality level, so they should not be modified. This algorithm sequentially produces an array with quad positions for every layer.

Such an array enables us to use the outward spiral method [Wolf, 01] for iterating over the quad matrix to assign the parameters to its corresponding quads. The outward spiral method has already been used to render quads. The spiral method also ensures a front-to-back rendering [Fernando, 04]. In the front-to-back rendering the objects closer to the camera are first rendered. It results in overlaying of other objects, in reducing calls to the pixel shader, and in saving rendering time. Therefore, in our method the closest quads are rendered earlier and further quads later as well.

## 5 Parameterization of Virtual Terrain Generation

In the proposed approach the Voronoi cells are identified as biomes. Moreover, several parameters are introduced such as biome size, speed of latitude change (which will affect type of areas), and dissolution.

The biome size parameter determines the cell size in the Voronoi diagram. Although all cells are generated of the same size without any random variations, however other parameters cause the area size differentiation.

To do it we just need to divide the input data (x and y) in the Algorithm 1 by some size factor. The larger the factor is, the bigger will be the cell. If cells are small the border between biomes are more straight, because they are more fragmented. The borders of biomes are not straight lines in real worlds, but they are some curvatures, so we should avoid very small cells. Furthermore, cell size has also influence on the speed of latitude changes. The changes are slower as the cell size grows.

The main problem is to change area type according to the given position and to propose the procedure for the situation when the last zone of area is reached. There are two possible solutions – after reaching the last zone we can start once more from the first zone. In such a case the terrain will be a sequence of zones like tropics-tundra-arctic zones, then once more tropics-tundra-arctic zones, so after the arctic there will be tropics. However, such a zone change is not realistic. Much more adequate for real worlds is the sequence like tropics-tundra-arctic-arctic-tundra-tropics. Some zones are used twice and in this example an arctic zone would be generated rather too big. To avoid such big zones a smaller range of latitude should be specified.

If x is the position value from the axis along the latitude and S is the speed of latitude change the function (2) can be applied to specify the factor used for changing biome type according to the given position.

$$f(x) = \begin{cases} \frac{x}{S} & \text{if } x < S \\ -\frac{x}{S} + 2 & \text{if } x \geq S \end{cases} \tag{2}$$

Before processing the position value by the function (2), we need to prepare the input position in such a way that it makes the function periodical. The easiest way is to use modulo $x = |x| \bmod (S \cdot 2)$. Having such function we can apply it to the Voronoi algorithm as a step to perform latitude change, where the x is the seed's horizontal/vertical position. Therefore, the output of the f(x) function will be the output of the method generating the Voronoi diagram.

The second parameter proposed in the extended Voronoi diagram ensures that as in the real world arctic and tundra areas are mainly generated on the high latitudes, whereas tropical and grass area rather in the terrain parts of the lower latitudes.

The third parameter is connected with a dispersion. To ensure the irregular spreading of biomes high values for the dissolution should be used to eliminate the straight connectivity line between cells of different types and to produce more diverse and less predictable world. It significantly improves the final look of a landscape.

Such feature provides more randomness to the map resulting in more interesting world. This feature is very easy to implement, because it requires only clamping the f(x) values to the range of [S–d, S+d], where d is the factor of dispersion.

After introducing these parameters we can extend (Algorithm 2) the Voronoi standard diagram to provide the complete improvements for distributing the biomes. It does not change anything inside the original algorithm, but only process its final output further to incorporate mentioned parameters. We can notice that we are in a very comfortable situation because we don't need to rely on neighbouring values. Moreover, this approach let us easily provide an additional functionality that is turning on or off some of the parameterization processing, so it may be even more flexible for end users.

**Algorithm 2**: Extended Voronoi diagram algorithm

```
input data: noise – final value generated from original Voronoi algorithm
input data: yCandidate – variable calculated from previous Voronoi algorithm

latitude = $\lfloor |yCandidate| \rfloor$ mod (S • 2)
if latitude <= S then
    latitudeGradientValue = latitude / S;
else
    latitudeGradientValue = – latitude / S + 2;
end if
noise = clamp noise into [latitudeGradientValue – d, latitudeGradientValue + d]
return clamped noise into [0,1]
```

The cells generated by the Voronoi algorithm have different shapes and thereby the terrain becomes more realistic. Each cell is of uniform colour. In our approach every area type has its specific colour, so the number of colours is equal to the number of area types. Then the terrain is rendered using the Gouraud interpolation algorithm. Between two vertices this algorithm produces very sharp edges, so we need to smooth them.

To create smooth transition between two areas we need to somehow blur them. It is possible to achieve such effect using well known method – Gaussian blur, although it reduces a lot of other details. The method uses a concept of weighting neighbouring pixels, which basically boils down to generating a matrix of weights (called a convolution matrix), which is used for the currently processed pixel and its neighbours. Dimension of the matrix is a kernel size, which results in strengthening blur as the kernel size increases. Every pixel is given a weight which means how much value of the pixel is taken into the final pixel. Such weights are generated using the Gaussian function [Nixon, 08]. Pixels closer to the centre have higher weight value making their values more important for the final pixel colour. Theoretically, the function never produces zero value for any given x, y, but practically after certain kernel size the values are getting so small that can be considered as zero, hence we can ignore higher kernel sizes.

The Gaussian blur has been used in our approach because it is very suitable way in the case of connectivity problem. However, every blur which produces smooth results might be used.

Our algorithm produces the results in 3D array forms. A weight is a number representing an area type. To ensure during the rendering process the proper blur on the edges of the visible parts of the terrain, that is to ensure for example the transition within 10 vertices, the Voronoi map should be extended by 20, i.e. 10 vertices from two sides, in height and width.

For different area types we can use different 2D generators like improved Perlin noise [Perlin 02], simplex noise, or other noise-based generators (as well as not noise-based generators). Because the connection on the edges is realized by the Gaussian blur at earlier stage they even don't have to fit on the edges. Furthermore, if the terrain geometry is diverse with different ecosystems the mesh is usually precalculated before rendering, but we can generate such terrain geometry on-the-fly without precalculating anything before rendering.

The last remark concerns the problem of texturing. Weights enable us to process different textures independently by having an array of textures. Unfortunately, when rendering a vertex on a border (on a transition between different areas) as many textures as there are weights greater than 0 should be sampled and the results should be combined to produce smooth transition. The 2D texture array is used as it is a convenient way to access appropriate texture when performing height-based texturing. Every texture can have its own chain of mipmaps. By using texture array it is easy to determine what texture should be sampled on a given vertex.

Many landscapes also consist of hand-made objects to make them closer to the reality. Sometimes it’s because they simply can’t be generated algorithmically or need to have some special look, particular for storyline (we are aiming objects that exist in the nature like flora or fauna). Such objects must be put on appropriate places, but we definitely don’t want to spread them manually – in fact we can’t do that, because the terrain is infinite.

Fortunately, in our approach with weighted vertices we can simply generate a random position and see what the weights are for the vertex on that position, because the weights tell us everything about biome type, thus we can choose the most suitable object. Moreover, every vertex contains height, so we can take it into account when selecting the object. We don’t have information about slope though, so we have to access neighbouring vertices to calculate it. This may be a bit uncomfortable when we generated vertex position on a quad edge, because we have to jump between different quads. Additionally, it may get a bit trickier if the quad is on the edge of the generated map, because we don’t have the information about further quads yet, so we prefer to avoid generating the objects on the edges of the last layer, but just generate the objects there when we are approaching this place.

## 6 Tests and Results

Due to this new approach a landscape with various biomes can be generated and moreover these biomes blend smoothly between each other. Also another area type can be added however such a procedure demands the implementation of a specific interface for introducing area latitude and the height for a given position of a vertex. Then the pixel shader should be modified to include a new area. To alleviate this inconvenience it is suggested to use Hammes technique [Hammes, 01] for texturing. Next this new implemented object can be added to the engine and then the engine is able to spread the areas with given parameters.

### 6.1 Testing Environment and Methods

The tests have been performed in the computer environment with the following parameters:

- motherboard: Asus P5K SE,
- CPU: Intel®Core$^{TM}$2 Quad CPU Q6060 2.40GHz,

- GPU: NVIDIA GeForce 8800 GT 512MB (driver version 340.52),
- RAM: GeIL CL5-5-5 DDR2-800 (400MHz),
- OS: Windows 8.1 64bit.

Our scene was set up using 17 layers covering around 4.5 square kilometres of visible terrain. On the edge of the last 17th layer we have 33 quads of the size 65 x 65. We assumed 1 meter distance between vertices in highest quality. Taking into account that consecutive welded sequentially quads have common vertices, the area size was calculated as follows: $(33 \cdot 65 - 32)^2 = 4{,}464{,}769 \text{ m}^2$ (about 4.5 km$^2$). The layers that have been used were L={2, 1, 2, 4, 8} and we placed camera 100 meters above terrain to include all of the added biomes. Our view included 234 (viewing frustum has been analyzed by special additional procedure) of 1089 quads with the following numbers of layers for given quality {4, 6, 16, 48, 160}. The biomes were placed randomly but determined, that means we had the same biomes placement during all tests. Depending on test type we were changing the numbers of layers and biomes to provide complete overview of performance impact. Gaussian kernel size was set to 35 – this value seems to have pretty good quality/speed ratio, although the value does not matter in the tests of frame times because it impacts only terrain generation time.

To calculate efficiency we used standard mechanism for calculating time needed to draw a frame (*FT*) implemented in our engine. We have resigned from calculating frames per second (*FPS*) as it provides non-linear and unreliable results [Mali, 11]. The tests included different aspects like idling, moving, or turning around. To obtain the results we ran our engine (using 1024x768 resolution) for some time to perform the actions and to take the frame times in order to prepare graphs.

In order to measure memory usage we used build-in software from Windows 8.1 called *Resource Monitor*. Provided memory results were taken from *Private (KB)* column. Results were generated by simply running the program with different values for numbers of layers and biomes. They are not affected neither by camera nor biome placement nor quad qualities.

## 6.2 Quality

The method with applying latitude into terrain type may produce more realistic results. The visitors may not observe any immediate change of biome types when moving over terrain. Such terrain feature doesn't distract them and makes the terrain look good from the flying perspective. Figure 2a presents the terrain with three biomes without latitude change whereas terrain on Figure 2b has been generated taking into account latitude. The inclusion of a latitude parameter ensures that on the north we have arctic biomes (white) while tropical biomes (sandy) on the south. Taking into account latitude results in a much more realistic view of generated terrains.

This significant quality improvement observed on Figure 2b is the main and crucial advantage of the approach based on the extended Voronoi diagrams. Instead of randomly dispersed parts of the landscape we have generated the terrain much more realistic, consistent with intuitions and observations of reality.

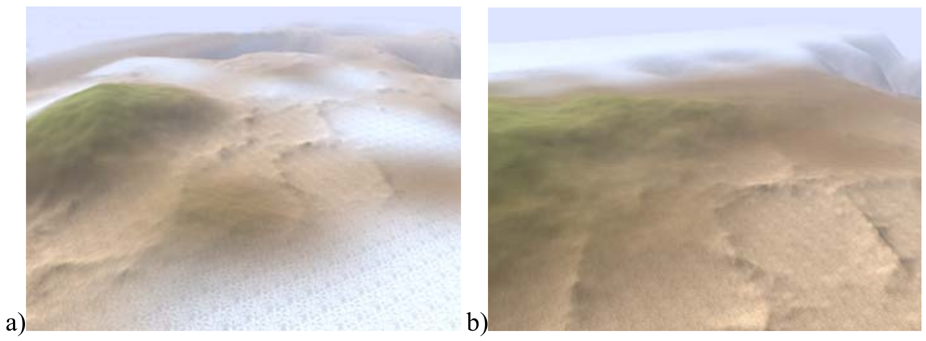
a) b)


*Figure 2: Quality improvement of a rendered landscape. Every biome has another shape generator and set of textures: a) three biomes without latitude change b) three biomes with latitude change*

### 6.3 Flexibility

To add another biome we need to implement an interface containing methods for returning biome placement (latitude) and height for given position of a vertex. In the implementation used to generate a given biome the height generation is completely independent of other biomes.

After that we have to modify our pixel shader (and vertex shader along with input layout to pass the new weight to pixel shader) to include the new biome. We have to include proper texture and weight from newly added biome into texturing mechanism. Finally, we have to add that implemented object to the engine via one method and the engine is capable of doing all of the work by itself to spread the biomes with given parameters.

The procedure of including a new biome can be improved even more. When there are many biomes and we want to move the biomes slightly, it would force us to go through the classes and fix the latitudes. A better solution would be to define the max latitude when adding the biomes to the terrain description.

### 6.4 Efficiency

The solution proposed in this method ensures that the geometry of every frame is not recomputed by the GPU. Nevertheless, it may be a risk that the FPS characteristic will significantly diminished in the possible situation when many biomes are visible at the same time and a lot of textures is sampled. It may especially happen when in blurring process a big kernel size is used producing many small weights and sampling several textures for most of pixels. For that reason the pixel shader should include bias for biomes weights because then the kernel size of Gaussian blur has not so great influence in the cases of many visible biomes. In the first tests of the efficiency of this approach the performance times have been measured of the generation of one quad of the size 65x65 including the following actions: creating the Voronoi map with area types, performing the Gaussian blur, preparing structure with data for a quad, saving quad to a file, and finally sending the quad data to the GPU. Every test was repeated ten times to finally get averaged value. The generation of one quad takes 19.527

milliseconds including 4.456 ms for creating Voronoi map with area types, 9.065 ms for performing Gaussian blur for smoothing areas borders (kernel size = 20), 2.608 ms for preparing structure with data for a quad (assumed height to be 0 to make it independent from noise generator), 3.044 ms for saving quad to a file, and finally 0.354 ms sending quad data to GPU.

These results show that in the case of a terrain with 17 layers and using the Gauss kernel size equal to 20 all 1089 quads are generated in about 21 seconds. In effect this time is much shorter because when moving over the terrain we need to generate only quads at one edge or two – if moving diagonally. Therefore, only 33 (or 65) quads should be generated. We can move over the terrain very quickly because it takes only 0.644 sec (or 1.269 sec). The second observation is that half of the time is needed to perform the Gaussian blur, so the implementation of blurring has a great influence on the total performance time. The last observation is that the quads already generated are saved into a file, so coming back to already visited parts of the terrain is limited to the loading of the quad in only around 1.156 milliseconds.

The graphics on Figures 3–6 present frame times of chosen actions. For each action we gathered results from first 1000 frames and then created groups consisting of 5 values for which we calculated an average, so overall we have got 200 points. The terrain included 4 different biomes.

Figure 3 shows results from idling, that is standing still and looking at the terrain. The average value for this action was 8.542 ms.

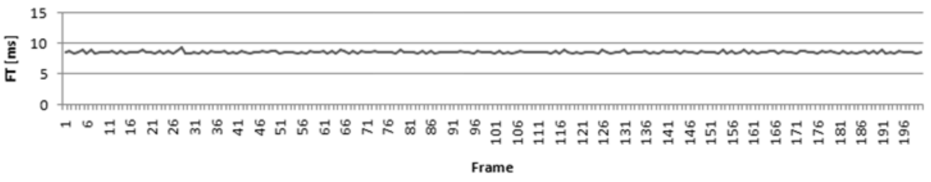


*Figure 3: Frame times when idling*

Figure 4 shows the results from moving straight on terrain towards one direction. The average value for this action was 6.740 ms. Average value is smaller than when idling, because when moving we are getting closer and closer to the next quad which hides little by little the quads with the highest quality until we reach the next quad. Frame times are getting smaller and at some points it jumps increasing the frame time when we moved into adjacent quad.

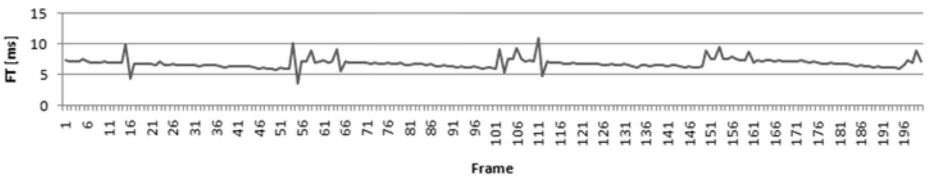


*Figure 4: Frame times when moving*

Figure 5 represents results from turning around very quickly. Our engine handles quick turnarounds without displaying any incomplete geometry. The part of the graph is with bigger values because our generated testing scene contained a hill next to the camera, which exposed more pixels to draw. The average value was 8.325 ms.

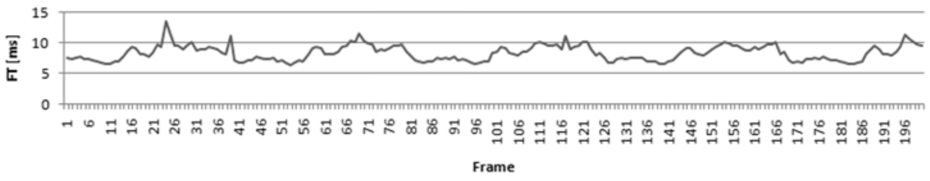


*Figure 5: Frame times when rotating quickly*

Figure 6 represents results when performing moving together with rotating. The results may be a bit sharper than on previous graphs, because we are using lazy approach to send data to the GPU. It means that when we moved to adjacent quad moving backwards, the generated quads are not sent immediately to GPU, but it is done when a quad gets into view of camera. The average value of this test was 8.626 ms.

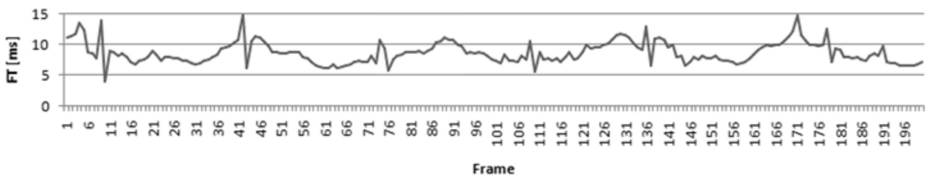


*Figure 6: Frame times when moving together with rotating quickly*

All functions have one characteristic property – they are similar to periodical functions (more or less), so that means the frame rate is stable and does not produce surprising freezing. However, the values range between 3.41 ms and 14.876 ms at some peaks. This is due to the mapping main thread on the GPU to be able to send data from the CPU to the GPU (when generating new quads), this is why we have to reduce the data as much as possible.

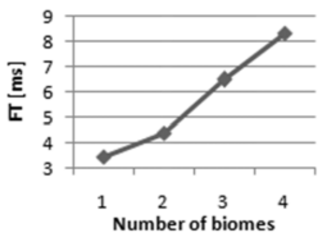


*Figure 7: Frame times for a given number of biomes*

Using a lot of biomes incorporates sampling a lot of textures, especially when many biomes are adjacent to each other. For example in the case of a square with four biomes of the same size inside filling up the whole space, the pixel will need to sample textures from all biomes in the centre of the outer square (where all of the inner squares are connected). The function on Figure 7 presents averaged frame time according to the number of biomes visible at once. From the graph we can notice that adding new biomes increases frame time almost linearly – about 2 ms. Although this is also affected by the Gaussian kernel size, therefore on Figure 8 we present frame times using different sizes for kernel, we took averaged value of 5,000 frames during idling. We repeated the test with different cell sizes – 100 and 40.

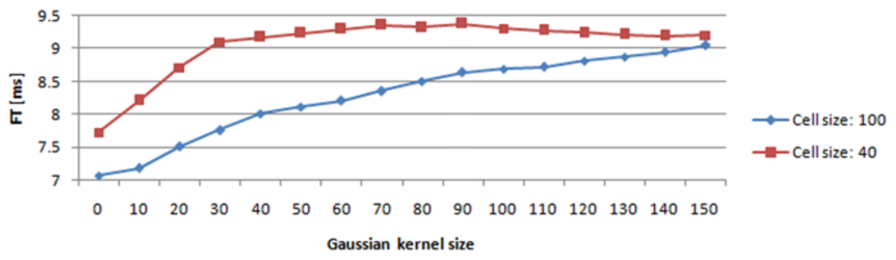


*Figure 8: Frame times for different kernel sizes*

It may be easily noticed that at some points the performance penalty starts to soften. It is due to the fact that many pixels have already sampled most of the textures, so in result we can see smoother terrain with less frame time impact. We can also see that the frame times are getting converged further we go.

Then we have tested the spiral rendering method. We used terrain generated with many hills that were hiding some parts of terrain and we were rotating the camera. The results were compared with using simple iterating over the quad matrix (from top left to right bottom) and presented on Figure 9. On that Figure we can clearly see that frame times from about 15 to 50 and from 155 to 200 are much higher for non-spiral method. This is the situation that some parts of terrain were hidden behind hills, but were still rendered completely. The inconvenience here is that we have to check all quads whether they are visible on the screen.

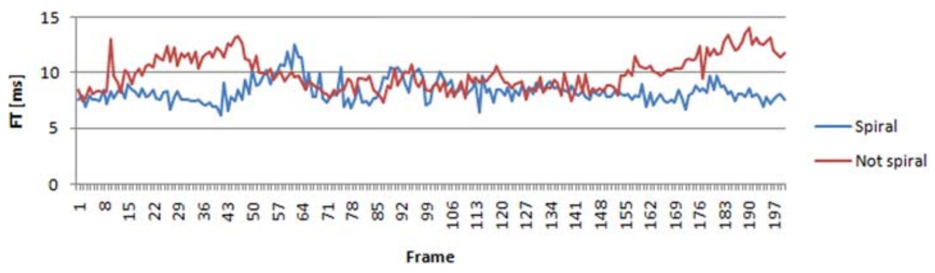


*Figure 9: Frame times for rotating action for cases when spiral order rendering is turned on and on*

Regarding to the quadtree method it is able to eliminate the quads behind us almost without any cost, but it has to check more "virtual" quads when the real quad is in view. This is because the quadtree approach packs the smaller quads into bigger ones, so before it determines whether the quad is visible it has to check whether those bigger ones are visible too.

Unfortunately, as it was already observed this method requires a lot of memory. However, we may discontinue to generate next new layers at around 500 MB memory usage and the generated terrain will be large enough for most cases. The graph on Figure 10 [Choroś, 15] shows memory usage depending on the number of layers.

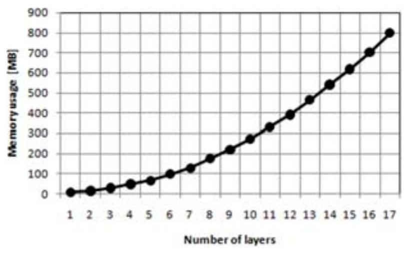


*Figure 10: Memory usage*

## 7 Conclusions and Future Work

The method presented and analyzed in the paper provides an intelligent solution for generating an infinite terrain with various biomes. It ensures a smart way for determining a biome landscape shape and ground texturing as well as exposed information necessary for spreading flora and fauna very easily. We used the CPU for generating terrain final data as the current CPUs are fast enough to generate it in runtime particularly while reducing the data as much that it is small enough to send it to the GPU when crossing terrain quads.

Firstly, we described a method for storing, updating, and in general managing terrain data. We used pretty common approach which is based on patched terrain concept, that is terrain mesh is divided into smaller parts individually rendered. Then, we described an algorithm for updating quad parameters for a terrain with a variable number of layers. Finally, we have applied an uncommon way for rendering order by using spiral order rendering method which gave us front-to-back rendering without any additional quad parameters.

The concept of generating borders and positions for the biomes is described in the paper. A position, in our assumptions, had to involve parameters like latitude, dispersion, or biome size. The method with applying latitude into terrain type produces more realistic results as it has been demonstrated on Figure 2. This is the main advantage of the proposed method. It has been achieved using the extended Voronoi diagram algorithm with extra parameterization. The method for generating landscape shape for a biome can smoothly blend between other biomes, even if they use different shape generators, and does not fit each other. It has been solved by using the Gaussian blur to smooth transitions between biomes. Moreover, we can use the kernel size of the Gaussian algorithm as an extra parameter to specify the strength of a smoothness. Lastly, we presented and discussed an approach for texturing the biomes. Our approach also easily gives us information to perform the placement of biome-specific stuff like flora or fauna. To be able to test the performance of our engine a special tool was created that shows us our generated terrain in different modes like simple heightmap or normal map.

Combining all the mentioned functionalities gives us a great flexibility to control the look of a terrain and an efficiency for handling all of the terrain data. Firstly, handling terrain by parts gives us the possibility to memorize them in a file and after all this allows us to deform terrain in runtime. Furthermore, we are able to generate the parts of terrain independently, therefore we may use separate thread for computations in order to avoid freezing when moving over terrain. This is also a very scalable method because it is pretty simple to implement the algorithm for updating parameters of the parts with a variable number of layers, thus our terrain may exists in different sizes. Secondly, by having an information about the biome in every vertex, we are actually able to define biome-specific stuff in an every place of the terrain or use it as an extra information with more complex algorithms. Moreover, to achieve the smoothly blended biomes we used the standard Gaussian blur algorithm for blurring which is decoupled as one of the steps necessary for preparing the terrain, therefore we may replace it with a different one to speed up the process (by dropping some quality) or to obtain different visual results. In our approach we also used a non-invasive way to extend the Voronoi diagram (which is just getting data from original algorithm and process it further), so the extended algorithm still contains the basic part which may be modified or extended without any deep knowledge of what we added. Biomes are also affected by other parameters like the distance from ocean, so this would be a great next step to incorporate such feature into our approach.

The proposed engine also contains few inconveniences. The main problem is that although including a new biome is not difficult we have to modify pixel shader in order to include a newly added biome. Another uncomfortable situation is that adaptive mesh cannot be applied for the terrain geometry because the engine is texturing terrain according to vertex information and on flat areas adaptive algorithm may produce only few vertices. Another inconvenience is the important memory usage.

However, the performed tests showed that this approach meets our criteria and it is efficient enough for real time rendering, although demands more memory. We measured time of generating quad of terrain, basic actions that are generally available to the users as well as used techniques like spiral order rendering or using different number of biomes.

## References

[Akenine-Moller, 02] Akenine-Moller, T., Haines, E.: Real-Time Rendering, 2nd ed., AK Peters/CRC Press 2002.

[Berg, 08] de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications, ed., Chapter 7, Springer Berlin Heidelberg 2008.

[Cassol, 11] Cassol, V.J., Marson, F.P., Vendramini, M., Paravisi, M., Bicho, A.L., Jung, C.R., Musse, S.R.: Simulation of autonomous agents using terrain reasoning. In Proc. of the Twelfth IASTED International Conference on Computer Graphics and Imaging (CGIM 2011), IASTED/ACTA Press 2011, 101–108.

[Choroś, 08] Choroś, K., Kaczyński, K.: Time and quality of 3D rendering process using programming code optimisation techniques, International Journal of Intelligent Information and Database Systems 2008, 2(3), 309–319.

[Choroś, 15] Choroś, K., Topolski, J.: A method of the dynamic generation of an infinite terrain in a virtual 3D space, In Proc. of the 7th Asian Conference on Intelligent Information and Database Systems, Part II, LNAI 9012, Springer International Publishing, 377–387.

[Cozzi, 14] Cozzi, P., Ring, K.: 3D Engine Design for Virtual Globes, CRC Press 2011.

[Deussen, 98] Deussen, O., Hanrahan, P., Lintermann, B., Pharr, M., Prusinkiewicz, P., Mech, R.:. Realistic modeling and rendering of plant ecosystems. In Proc. of the 25th Annual Conference on Computer Graphics and Interactive Techniques, ACM 1998, 275–286.

[Dollins, 02] Dollins, S.C.: Modelling for the Plausible Emulation of Large Worlds, Ph.D. thesis, Brown University 2002, http://cs.brown.edu/~scd/world/dollins-thesis.pdf

[Fernando, 04] Fernando, R., Haines, E., Sweeney, T.: GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics. Addison-Wesley Professional 2004.

[Greuter, 03] Greuter, S., Parker, J., Stewart, N., Leach, G.: Real-time procedural generation of 'pseudo infinite' cities. In Proc. of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia, ACM 2003, 87–94, 295.

[Hammes, 01] Hammes, J.: Modeling of ecosystems as a data source for real-time terrain rendering, In: Digital Earth Moving, LNCS 2181, Springer Berlin Heidelberg 2001, 98–111.

[Hendrikx, 13] Hendrikx, M., Meijer, S., Van Der Velden, J., Iosup, A.: Procedural content generation for games: a survey, ACM Transactions on Multimedia Computing, Communications, and Applications 2013, 9(1), 1–22.

[Kopácsi, 13] Kopácsi, S., Kovács, G. L., Nacsa, J.: Some aspects of dynamic 3D representation and control of industrial processes via the Internet, Computers in Industry 2013, 64(9), 1282–1289.

[Li, 15] Li, H., Tuo, X., Liu, Y., Jiang, X.: A parallel algorithm using Perlin noise superposition method for terrain generation based on CUDA architecture. In Proc. of the International Conference on Materials Engineering and Information Technology Applications (MEITA), Atlantis Press 2015, 967–974.

[Lin, 09] Lin, J.C., Wan, W.G., Cui, B., Ding, H.: An algorithm for real-time rendering of very large-scale terrain. Computer Simulation 2009, 11, 059.

[Liu, 11] Liu, C., Fan, L.: On a method of infinite terrain generation in XNA. Journal of Shenyang Aerospace University 2011, 28(2), 12–15 (in Chinese).

[Livny, 09] Livny, Y., Kogan, Z., El-Sana, J.: Seamless patches for GPU-based terrain rendering, The Visual Computer 2009, 25(3), 197–208.

[Losasso. 04] Losasso, F., Hoppe, H.: Geometry clipmaps: terrain rendering using nested regular grids, ACM Transactions on Graphics 2004, 23(3), 769–776.

[Luna, 06] Luna, F.D.: Introduction to 3D Game Programming with DirectX 9.0c: A Shader Approach. Wordware Publishing, Inc. 2006, Chapters 17–18: Terrain Rendering.

[Mali, 11] Mali GPU Application Optimization Guide, ARM 2011, Section 3.2.2.

[Mat, 09] Mat, R.C., Shariff, A.R.M., Mahmud, A.R.: Online 3D terrain visualization: a comparison of three different GIS software. In Proc. of the International Conference on Information Management and Engineering ICIME'09, IEEE 2009, 483–487.

[Minecraft, 14] Minecraft Blueprints: Step by Step Guide for Building Houses & Other Structures, Minecraft Books 2014.

[Nixon, 08] Nixon, M.: Feature Extraction & Image Processing. Elsevier Academic Press 2008.

[Okabe, 09] Okabe, A., Boots, B., Sugihara, K., Chiu, S.N.: Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, Vol. 501. John Wiley & Sons, New York 2009.

[Parberry, 14] Parberry, I.: Designer worlds: procedural generation of infinite terrain from real-world elevation data. Journal of Computer Graphics Techniques 2014, 3(1), 74–85.

[Perlin, 02] Perlin, K.: Improving noise. ACM Transactions on Graphics 2002, 21(3), 681–682.

[Raffe, 12] Raffe, W.L., Zambetta, F., Li, X.: A survey of procedural terrain generation techniques using evolutionary algorithms, In Proc. of the IEEE Congress on Evolutionary Computation CEC 2012, 1–8.

[Reed, 10] Reed, A.: Learning XNA 4.0: Game Development for the PC, Xbox 360, and Windows Phone 7. O'Reilly Media, Inc. 2010.

[Ruzinoor, 12] Ruzinoor, C.M., Shariff, A.R.M., Pradhan, B., Rodzi Ahmad, M., Rahim, M.S.M.: A review on 3D terrain visualization of GIS data: techniques and software. Geo-spatial Information Science 2012, 15(2), 105–115.

[Schneider, 06] Schneider, J., Boldte, T., Westermann, R.: Real-time editing, synthesis, and rendering of infinite landscapes on GPUs, In Proc. of the 11th International Workshop on Vision, Modeling, and Visualization, IOS Press 2006, 145–152.

[Shiau, 07] Shiau, Y.H., Liang, S.J.: Real-time network virtual military simulation system. In Proc. of the 11th International Conference on Information Visualization (IV'07), IEEE 2007, 807–812.

[Sinclair, 10] Sinclair, D.: S-hull: a fast sweep-hull routine for Delaunay triangulation 2010, http://www.s-hull.org (accessed July 31, 2014).

[Smelik, 09] Smelik, R.M., De Kraker, K.J., Tutenel, T., Bidarra, R., Groenewegen, S.A.: A survey of procedural methods for terrain modelling. In Proc. of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS), 2009, 25–34.

[Wei, 13] Wei, W., Dongsheng, L., Chun, L.: Fixed-wing aircraft interactive flight simulation and training system based on XNA. In Proc. of the International Conference on Virtual Reality and Visualization (ICVRV), IEEE 2013, 191–198.

[Wolf, 01] Wolf, M.J.P. (ed.): The Medium of the Video Game, University of Texas Press 2001.

[Yannakakis, 15] Yannakakis, G.N., Togelius, J.: A panorama of artificial and computational intelligence in games. IEEE Transactions on Computational Intelligence and AI in Games 2015, 7(4), 317–335.