APPLYING PERLIN NOISE ON 3D HEXAGONAL TILED MAPS

by
Fatih Gürler

Submitted to Graduate School of Natural and Applied Sciences
in Partial Fulfillment of the Requirements
for the Degree of Master of Science in
Computer Engineering

Yeditepe University
2023

APPLYING PERLIN NOISE ON 3D HEXAGONAL TILED MAPS

APPROVED BY:

Assist. Prof. Dr. Esin Onbaşıoğlu ……
(Thesis Supervisor)
(Yeditepe University)

Assoc. Prof. Dr. Dionysis Goularas ……
(Yeditepe University)

Prof. Dr. Cem Ünsalan ……
(Marmara University)

DATE OF APPROVAL: ..../..../20..

I hereby declare that this thesis is my work and that all information in this thesis has been obtained and presented by academic rules and ethical conduct. I have fully cited and referenced all material and results as required by these rules and conduct, and this thesis study does not contain any plagiarism. If any material used in the thesis requires copyright, the necessary permissions have been obtained. No material from this thesis has been used for the award of another degree.

I accept all kinds of legal liability that may arise in cases contrary to these situations.

Name, Last Name Fatih Gürler .

Signature ……..……..

ACKNOWLEDGEMENTS

It is with immense gratitude that I acknowledge the support and help of my Assistant Prof. Dr. Esin Onbaşıoğlu. Pursuing my thesis under her supervision has been an experience that broadens the mind and presents an unlimited source of learning.

Finally, I would like to thank my mother, my wife and my son for their endless love and support, making everything more beautiful.

ABSTRACT

APPLYING PERLIN NOISE ON 3D HEXAGONAL TILED MAPS

Procedural Content Generation gained importance together with the rapid increase in the video computer game industry. Procedural terrain generation, which is a sub-category of procedural content generation, is used to generate terrains for games. However, using terrain generation in strategy games is inadequate, as it does not support tiling. This study aims to generate 3D hexagonal tiled terrains using procedural terrain generation methods with Perlin noise. This method can also be used as a filter algorithm to tile terrains that have already been generated.

ÖZET

3B ALTIGEN KARO HARİTALARDA PERLİN GÜRÜLTÜSÜ UYGULAMASI

Video bilgisayar oyun endüstrisindeki hızlı artışla birlikte prosedürel içerik üretimi önem kazanmıştır. Prosedürel içerik oluşturmanın bir alt kategorisi olan prosedürel arazi üretimi, oyunlar için araziler oluşturmak için kullanılır. Ancak, karo yapıyı desteklemediği için strateji oyunlarında prosedürel arazi oluşturma kullanılamaz. Bu çalışma, Perlin gürültüsü ile prosedürel arazi oluşturma yöntemlerini kullanarak 3B altıgen karo tabanlı araziler üretmeyi amaçlamaktadır. Bu yöntem, hâlihazırda oluşturulmuş olan haritaları karo yapıya çevirmek için bir filtre algoritması olarak da kullanılabilir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF SYMBOLS/ABBREVIATIONS

PCG	Procedural content generation
PTG	Procedural terrain generation
ER	Easily recognizable
FPT	Fast process time
TS	Terrain smoothness
LI	Linear Interpolation
PN	Perlin noise
RTS	Real-time strategy
RPG	Role-playing game
TBS	Tile-based strategy
CPU	Central processing unit
GPU	Graphics processing unit
ms	Millisecond
s	Second

1. INTRODUCTION

As game-development industry grows, procedural content generation (PCG) methods are more commonly used. When preparing a map or terrain, it takes a lot of time to draw by hand. This makes PCG more efficient as PCG is much faster. As PCG is a procedural method, it is the subject of computer engineering. One of the main aims of engineering is to reduce production time with automation applications. If this principle is applied to the game industry, one of the basic aim of an engineer must be to decrease the game-development time. This can be achieved by using PCG. Engineering speeds up processes in industry, likewise procedural content generation speeds up game generation process.

There can be a lot of PCG methods for different parts of a game. Such as game flow, character movement, and most importantly terrain generation. The content of this thesis is procedural terrain generation (PTG).

PTG is commonly used for real-time strategy (RTS) games, role-playing games (RPG), action, and adventure games where terrains are generated as a mesh where characters and buildings stand. PTG can create such terrains very fast. But existing PTG methods do not support tiled maps.

Tiled maps are commonly used in turn-based strategy games and simulation games. Tiled maps consist of tiles. Tiles are generated separately and united later. That makes tiled map generation, expected to be a longer process. Although a grid can be put on a terrain generated with PTG, tiles are not uniform and are not easily recognized (ER). In this thesis, a procedure for PTG to support tiled maps is developed.

1.1. OUR APPROACH

In our approach, mainly PTG algorithms are used and some additional algorithms are developed to support tiling where each tile should be easily recognized. So PTG (a better way) can be used for tiled-based strategy games (TBS). Hexagonal tiling is used instead of quad tiling.

Speed optimization is preferred instead of memory optimization in order to succeed fast process time. PTG has very soft discrete curves. The softness (on the contrary sharpness is defined in the following chapters) of PTG is tried to be preserved. PTG generates real 3D terrains where each vertex on the mesh have different height values depending on a 2D noise function.

1.2. DEFINITION OF PROBLEMS

1.2.1. First Problem

When grid is applied to maps generated with PTG, grid does not fit to terrain layers**.** A basic map is generated by using PTG and it is presented in Figure 1. Burdzieg defines procedures for modelling real world maps with hexagonal grids [1]. A real-world map is taken and grid is placed on the map. Map regions corresponding to tiles are converted to predefined tiles for tiling support. Level of detail is analyzed for this tile conversion method. As level of detail increases, process time and memory requirements increase.

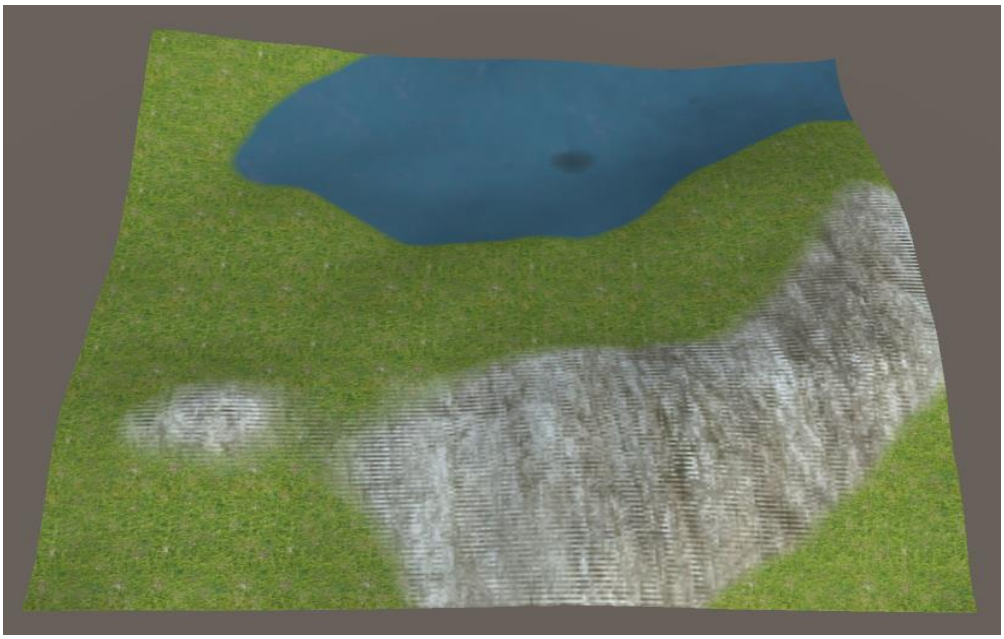


Figure 1. A basic map with PTG

The grid can be directly placed on the generated map as in Figure 2. But it is unclear that the marked tile in Figure 2 is land or sea. Because half of the marked tile sea and half of it is land.

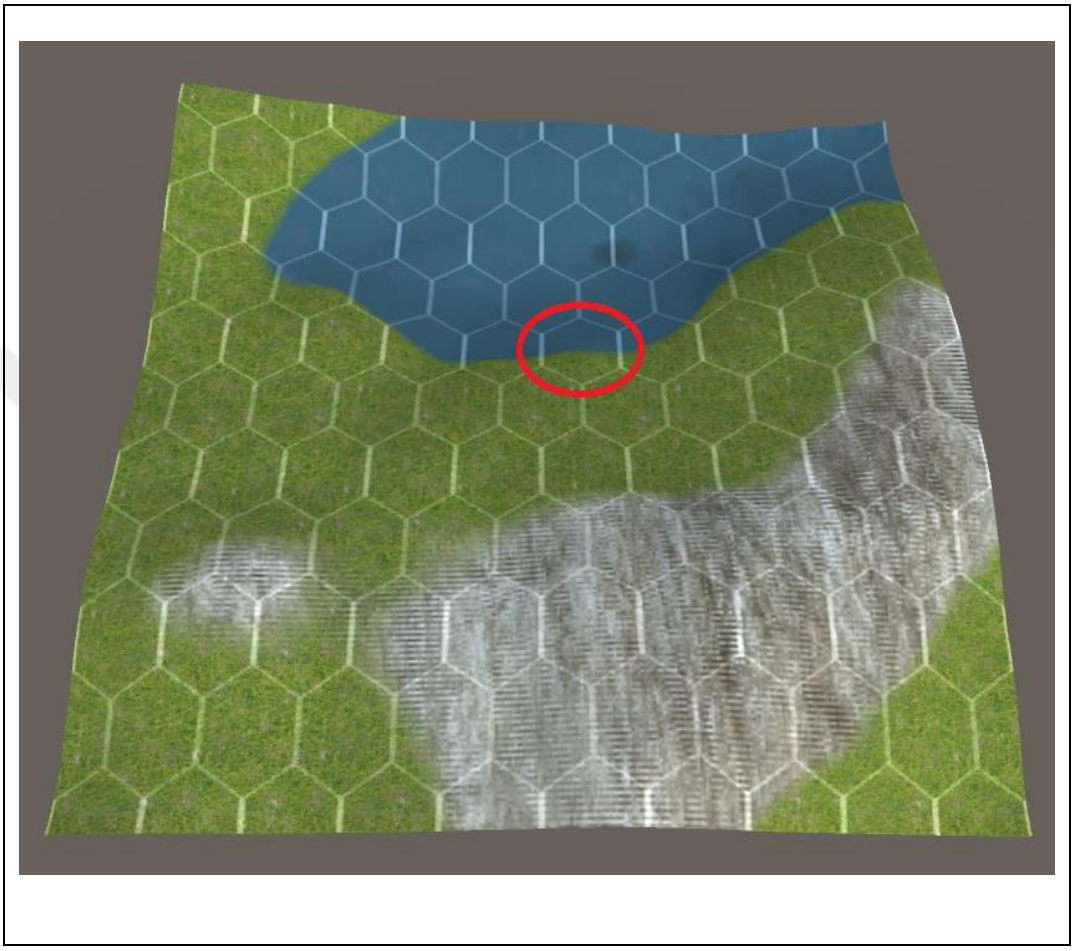


Figure 2. A basic map with hexagonal tiling

1.2.2. Second Problem

Quad form of height maps is not compatible with hexagonal grid. For generating 3D maps; gray-scale real world maps which is presented in Figure 3 or 2D noise-based height maps which is presented in Figure 11, are used. In both cases 2D height maps consist of pixels. Pixel-based height maps use cartesian coordinates. If pixels are converted to tiles, the map consisting of generated tiles, should be in cartesian coordinates. But our map should consist of hexagonal tiles. So, pixels should be generated in a hexagonal shape or cartesian pixels should be converted to hexagonal shape.

Figure 3. Low resolution Mediterian map

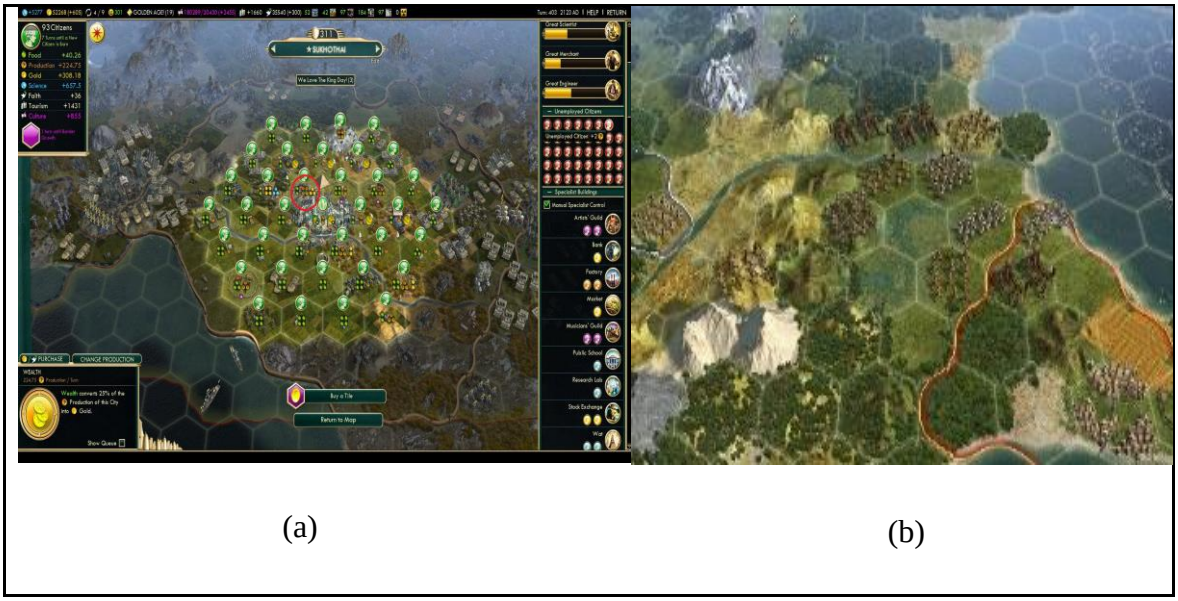
2. TILE-BASED STRATEGY GAMES

In tile-based strategy games (TBS), maps consist of tiles and agents go from one tile to another tile. Movements of agents are defined on tiles. Although there are a lot of different tile shapes, square and hexagon tile shapes are most commonly used. Before the last decade square tiles were used but they were started to be used in the last decade. Maps are used instead of terrains in TBS. Therefore, third person camera is used instead of first-person camera. Graphics in TBS are not as good as action or RTS games. High-level shader graphics are not generally used.

2.1. CIVILIZATION 5

Civilization5 [2] is published in 2010. Civilization6 is published in 2016 and has better textures and resolutions but with the same map form.

Independent of tile movement, an economy is defined by tile types. Different tile types can have different production parameters. For example, in Figure 4a, the selected tile has 2 food, 4 production, and 3 economy capability. There are different values on different tiles. The terrain economy is one of the most important economic parameters in the Civilization 5 game.



(a) (b)

Figure 4. Civilization 5 (a) Civilization 5 city view, (b) Civilization map view

In civilization 5 as seen in Figure 4b, although the map seems to be 3D, it is defined as 2D. There are prepared meshes for different tiles types. They do not have different heights. Mountain meshes are not selected according to height level but instead selected randomly or according to neighboring tiles. A pre-defined mesh is used for certain height type. Total mesh is formed by joining pre-defined tile meshes. Tile meshes are 3D but there must be varying vertices according to map shape in a 3D map mesh. Varying vertices are discussed in the following chapters.

There are prepared meshes for mountains, grass, and dry lands. There is more than one mesh for each terrain type to increase map variety. The prepared meshes for tile types prevent civilization maps from appearing like real-world maps.

2.2. ANNO SERIES

Anno 1404 [3] is published in 2009 and Anno 1800 is published in 2019. They are in Figure 5. This game is 2D but uses 3D mountain and tree objects for 3D appearance.

(a) (b)

Figure 5. Anno Series (a) Anno1404, (b) Anno 1800

2.3. ENDLESS LEGEND

Endless Legend [4] has different height definitions. Being on the high ground is a tactical parameter in the game. There is one height definition for each tile. Tiles support ER. There is a tile economy in this game but with different parameters from civilization 5. Tile blending

is very sharp. It is published in 2014. It can be observed in Figure 6 that game has sharp edges in neighbor tiles.

Figure 6. Endless Legend

2.4. THEA 2: THE SHATTERING

In Thea 2 [5] 3D appearance is more realistic than previous games but the hexagonal grid does fit with tiles. It is published in 2018. It can be observed in Figure 7 that sharpness problem (mentioned in Section 2.3) is solved.

Figure 7. Thea 2

3. PROCEDURAL TERRAIN GENERATION

All procedural processes in game design are called procedural content generation (PCG). PTG is one of these fundamental procedural processes in PCG.

3.1. PROCEDURAL CONTENT GENERATION

In procedural content generation (PCG) [6], the contents of games are created automatically. Different stages or parts of games can be generated procedurally for games. It contains a spectrum of game contents [7] using different methods. Game space generation is one of the most important topics of procedural generation. The whole game world can be modelled as landscapes, road networks, buildings, living beings and stories, where landscapes refer to terrains [8]. In this thesis, terrain generation is discussed.

3.2. TERRAIN/MAP GENERATION

Terrains and maps differ in some ways. Terrains are commonly used in RTS, RPG, action, and adventure games. Maps are commonly used in TBS games. The differences between a map and a terrain can be observed in Figure 8 and listed in Table 1.

Table 1. Terrain vs Map

Terrain	**Map**
Near Sight	Far Sight
Generally, 1st person camera	Generally, 3rd person camera
Can be a region (ex: lakeside)	Can be a country or a world
Visual quality: must be good	Visual quality: not so good
No tiling support	Can be **TILED**

(a) (b)

Figure 8. Games (a) Pubg, RPG game, (b) Civ 5 TBS game

3.3. PROCEDURAL TERRAIN GENERATION METHOD

In PTG, the complete mesh is generated once in runtime. It has a reasonable process time until recursive erosion algorithms (such as water and thermal weathering) are used. It can be observed that most of the TBS games loads very slow, while both starting and loading. Even

maps of TBS games are created slowly when maps are created by related map editors. Although games use most of PCG processes, in map editors only map generation process is active. PTG is so fast that new regions of a terrain can be generated in run-time. So seamless maps can be generated with PTG. PTG also increases the playability of a game because different maps can be created by just changing noise parameters. This method is commonly used in role-playing games (RPG) and action games (Figure 8a). But it has not been used in TBS games (Figure 8b) as it does not support the generation of tiled maps.

3.4. SEAMLESS TERRAIN GENERATION

PTG is so fast that chunks can be created in run time while an agent passes from one chunk to another one. So, a seamless terrain can be generated in run-time. In a chunk-based approach, when the agent gets close to the edge of the terrain new tiles are generated along the edge.

Very large terrains are generated by joining chunks. In other words, very large terrains are divided into chunks. There is a vertex limit for any mesh, therefore large meshes must be divided into smaller meshes. Figure 9 is a terrain consisting of 9*9 chunks. The squares represent different terrains.

The little black circle is our agent where the camera is focusing. It is in the red chunk. So, terrain of the red chunk must exist. It can go to yellow, orange or brown chunks. So yellow, orange and brown chunks must also exist. There is no need to create green and white chunks as our agent cannot go to those chunks. The gamer just sees what the camera shows and there is no need to generate and render the chunks that the camera is not showing.

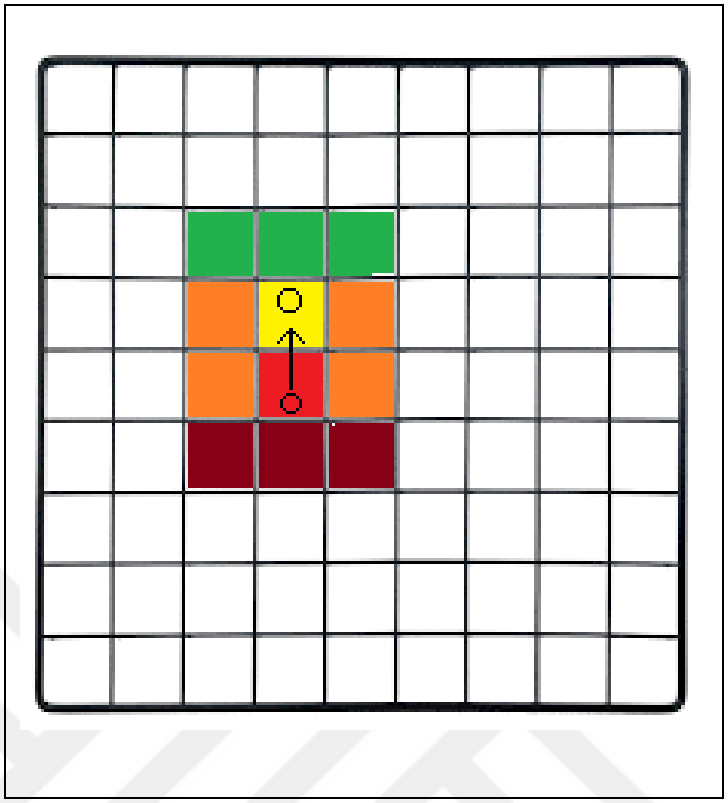


Figure 9. Chunks

If an agent goes to the edges of the red chunk and enters into any neighboring chunk, before it enters, the possible chunks must be created before the entrance.

Agent moves from red chunk to yellow chunk. After it enters to yellow chunk, green chunks must be created as the agent can go to the green chunk now. Brown chunks can be deleted to preserve memory. By using this method seamless worlds can be created in runtime.

3.5. RELATED WORK

Procedural terrain generation (PTG) consists of 5 parts; height map, water parts, vegetation, road network, and urban environment generation [9]. Height-map forms the main mesh where all objects of the game are placed. The methods that can be used to generate height maps include Diamond square iteration [10] and Voronoi diagrams [11]. The most common way of PTG is to use 2D procedural noise functions [12] such as Perlin noise [13] which generates noise by sampling and interpolating points in a grid of random vectors. Rescaling and adding several levels of noise to each point in the height-map results in natural mountain-

like terrains. The appearance of a terrain can be improved and can be more realistic by using hydraulic erosion and thermal weathering [14].

3.6. MESH GENERATION

1D noise signals consist of 2 values; input and output. There is a y value for each x value in the function (Figure 10).

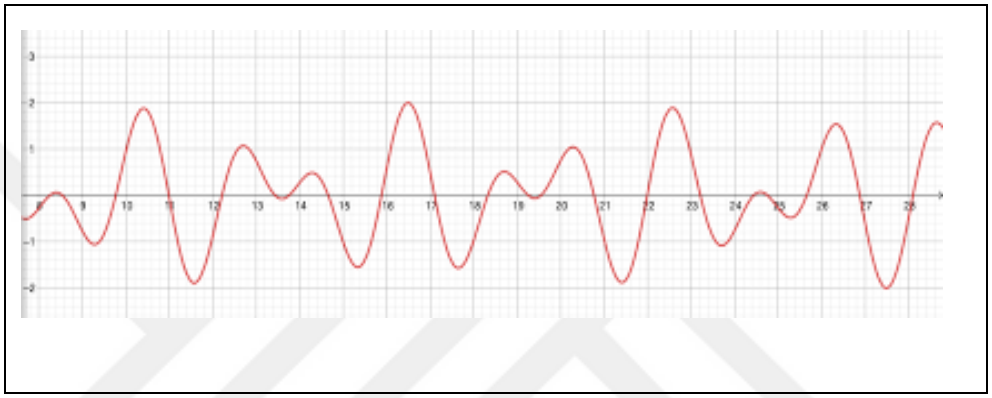


Figure 10. 1D noise signal

Noise signals can also be 2D. 2D noise signals consist of 3 values; 2 inputs and 1 output. There is a y value for each x and z combination (Figure 11). Three values can define a vertex in space. So, the mesh vertices can be presented as a 2D signal. This type of textures, as in Figure 8, are also called height maps.

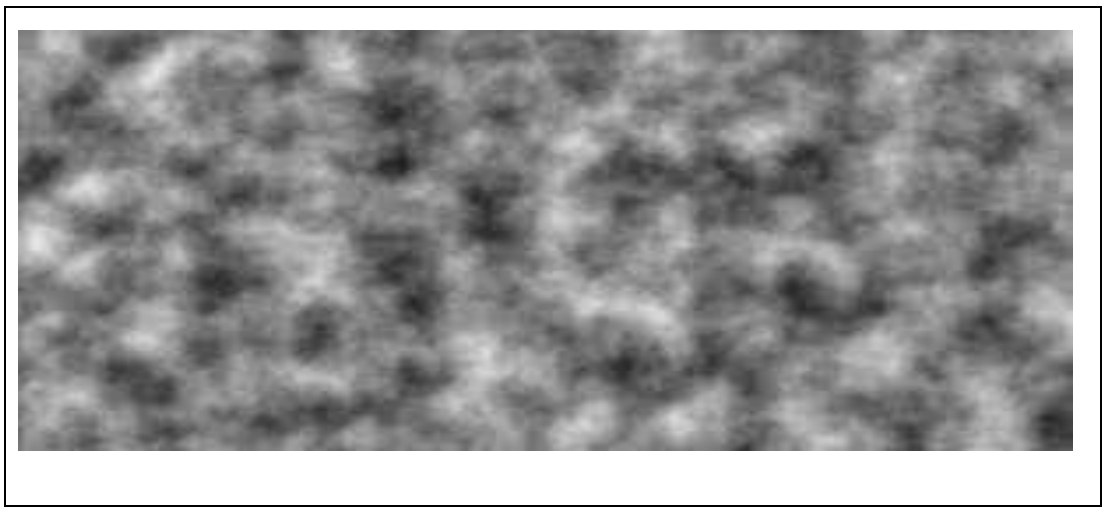


Figure 11. Height map of 2D noise signal

In Figure 11 each pixel can be converted to a vertex. x and z coordinates of the pixel are the same as the x and z coordinates of the vertex. The color of the pixel can be converted to the height of a vertex. In this way 2D array of vertices can be formed in cartesian coordinates.

In Figure 12, there are 4x4=16 vertices numbered in blue. Each square contains 2 triangles. There are 9 squares. There are 18 triangles numbered in red. Triangle 1 is formed by using vertices 1, 5, 6 and triangle 2 is formed by using vertices 1, 2, 6. After triangulating all triple vertices to form quad tiles, mesh in Figure 13 is formed. Number of squares can be calculated with Equation 3.1.

$$Number\ of\ squares = (x - 1) * (z - 1) \tag{3.1}$$

x in the number of vertices in x dimension and z is the number of vertices in z dimension.

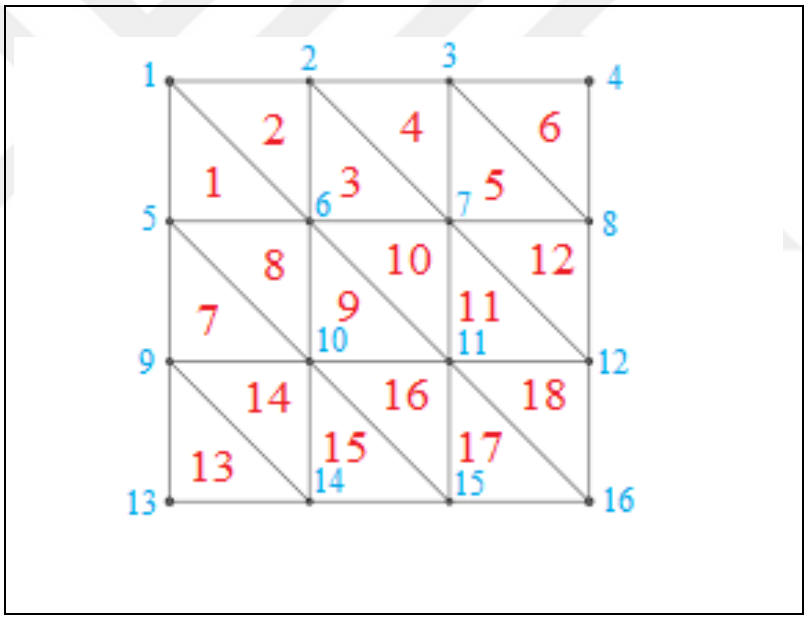


Figure 12. Triangle and vertex numbers

In Figure 13, squares of a mesh are presented. All vertices of a square have different height values. All squares in Figure 13 consist of 2 triangles as shown in Figure 12. All tiles have different shapes (meshes).

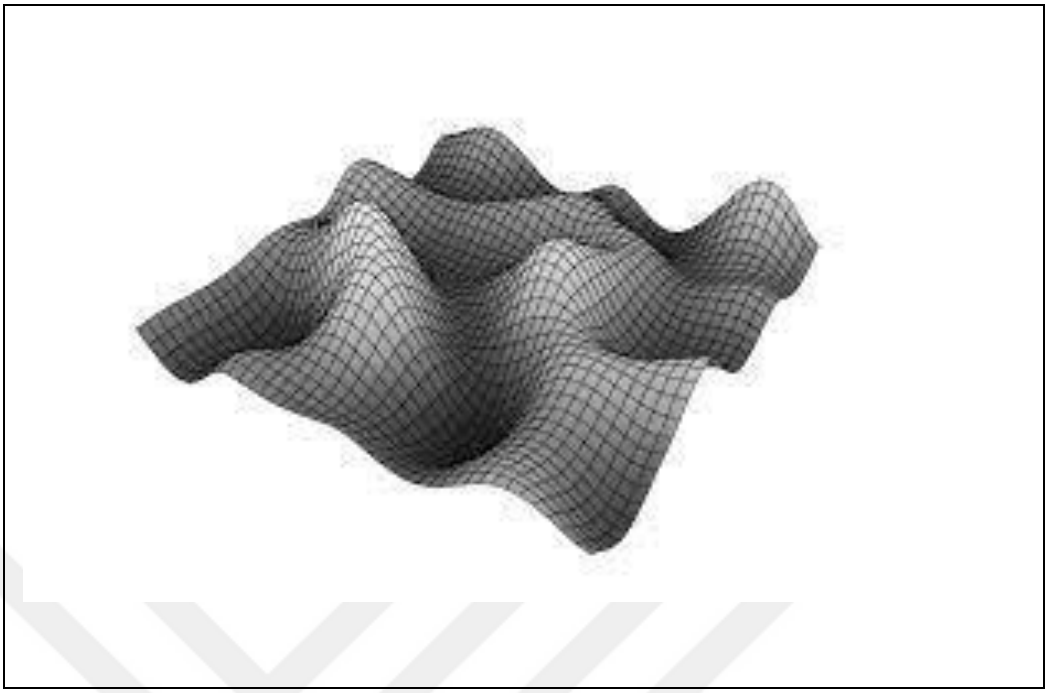


Figure 13. 3D terrain mesh

3.7. TEXTURING

Texturing is a process of adding textures to game objects. After colourless mesh is generated, the mesh should be painted with some textures. The mesh is painted according to the height of vertices. In my application, 3 textures are used to paint our mesh. Sea texture, land texture and mountain textures are used. Figure 14a is a non-textured mesh and Figure 14b is the textured mesh.

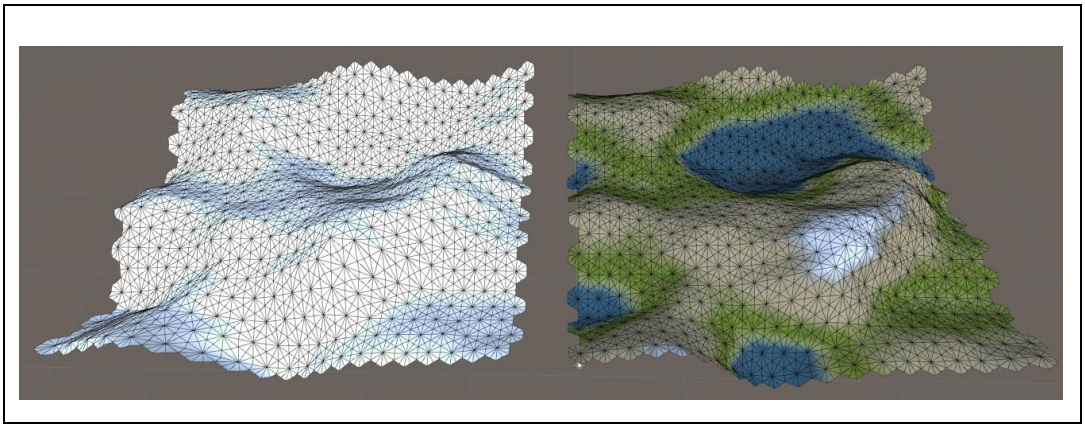


Figure 14. Mesh (a) Hex mesh, (b) Shader applied to hex mesh

3.8. TILED MAPS

In tiled games (Figures 1-2-3-4), maps consist of individual tiles [16]. Each tile may contain a predefined texture. These predefined tiles are called tile types. A tileset consists of different tile types. By placing different tiles side by side, a tiled map is generated. Even tile selection can be done by a procedural noise function [12], all tiles have separate meshes. Then neighbouring tiles must be blended [17] for a better total view. This process is expected to be slower than PTG. This is discussed in Section 7.1.4.

In Figure 15 tiles are defined as different colours with black borders. In Figure 15a all tiles are generated separately. In Figure 15b tiles are joined to make the map. As all borders are black there is no need for blending.

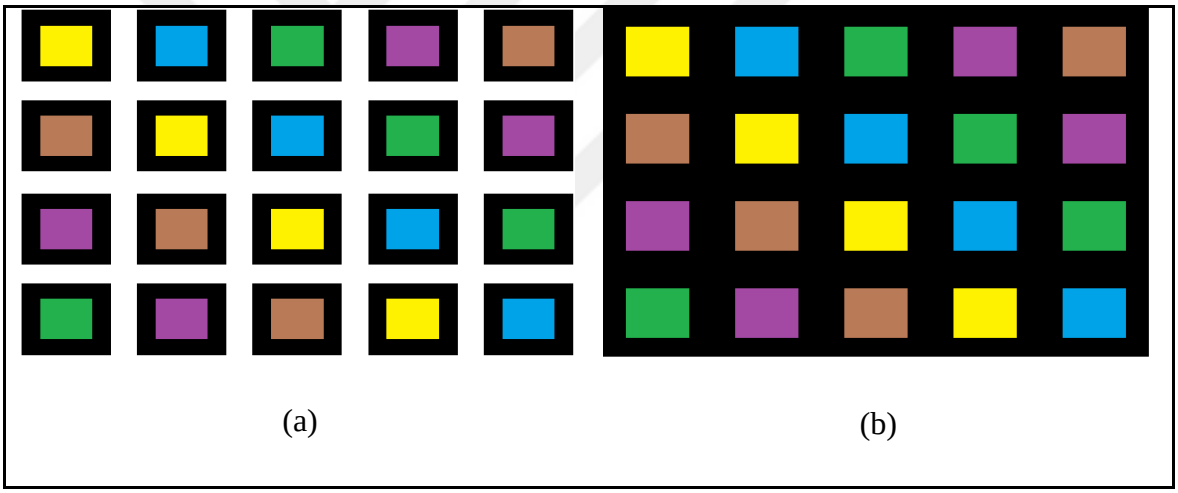


Figure 15. Map tiling (a) Separate tiles, (b) Joined tiles

3.8.1. Related Work

A tile-based game uses tiles as fundamental elements of the map. In general, the Domino game is accepted as one of the first tiled games. In the Domino game, the following man's starting number must be same with the previous man ending number. Domino game uses 1d blending. Wang tiles [18] are similar to the Domino game. The edge of neighbouring tiles must have the same colours in both the x and y directions. Wang tiles game is the first form of tile blending. In Figure 16, tiles consist of squares having different colours at the top, bottom, left and right sides. Neighbour tiles have the same colour for tile blending.

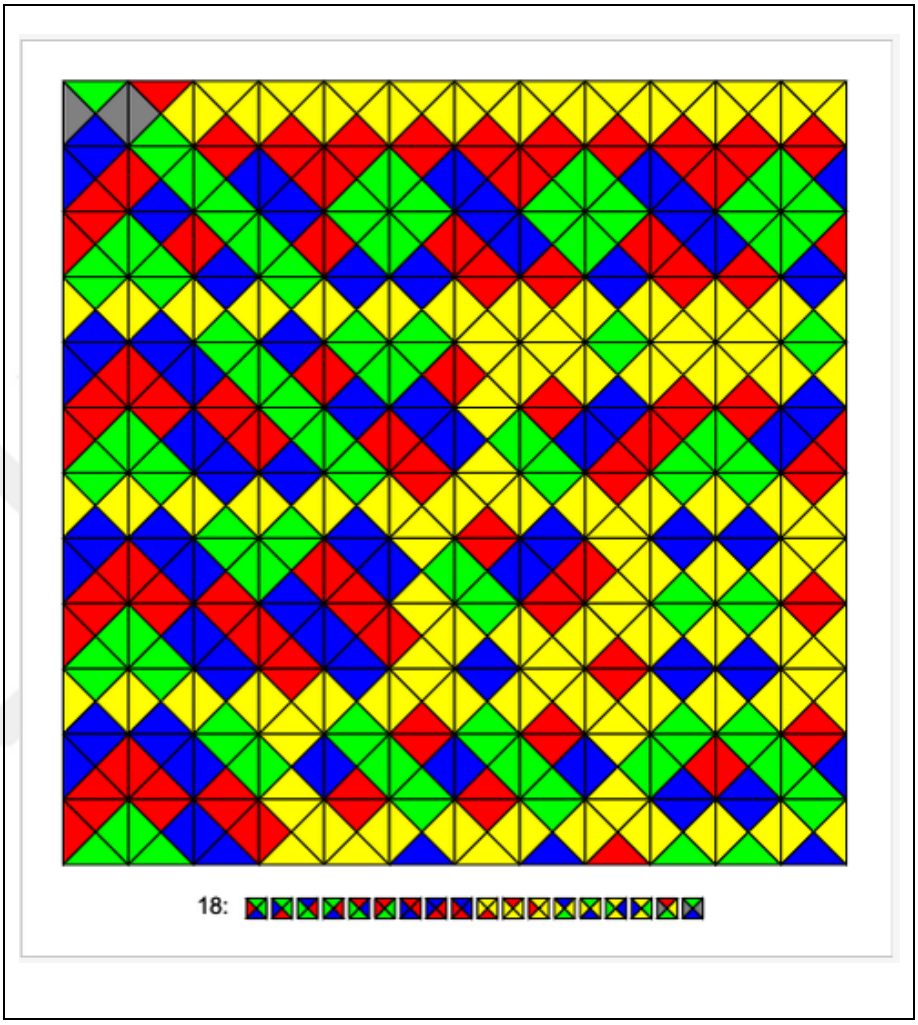


Figure 16. Wang Tiles

Maung generated procedural textures by using tiling with noise. Tiles are generated from gradients which are stored as integer lattice points [19]. Maung developed a tile base framework for generating procedural terrain for video games. There is survey about tiling in his work [20].

Dominik Scholz prepared a bachelor thesis about generating a map from tiles [21]. “Tile matcher” method is used. The basic idea of tile matcher is same with wang tiles. Colours in wang tiles can be assumed as tile types. Each quad has 4 sides. If intersecting sides of quads

are same (same colour in Wang tiles, same terrain type in the reference thesis), these tiles can match.

Figure 17 shows the tile set used in the reference thesis. Each tile can be rotated 90º 3 times. So, there are 33x4=132 tile types. Colours are used for tile types. Green dot is used for grass, dark blue is used for hill, brown is used for sea and purple is used for river. Using these tile types, map in Figure 18 is created by using tile matcher method.

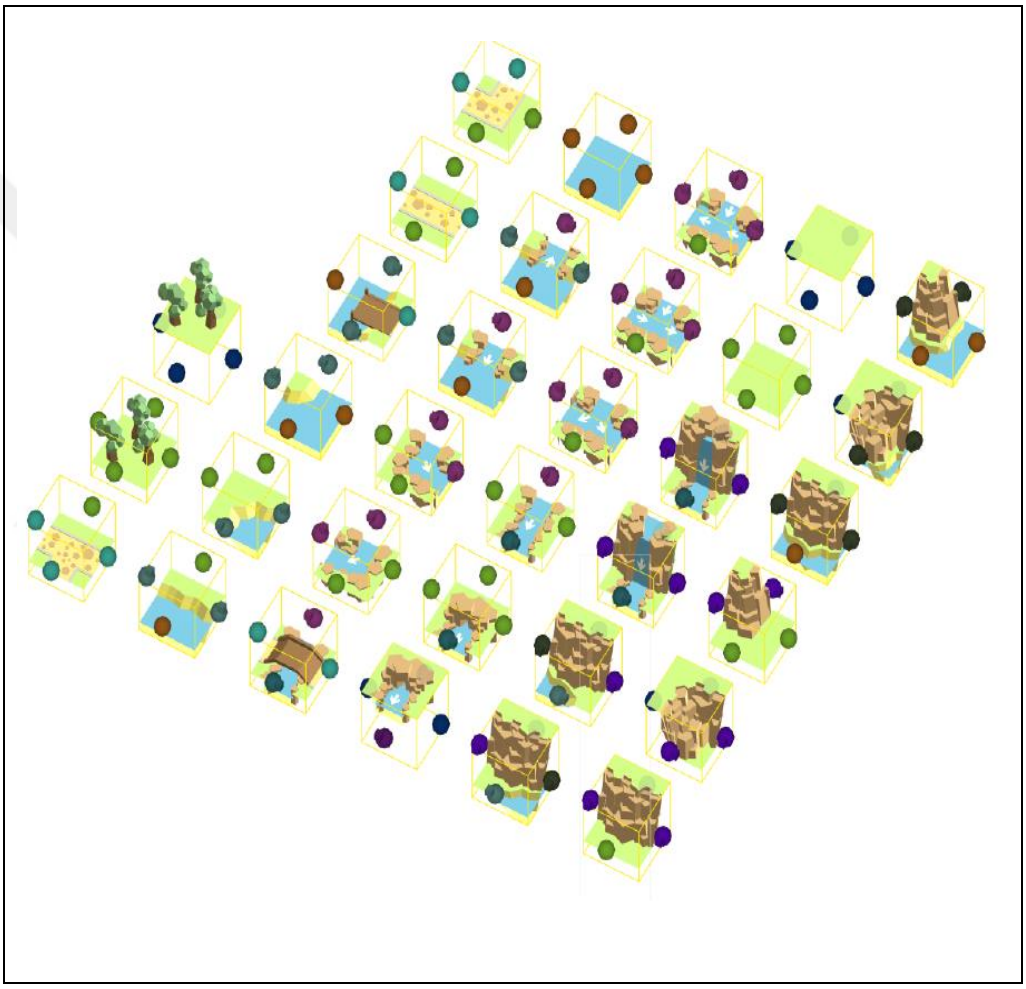


Figure 17. Tiles used in the thesis [21]

Predefined meshes are used for tile types. Although the resulting map is 3D, as it uses pre-defined meshes, it is not realistic. In a realistic world map, each vertex should have different heights. Blending is succeeded by using matching tiles.

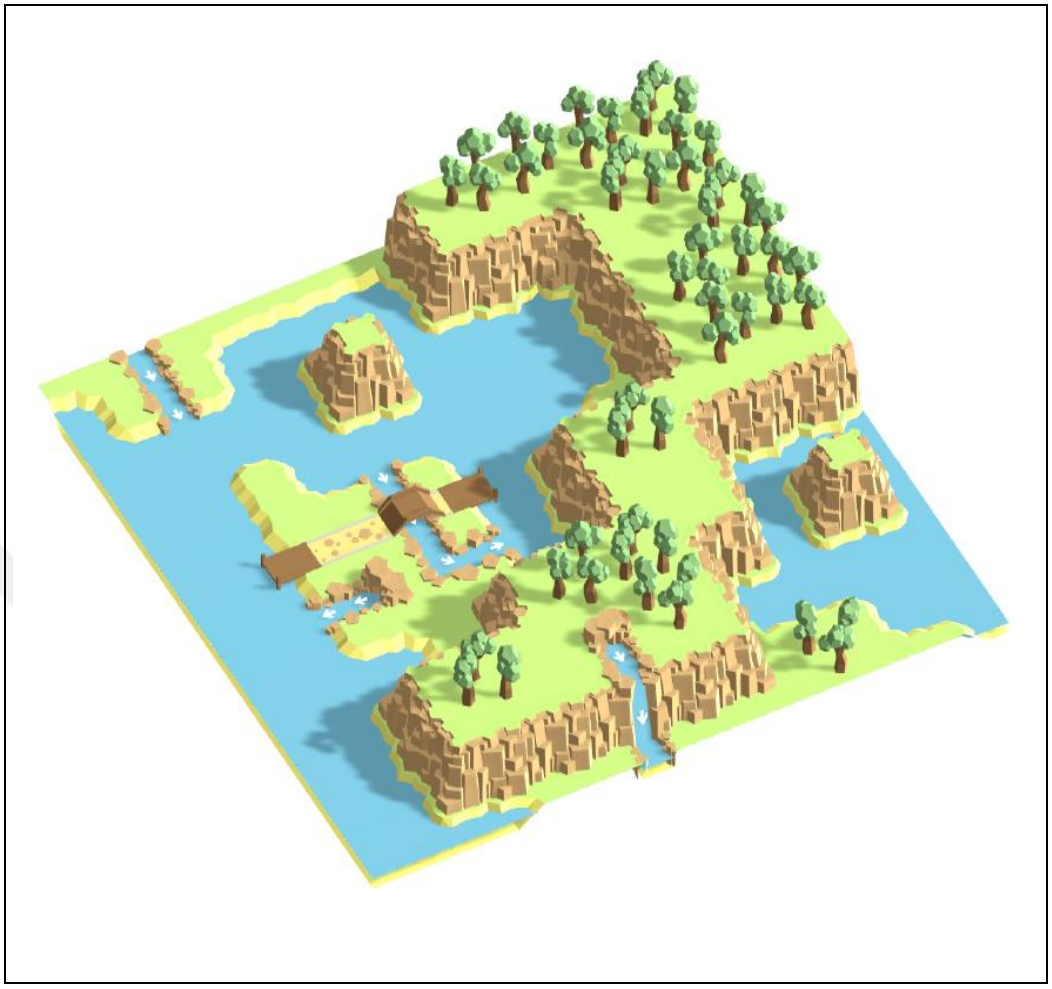


Figure 18. Map generated in the thesis [21]

4. OUR APPROACH

The target of this thesis is to speed up the map generation time and to avoid the pattern repetition problem in turn-based strategy maps. Tiling support for PTG has been developed in order to solve this problem.

4.1. REQUIREMENTS

Our approach has five requirements: mesh is 3D, hexagonal tiles should be generated, the tiles should be easily recognizable, the terrain must look smooth and it should have fast processing time.

4.1.1. 3D Mesh

If realistic terrain objects are modelled such as mountains, hills, etc., vertices of the model should have different heights. Vertices with different heights make a terrain object more realistic.

Another 3D modelling is done by using pre-defined 3D tile meshes. 3D meshes are prepared before map generation. If a tile should be mountain, a predefined mountain mesh is used for that tile. Map is prepared by joining these pre-defined tile meshes. But this process makes the map less realistic. To prepare a realistic map, 2D noise signal can be converted 3D mesh.

4.1.2. Hexagonal Tile

Tiles should have hexagonal shape. Before the last decade, almost all tiled maps in video games were formed of quads. In the last decade hexagonal tiled maps started to be used. In quad tiled maps, characters can move in 8 directions; 4 of them are diagonal and 4 of them are horizontal or vertical. The costs of horizontal and vertical movements are different than the cost of diagonal movement. However, in **hexagonal tiled** maps all movements have equal movement cost. So, path finding in hexagonal tiled maps is more efficient than quad tiled maps [22]. For this reason, hexagonal tiles are considered in this thesis.

4.1.3. Easily Recognizable

In a tile-based game, characters move from one tile to another. Therefore, tiles must be distinct and **easily recognizable (ER)**. In Figure 19, the pointed tile is a land tile but substantial part of it is sea. The pointed tile's type is not easily recognizable.

Figure 19. Thea 2, strategy game

In order to make the tile easily recognizable, all vertices except border vertices must belong to the same terrain type. Border vertices can be changed to support blending between neighbouring tiles.

4.1.4. Fast Processing Time

For **fast processing time,** the mesh must be generated as a whole and must be textured by a single shader.

4.1.5. Terrain Smoothness

Mesh in the Figure 13 has a soft appearance. Peaks increases slowly and dips decreases slowly. This slow increase creates the softness of terrain. On the contrary, sharpness is measured with curvature. In fact, smoothness is a not requirement, it gives an idea about map. This is not an acceptance criterion. Because there are very sharp terrains in nature; such as cliffs, waterfalls, sharp mountains(ridges).

4.1.5.1. Curvature

Curvature determines the curve's **sharpness** (or on the contrary smoothness). The curvature of a curve at any point is measured with the second derivative of that curve at that point. In a discrete system (as vertices are discrete), at least 3 points are needed to measure the curvature. The curvature is measured using the edges of the neighbouring triangles. If a_1, a_2, and a_3 are height values of three vertices on these edges where the second one is the common vertex, the curvature at the second vertex can be defined in equation (4.1).

$$Curvature = |(a_3 - a_2) - (a_2 - a_1)| = |2 * a_2 - a_1 - a_3| \qquad (4.1)$$

a1, a2, a3 vertices is presented in Figure 20. Figure 20a has a smooth curvature, but Figure 20b has a sharp curvature. If difference between peak vertex and vertices just near the peak vertex is small, map is smoother. Because a3-a2 and a2-a1 has opposite signs at peaks. Their difference is added for curvature calculation.

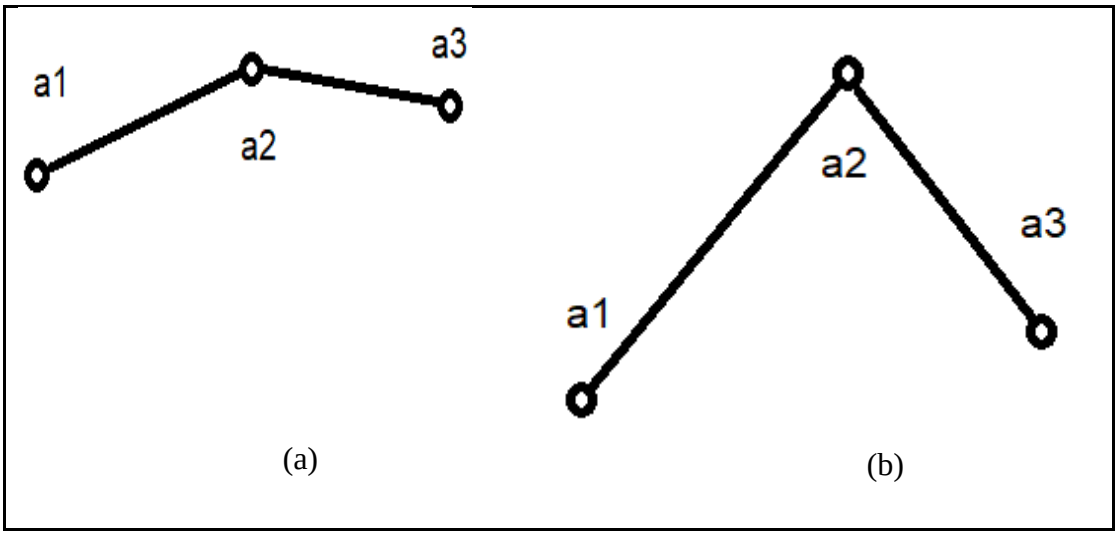


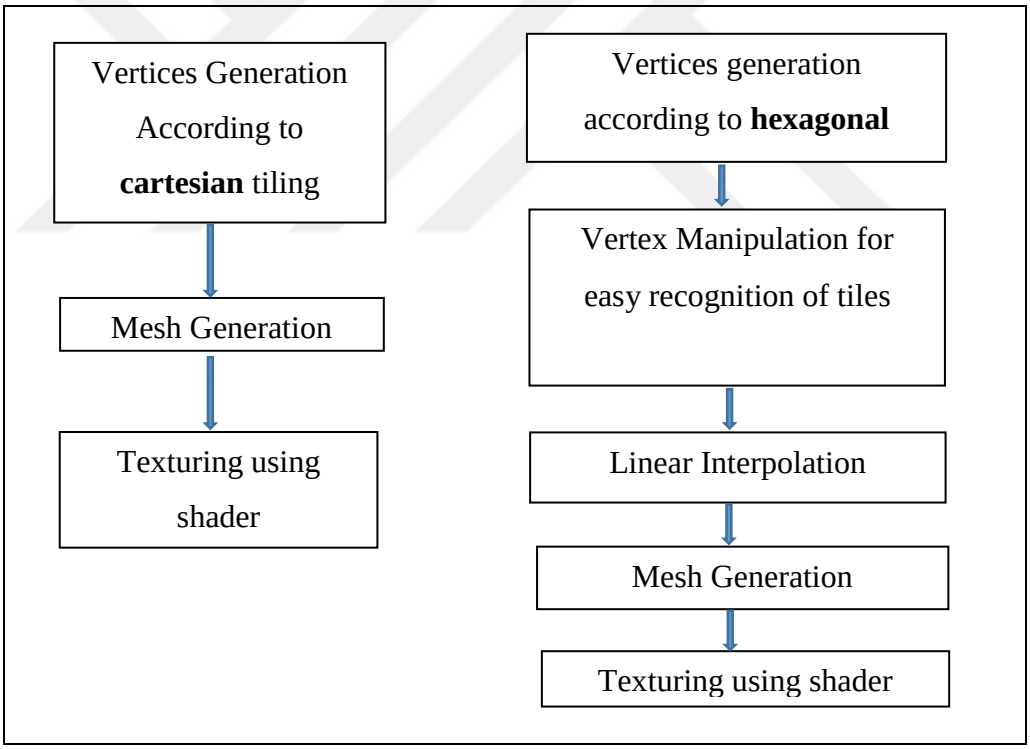
Figure 20. discrete curves (a) softer (b) sharper

5. THEORY

Perlin noise is used to generate a 2D noise signal. Basic PTG and our approach are compared in Algorithm 1. After converting Cartesian tiling to hexagonal tiling, two algorithms; vertex manipulation for easy recognition and linear interpolation algorithms; are used in our approach.

Basic PTG consist of vertices generation, mesh generation and texturing using shader. Vertex manipulation and linear interpolation is added after vertices generation in our approach. Vertices are generated according to hexagonal tiling instead of cartesian tiling in our approach.

Algorithm 1. Basic PTG and our approach



5.1. PERLIN NOISE

Perlin noise generates coherent noise over a space. Any two points in the space, the value of the noise function changes smoothly as you move from one point to the other. There are no discontinuities. Perlin noise is pseudo-random noise function that fills all three-dimensional space. But it can be also considered 2D.

Perlin Noise can be used to create curvy smooth terrains as it has a fading function in the last step [23]. Although Perlin noise can be defined as 3D, it will be used as 2D in our application. 3D vertices are defined with 2D coordinates (x, z) and height value (y) is generated from noise function. Noise function is presented in Equation 5.1. *x*, *y*, *z* values in equation 5.1 determine a vertex in the space.

$$noise2D(x, z) = y \tag{5.1}$$

Blue vectors are pseudo-random gradient vectors in Figure 21. These vectors create the randomness of noise. Figure 21 is a part of an infinite grid. Corners of grid consist of whole numbers.

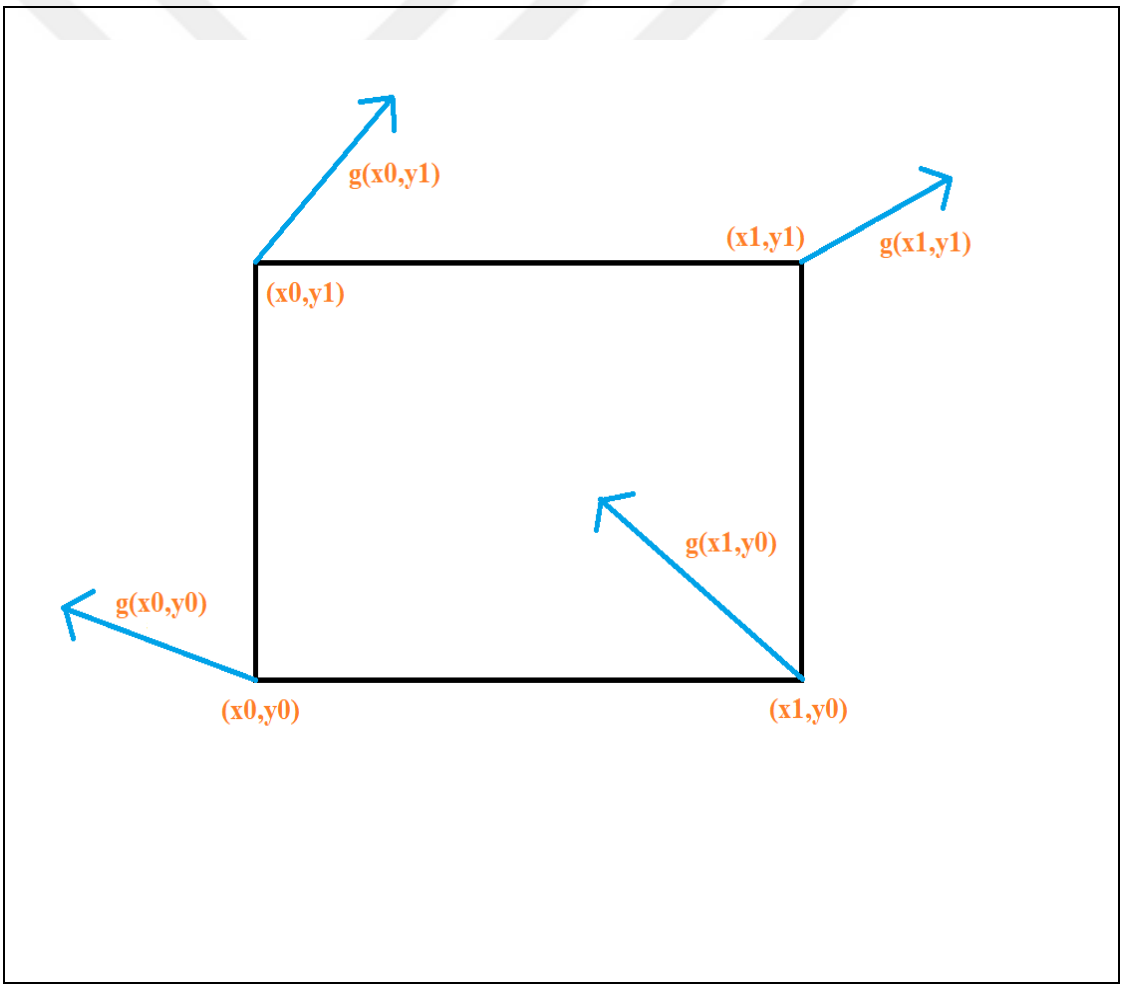


Figure 21. Example Gradient Vectors

Red dot is coordinate (x, z) of a vertex in Figure 22. Perlin value is generated at that vertex coordinate. Perlin value determines the height of that vertex. There are always 4 corner

values (coordinates of corners are whole numbers) in any coordinate in a 2D plane. So perlin function can be calculated at any point in a 2D plane (as 2D perlin noise is used in our application).

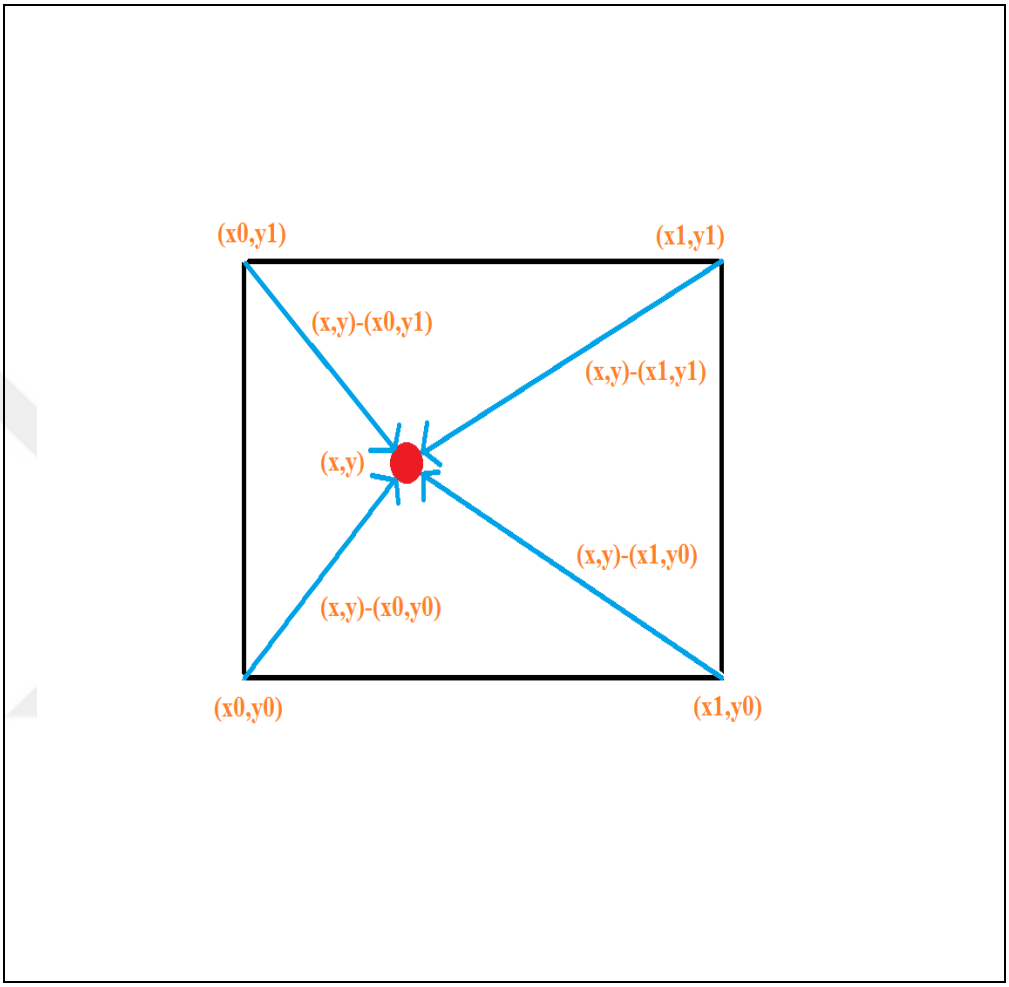


Figure 22. Distance Vectors

Dot product of distance vectors and gradient vectors are calculated. Formula is presented in Equation 5.2. *s*, *t*, *u*, *v* are the influence values of corners and presented in Figure 23.

$$s = g(x0, y0) \,.\, \big((x, y) - (x0, y0)\big)$$

$$t = g(x1, y0) \,.\, \big((x, y) - (x1, y0)\big)$$

$$u = g(x0, y1) \,.\, \big((x, y) - (x0, y1)\big) \qquad (5.2)$$

$$v = g(x1, y1) \,.\, \big((x, y) - (x1, y1)\big)$$

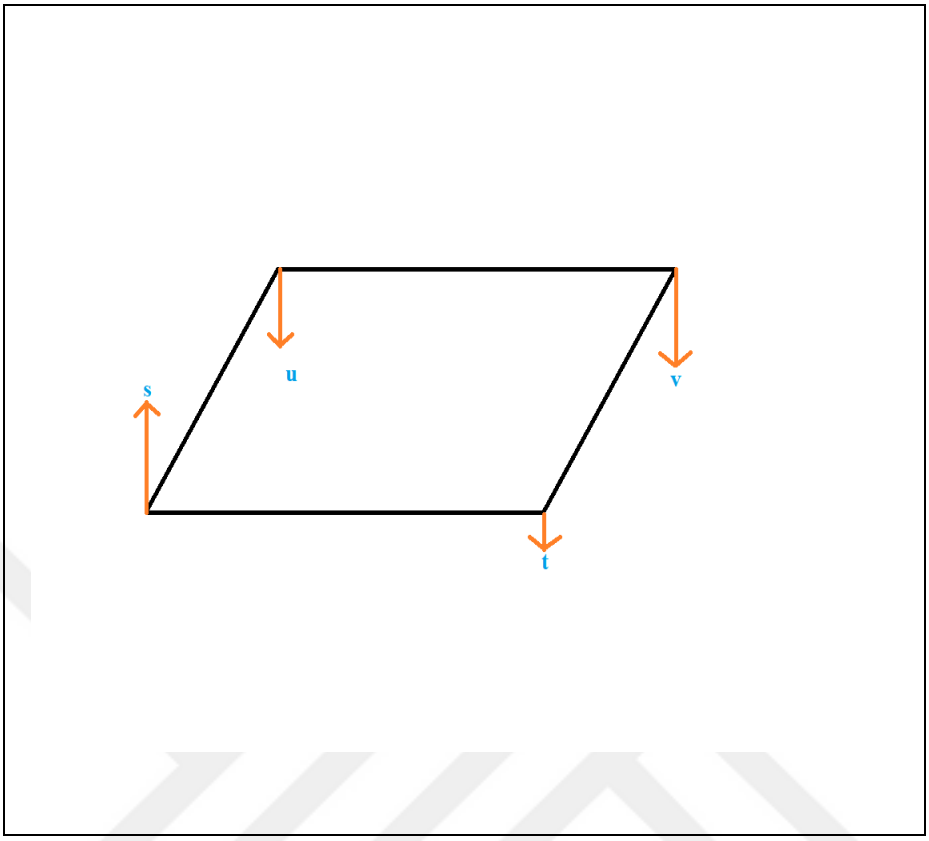


Figure 23. Corner influences

In order to find the average effect of four corners, average value of 4 dot products are taken. But the resulting 2D noise is not a smooth function. So, a fade function should be applied to the resulting noise. Fade function is presented in Figure 24. Fade effect is calculated by using Equation 5.3. *y* is the average value of influence effects, *Y* is the Perlin function value at the input coordinate.

$$Y = 6y^5 - 15y^4 + 10y^3 \tag{5.3}$$

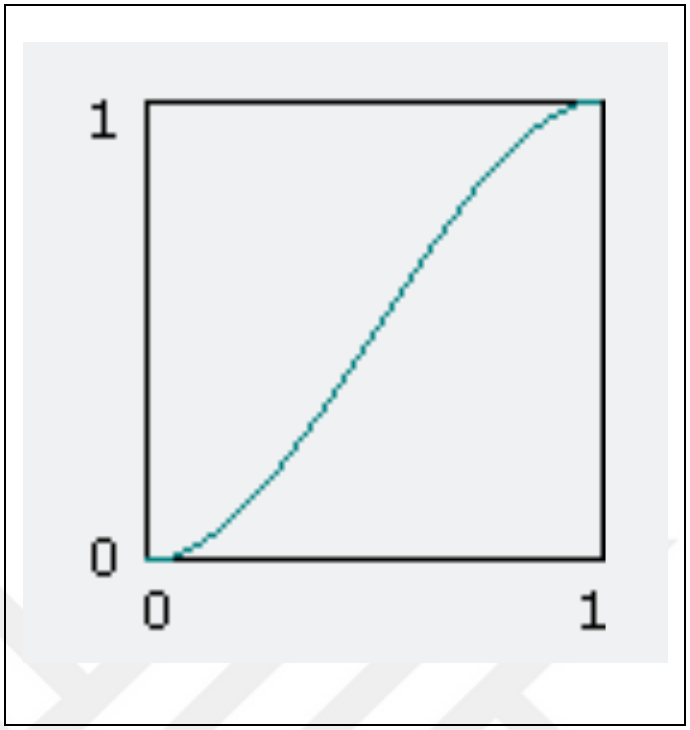


Figure 24. Fade Function

5.1.1. Octaves

Even though Perlin noise does provide a certain degree of natural behavior, it does not fully express the irregularities that is expected in nature. So additional octaves are added to Perlin noise. At each octave, frequency is multiplied by a number (where number > 1) and amplitude is multiplied by another number (where number < 1). As frequency of the signal increases, amplitude of the signal decreases. Therefore, higher frequency noise signals have less effect on the total signal. Octave is the total number of noise signals. Persistence is the amplitude multiplier which should be less than 1. Lacunarity is the frequency multiplier which is generally greater than 1. A lacunarity of 2.0 works fine. Adding octaves to Perlin Noise function is presented in algorithm 2.

Algorithm 2. Octave Calculation

```
total = 0;
freq =1
amp=1
for(i=0; i<octaves; i++){
   total +=amp*Perlin(x*freq, y*freq , z*freq)
   value += amp
   amp *= persistence
   freq *= lacunarity
}
Result = total / value
```

5.2. VERTEX GENERATION

Procedural noise creates a 2d array of values between 0-1 where 1 represents the top height and 0 represents the bottom height. Each value in the 2d array is assigned to a vertex in the mesh in Cartesian coordinates. Height values of the vertices can also be adjusted by multiplying with a maximum height value.

5.3. HEXAGONAL TILE

The simplest hexagonal tile (hex) can be defined with 7 vertices; 1 at center and 6 at the corners. To increase the resolution and support river generation, edge vertices are added. Therefore, there are 13 vertices and 12 triangles in a hex. Center, edge and corner vertices is presented in Figure 25.

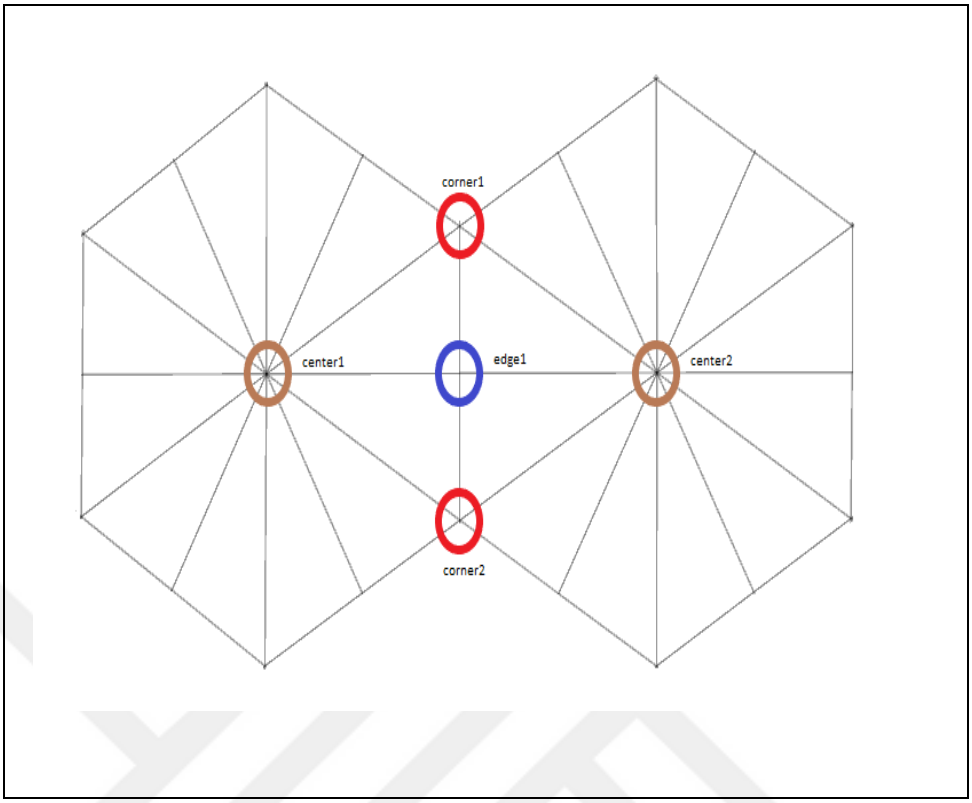


Figure 25. 2 neighbor tiles

The map can be generated by reproducing 13 vertices hex. But generating a map like this creates some problems. There are duplicate vertices. There are 2 vertices for edge locations as 2 hexes intersect at the edges and there are 3 vertices for corner locations as 3 hexes intersect at the corners. When vertices manipulation is needed, 2 or 3 vertices belonging to different hexes should be manipulated. Doing this increases also the total number of vertices.

Instead of doing this, all vertices are generated together and triangles are formed. Therefore, a unit frame is needed to support hexagonal tiling. Figure 26a is the unit frame. The mesh is produced by replicating the unit frame in x and z directions. Figure 26b shows the hex formed by two-unit frames. There are 4 half hexes in our unit frame, meaning that there are 2 hexes in each unit frame. There is the lower part of a hex in the upper part of the unit frame, there is upper part of a hex in the lower part of the unit frame, there is right part of a hex in the left part of unit frame and there is left part of a hex in the right part of unit frame. Red dots are the centre vertices, blue dots are the corner vertices, and green dots are the edge vertices. As the heights of vertices are manipulated, all vertices must be easily accessible. Therefore, a vertex array is used.

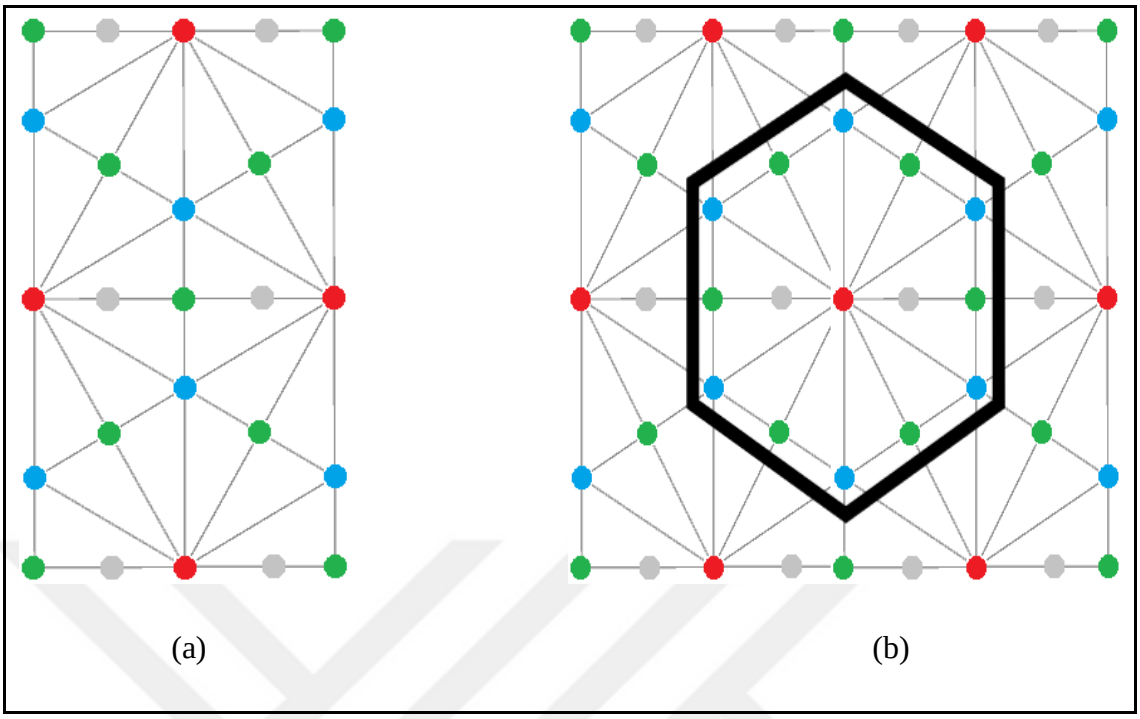


Figure 26. Unit Frame (a) Unit Frame, (b) a Hex in Unit Frame

An unit frame can be stored in 5x5 array (Figure 27). By using this frame, hexes can be generated using cartesian coordinates. Physical locations (x and z locations as y is already defined with height value) of the vertices are different than x and z indices in the array.

4x4 vertices is needed to define a new neighbour unit frame. As there are 2 hexes in a unit frame, we need 8 (4x4 for a frame and div by 2 as there are 2 hexes in a frame) vertices to define a new hex. There 12 triangles in a hex, which is presented in Figure 26b. The number of vertices in the mesh can be calculated using equation 5.4.

$$Number\ of\ vertices = (4x + 1) * (2z + 1) \quad (5.4)$$

x is the number of hexes in x dimension and z is the number of hexes in the z dimension. z must be an even number as 2 vertices are generated the z-direction with 1 unit frame.

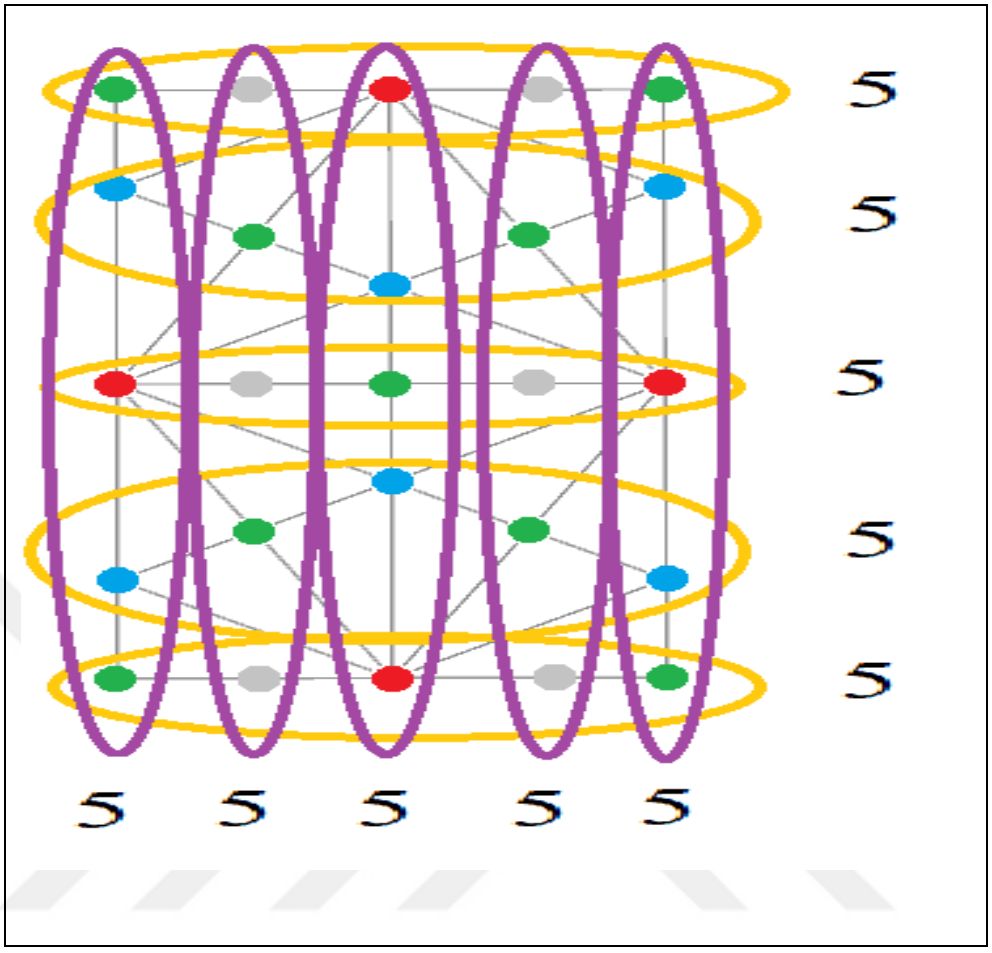


Figure 27. 5x5 unit frame

5.4. HEXAGON INSIDE HEXAGON

A basic map is ready but it does not support easy recognition of tiles yet. For this purpose, all border vertices may be manipulated. If most of the 12 vertices of the hex are manipulated, this may create a gap or peak at the center vertex, which causes a high sharpness. A sample peak is presented in Figure 28. Mountain tiles surrounded by land tiles created peaks. Peaks are circled in red in Figure 28b.

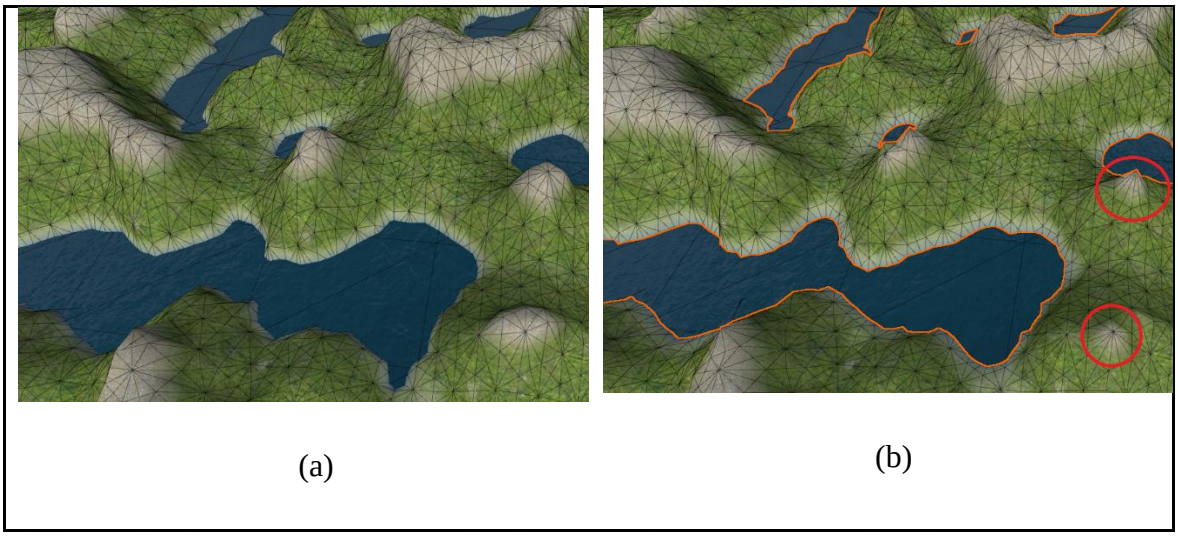


Figure 28. Hex map (a) Non-manipulated, (b) Peak

To solve this problem, we have to increase the vertices that are not corner or edge vertices. The unit frame is changed by adding a new hexagon inside the original. There are 2 hexagons inside each other. The outer hexagon is used for border manipulation and the inner hexagon is used to conserve the shape of the hex and to support easy recognition of the tile.

The new unit frame is shown in Figure 29, where blue vertices belong to the outer hexagon, red vertices belong to the inner hexagon, and yellow vertices are the center vertices. The unit frame consists of 9x9 vertices.

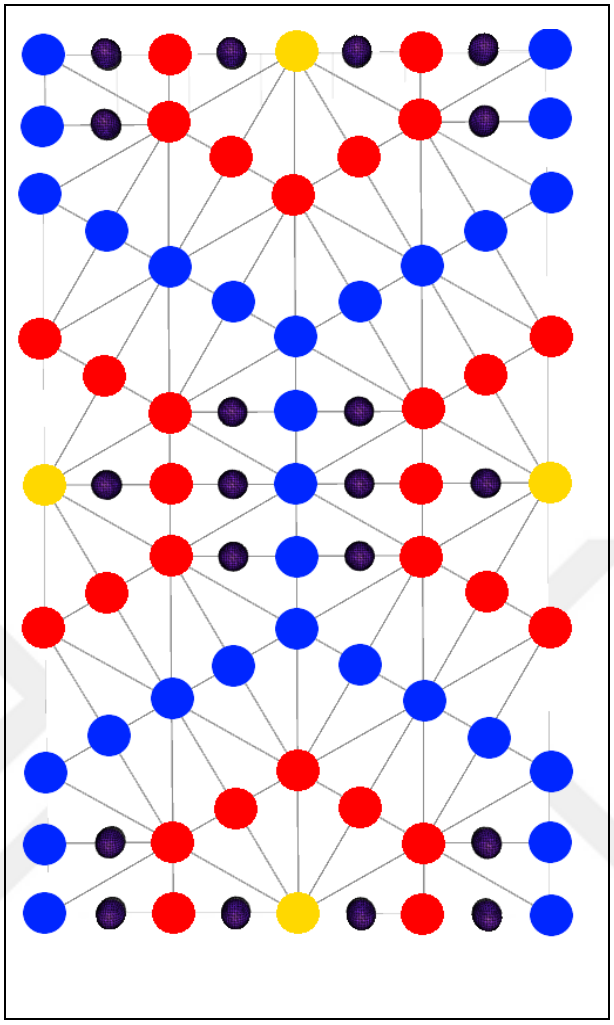


Figure 29. Hex-in-hex Unit Frame

We need 8x8 vertices to define a new neighbour unit frame. As there are 2 hexes in a unit frame, we need 32 vertices to define a new hex. There are 48 triangles in a hex. Equation 5.5 presents the number of vertices calculation.

$$Number\ of\ vertices = (8x + 1) * (4z + 1) \quad (5.5)$$

x is the number of hexes in the x dimension and z is the number of hexes in the z dimension. z is an even number as 2 hexes are generated in the z direction with 1 unit frame

5.5. VERTEX MANIPULATION

All **tile types** have a height range. These heights may be scaled from 0 to 1; 0 is used for the minimum height and 1 for the maximum height. The sea tile type starts from 0 and ends at a

threshold value (sea threshold). Mountains start from another threshold value (mountain threshold) and end at 1. Everything between these two thresholds is considered as land tile type.

In a tile, the height of the center vertex defines the tile type (sea, land, and mountain). The heights of all tile vertices must be between the threshold values of their tile type. Otherwise, it must be manipulated by shifting the vertex up or down.

Each hex has 5x9 vertices. In order not to create sharpness each vertex must be shifted up or down according to its location in the hex. If a manipulated vertex is on the outer hexagonal, its height must be closer to the threshold height. If a manipulated vertex is on the inner hexagonal, its height must be closer to the height of the center. Vertex conversion algorithms are given in Algorithm 3. Colours are presented in Figure 27.

Algorithm 3. Vertex Conversion Algorithms

```
//land to mountain
if (vertex height < mountain_threshold)
      if (vertex is blue )//border vertices
         vertex height= mountain_threshold
      if (vertex is red ) // inner vertices
         vertex height= (mountain_threshold + vertex height) / 2
      // if (vertex is yellow ) vertex height is not changed
//land to sea
if (vertex height > sea_threshold)
      if (vertex is blue )//border vertices
         vertex height= sea_threshold
      if (vertex is red ) // inner vertices
         vertex height= (sea_threshold + vertex height) / 2
      // if (vertex is yellow ) vertex height is not changed
//sea to land
if (vertex height < sea_threshold)
      if (vertex is blue )//border vertices
         vertex height= sea_threshold
      if (vertex is red ) // inner vertices
         vertex height= (sea_threshold + vertex height) / 2
      // if (vertex is yellow ) vertex height is not changed
//mountain to land
if (vertex height > mountain_threshold)
      if (vertex is blue )//border vertices
         vertex height= mountain_threshold
      if (vertex is red ) // inner vertices
         vertex height= (mountain_threshold + vertex height) / 2
      // if (vertex is yellow ) vertex height is not changed
```

In Figure 30, blue points mark the vertices that are converted from land to sea, orange points mark the vertices that are converted from sea to land, red points mark the vertices that are converted from mountain to land, and green points mark the vertices that are converted from land to mountain.

When there are two neighbouring tiles having different tile types, only one side need vertex manipulation. Manipulated side defines the behaviour of intersecting vertices. Behaviour of corner vertices is organized according to three intersecting tiles. If two tiles have the same tile type, then corner vertex is manipulated according to that tile type.

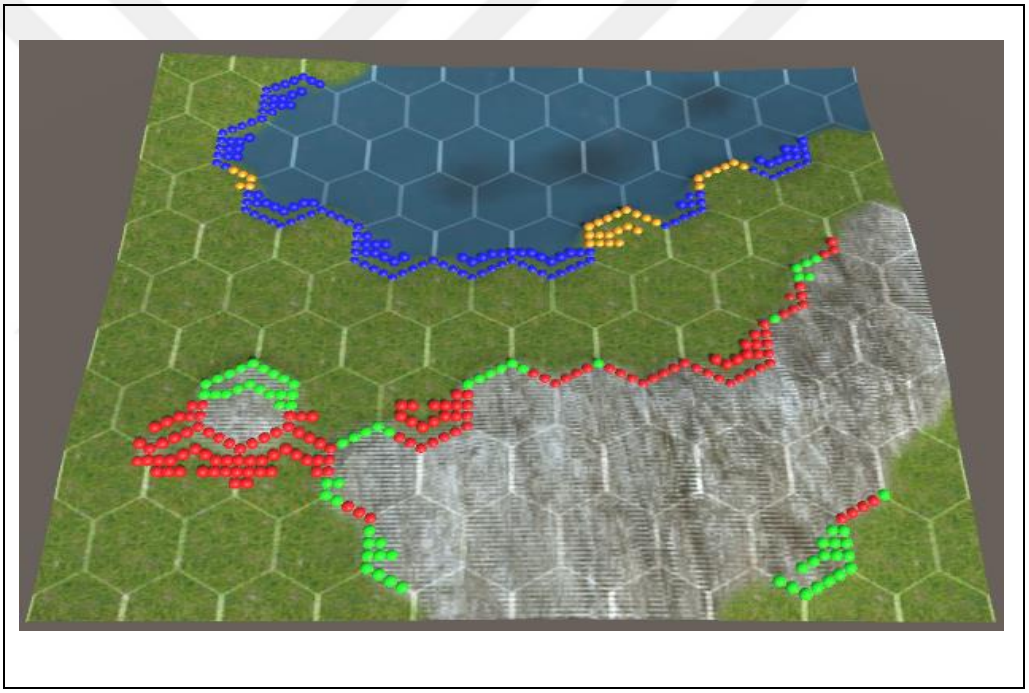


Figure 30. Manipulation Vertices

Coefficients in vertex manipulation are adjusted for minimum curvature. **Threshold levels** are transition heights between different terrain types such as height level between sea and land. In Figure 31, vertices are coloured according to their distances to the center, where yellow is the center vertex, reds are inner vertices and blues are outer vertices.

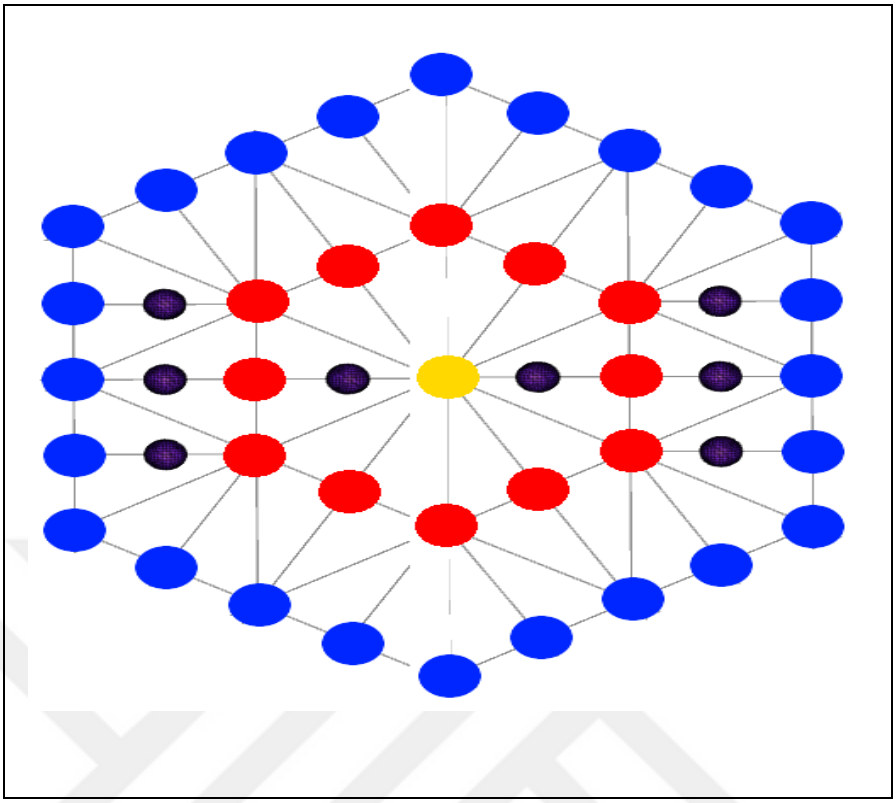


Figure 31. Tile Vertex Numbering

5.6. LINEAR INTERPOLATION

In Figure 32, linear interpolation is applied for easy recognition of shores. Ağrı Mountain is the highest elevation about 5km in this map and can be accepted as 1.0. Sea level is black and can be accepted as 0.0. I know that lands of Istanbul are not below 100m except some hills but coloured as grey. Grey is 0.5. If linear interpolation is not applied, colour of Istanbul lands must be 100*255/5000=5 where 5 is almost black. But if linear interpolation is not applied shores of Istanbul can not be recognized.

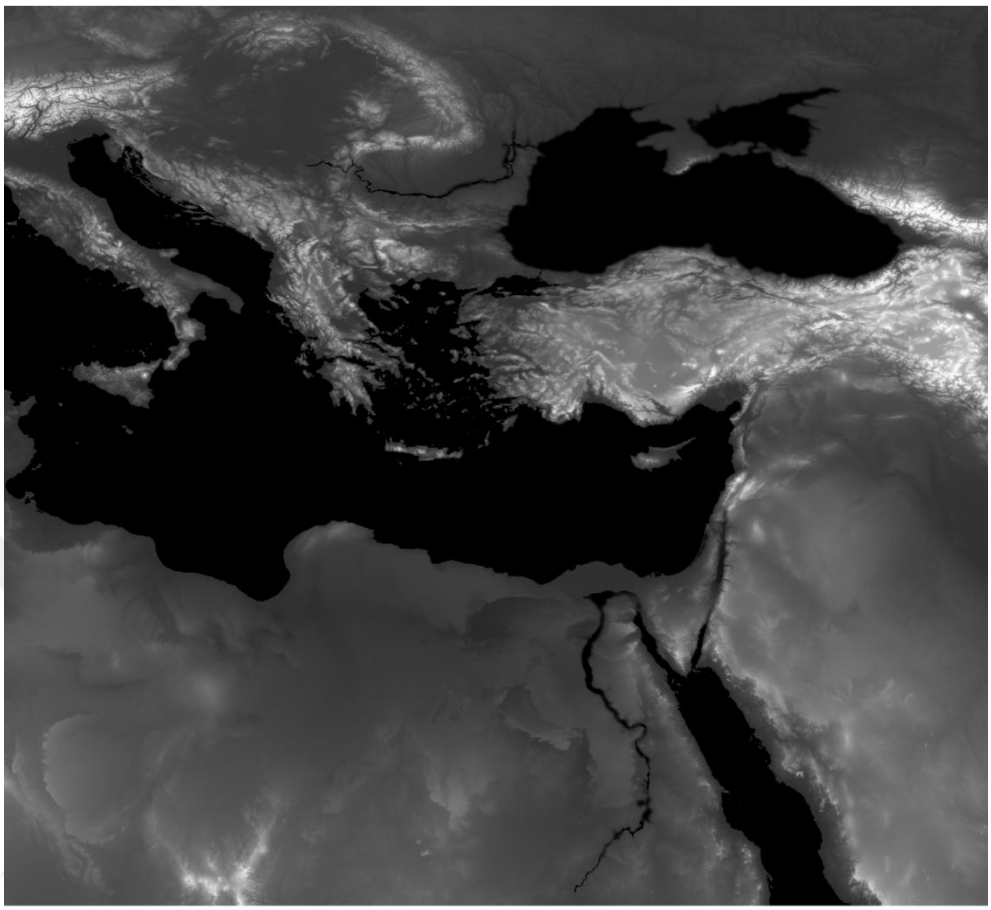


Figure 32. East Mediterian Map

In this work, in order to distinguish the hexes in threshold levels, linear interpolation is used. If some sea levels are just a little below the sea level, they must be shifted down to distinguish those vertices from land vertices. However, if interpolation is done so much, sharpness increases. Sea tiles look like land tiles if these tiles are just a little below the sea level threshold. Blending effect in the shader code make them look like sea tiles. Appearance is corrected by using linear interpolation. Linear interpolation algorithm is given in Algorithm 4. Any vertex between 0 to sea level range is interpolated between 0 to (seaLevel-interpolationTolerance) range. Where sea-level is between 0 and 1. Linear interpolation graph is presented in Figure 33. Blue line is the non-manipulated line and red line is the manipulation affect.

Algorithm 4. Linear Interpolation Algorithm

```
LinearInterpolation (inmax,inmin,outmax,outmin,input)
  output = ((outmax-outmin)*input+outmin*inmax-inmin*outmax)/(inmax-inmin);
```

inmin - inmax is the interpolating range. outmin – outmax is the interpolated range. Interpolation tolerance is the difference between interpolating range and interpolated range. For mountain level max values of input and output are 1 and same. For sea level min values of input and output are 0 and same. For the land values there is an interpolation difference for both min and max values. Equation 5.6 presents the relation between interpolation tolerance and minimum, maximum input, output values

$$\begin{aligned} &for\ mountain \\ &\quad interpolation\ tolerance = outmin - inmin \\ &for\ land \\ &\quad interpolation\ tolerance = outmax - inmax = outmin - inmin \\ &for\ sea \\ &\quad interpolation\ tolerance = outmax - inmax \end{aligned} \tag{5.6}$$

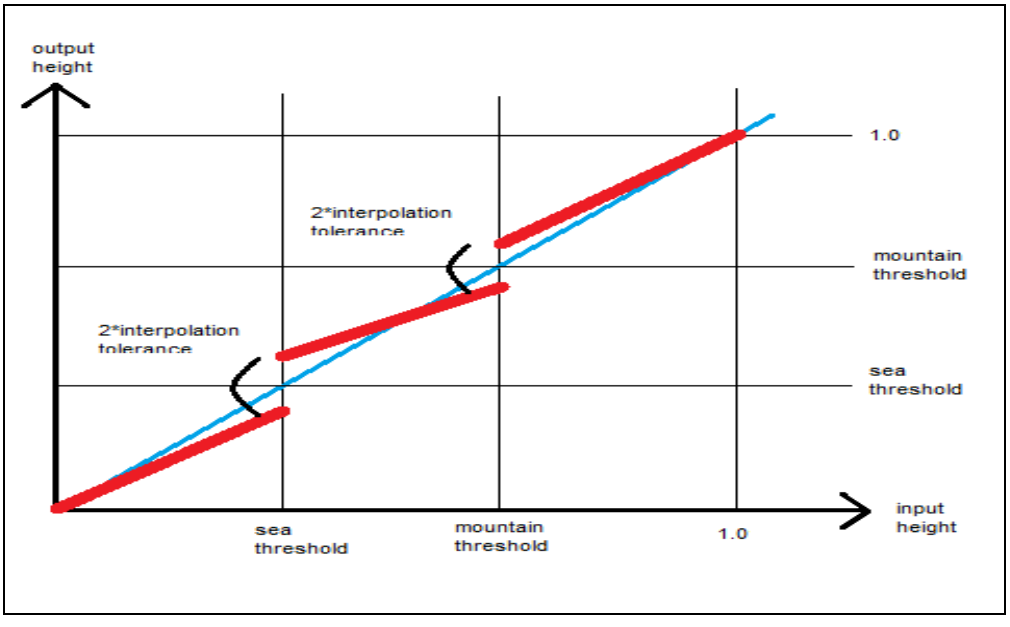


Figure 33. Linear Interpolation Function

6. IMPLEMENTATION

Unity game engine is used for the implementation, but specific tools or assets of unity such as terrain asset are not used. So, this implementation can be converted to Open GL or DirectX application. It can be also used with other game engines such as Godot or Unreal.

Basic PTG has 2 parts

1. Mesh Generation
2. Texturing

This application has 3 parts.

1. Vertex Manipulation
    a. Hex Unity
    b. Linear Interpolation
    c. Edge/Corner Correction
2. Mesh Generation
3. Texturing

6.1. NOISE GENERATION

2D noise signal is created by using Perlin noise. Perlin parameters of reference map are:

Octaves:10

Persistence:0.3

Lacunarity:2

Scale :40

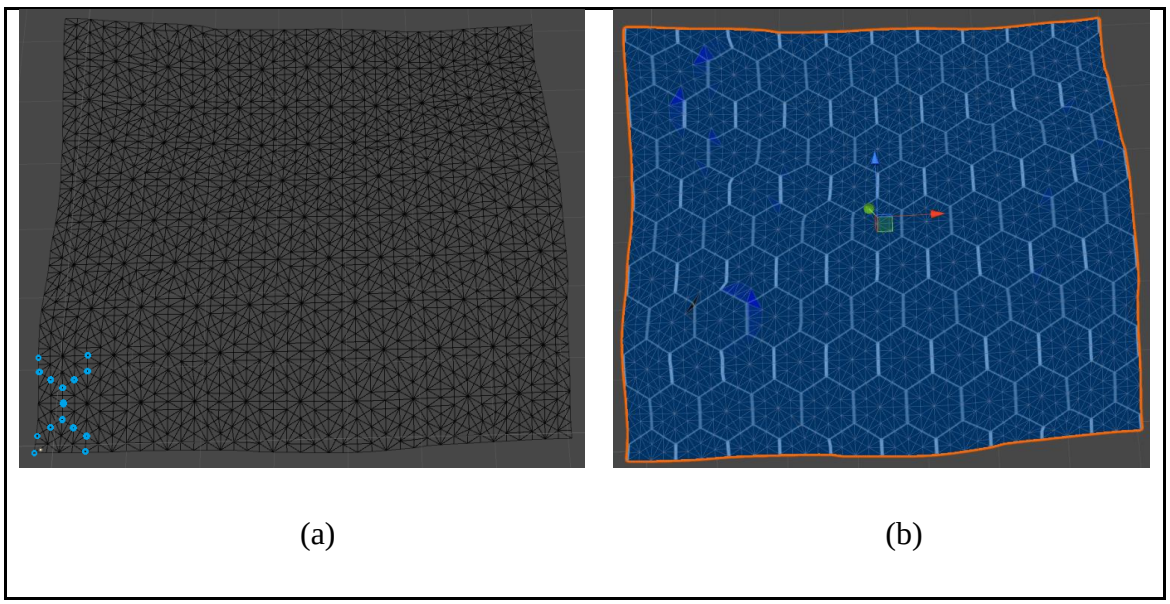
Seed:1000

If this Perlin noise is applied to 50x50 quad formed vertices, the generated heightmap is presented in Figure 34. Although the height map uses the same Perlin parameters, maps are different than the height map. Because hexagonal coordinates are used instead of quad coordinates in the maps.

Figure 34. Quad Heightmap

6.2. MESH GENERATION

The vertices and triangles generated with our method are given in Figure 35. In Figure 35a edges of outer hexagonal (defined in 5.3 hexagon inside hexagon) are marked with blue circles. 96 triangles in one unit frame can be seen. In Figure 35b hexes are created with hex frame. There are 10x5 hex frames forming 10x10 hexagonal map.



(a) (b)

Figure 35. Mesh Generation (a) Hex frame in mesh, (b) Meshes

6.3. TEXTURING

Texturing is done with the shader written in Cg. Cg codes run on graphics processing unit (GPU). So, they consume less time compared to codes running on central processing unit (CPU) [15] . There are 3 layers; sea, land and mountain. Layers are divided with threshold levels. There are 2 threshold levels to divide 3 layers. Min. value is 0.0 and max. value is 1.0.

Mountain height range is between mountain threshold and 1.0. Land height range is between sea threshold and mountain threshold. Sea height range is between 0.0 and sea threshold.

6.3.1. Color texturing without blending

Each layer is painted with different colours. Colour painting according to vertex heights can be seen in Figure 36. Mountain is painted yellow. Land is painted red. Sea is painted blue. If there is no blending, map is painted as presented in Figure 37.

Figure 36. Height Ranges

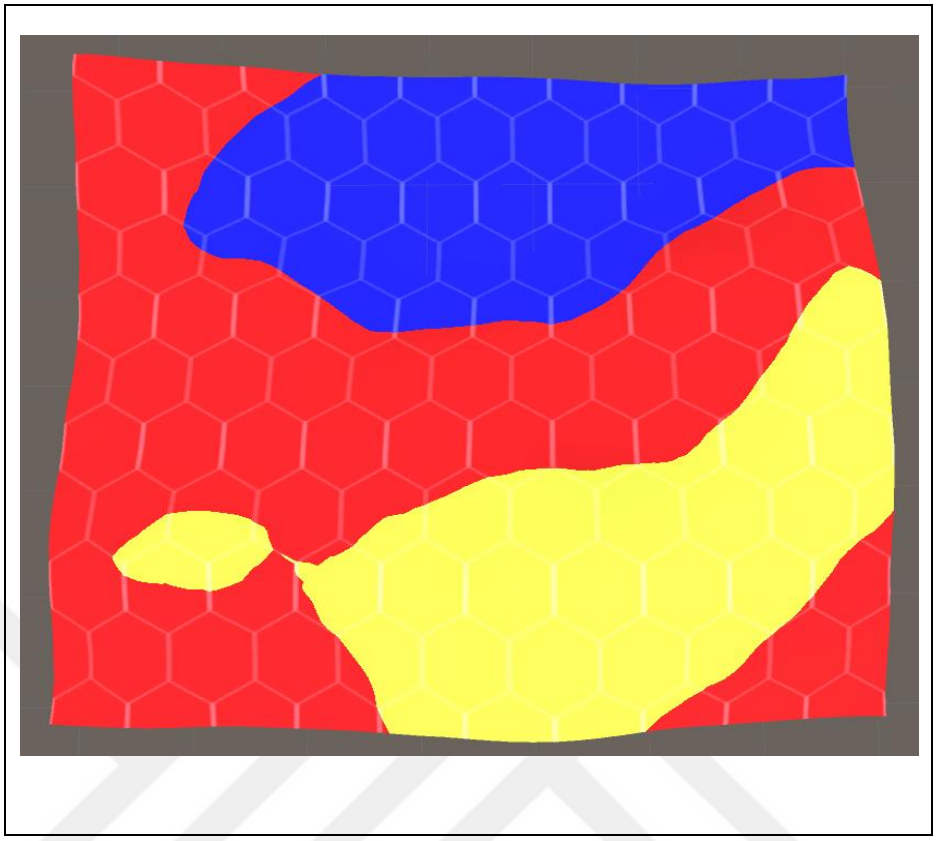


Figure 37. No Blending

6.3.2. Color texturing with blending

There must be soft transition between layers. Linear blending [17] is used. Each layer interferes into other layers depending on the distance to threshold level. Sea interferes to land. Mountain interferes to land. Land interferes to both sea and mountain.

Drawstrength is the interference affect depending on distance to threshold level. The shorter the distance between vertex and threshold level is, the greater the drawstrength is. Blendvalue is the distance of blending. Texture is the affecting texture or colour. Albedo is the colour of any location. Surface Shader in Algorithm 5 is used for texturing the mesh.

Algorithm 5. Surface Shader Algorithm

```
For (all pixels and layers)
  If ((surface.y coordinate > threshold) &&
     (surface.y coordinate < threshold + blendvalue))
     drawStrength = | thresholdlevel+ blendvalue -pixel y coordinate|
   else drawStrength = 0;
   Albedo = (1-drawStrength)*Albedo + texture *drawStrength
```

Blending regions is presented in Figure 38. First blending region (orange) is the transition region between mountain (yellow) and land (red). Second blending region (purple) is the transition region between sea (blue) and land (red).

Figure 38. Blending Ranges

With 0.1 blending tolerance, the blending effect is presented in Figure 33. Orange and purple regions can be observed in Figure 39. There is a soft transition between colors in Figure 35. This soft transition is succeeded by using blending.

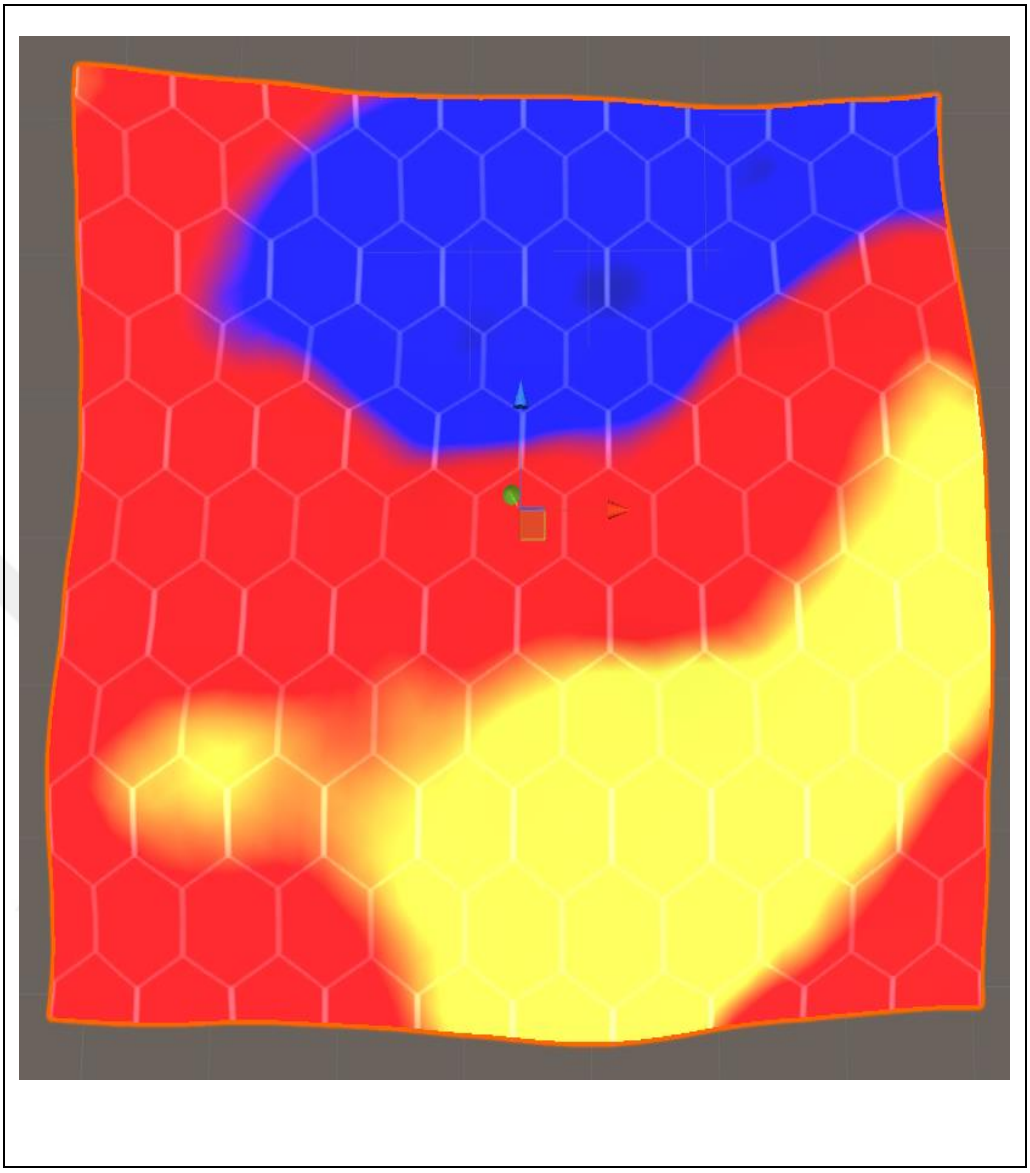


Figure 39. Color Blending

6.3.3. Texturing with blending

Texturing in Figure 39 is not realistic, because a layer never consists of one color. Textures should be used for more realistic maps. Textures presented in Figure 34 has been used for texturing instead of colors.

Below sea level, sea texture seen in Figure 40a is used; between sea level and mountain level, grass texture seen in Figure 40b is used; above mountain level, mountain texture seen in Figure 40c is used.

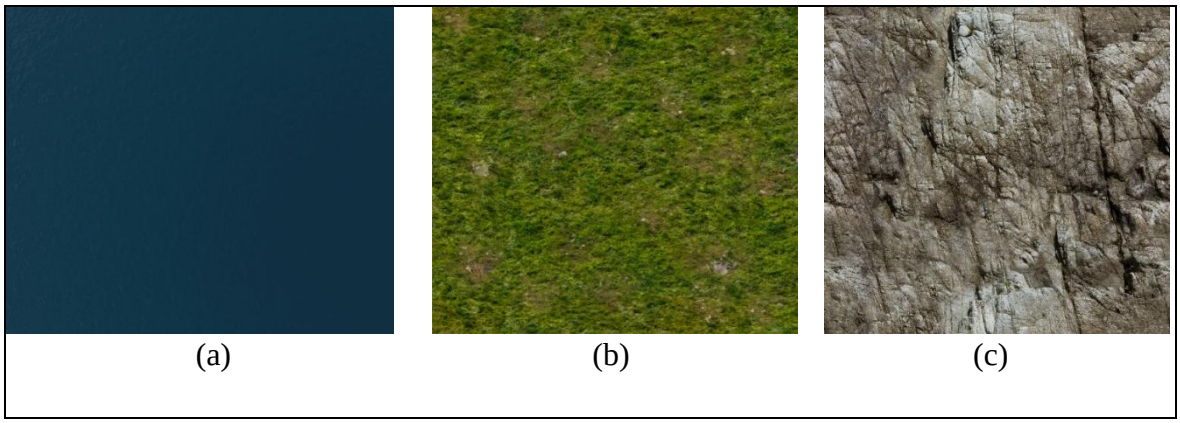


Figure 40. Textures used in the map (a) sea texture, (b) grass texture, (c) mountain texture

Blending distance of textures can be adjusted by varying “blendvalue” in algorithm 5. Map in Figure 41a has high sharpness and shore line can be easily detected. However, map in Figure 41c has low sharpness but shore line can not be easily detected.

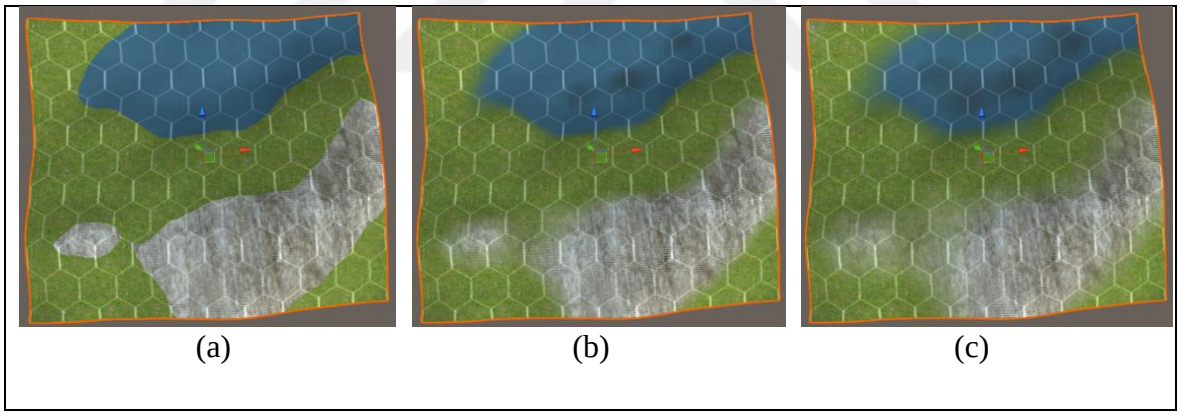
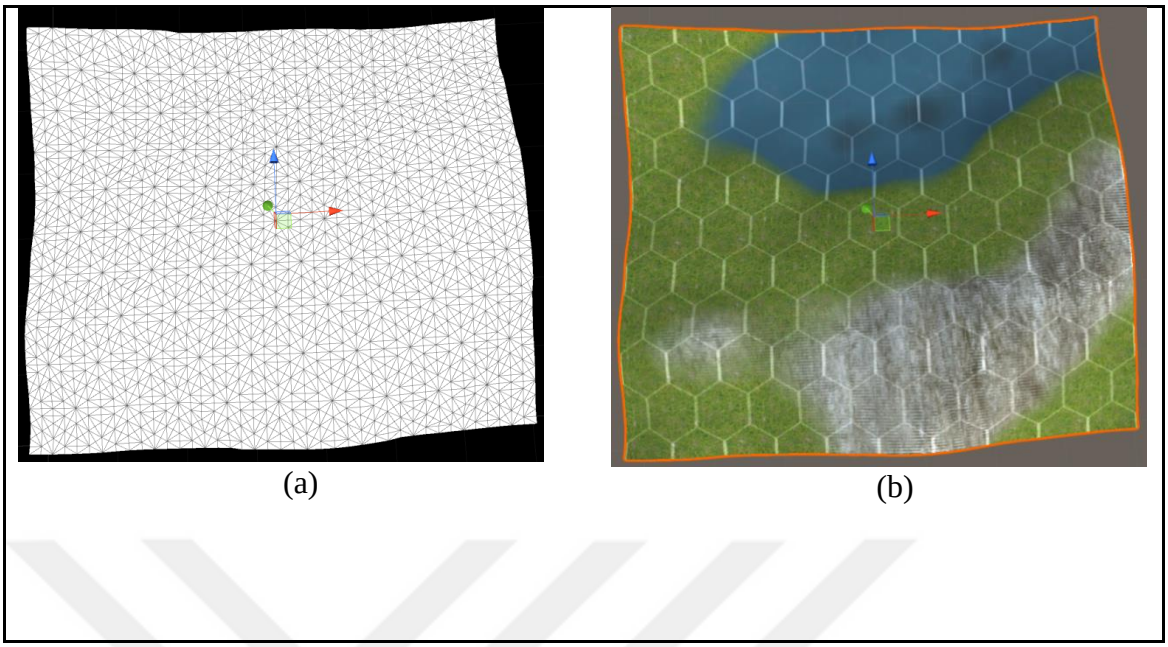


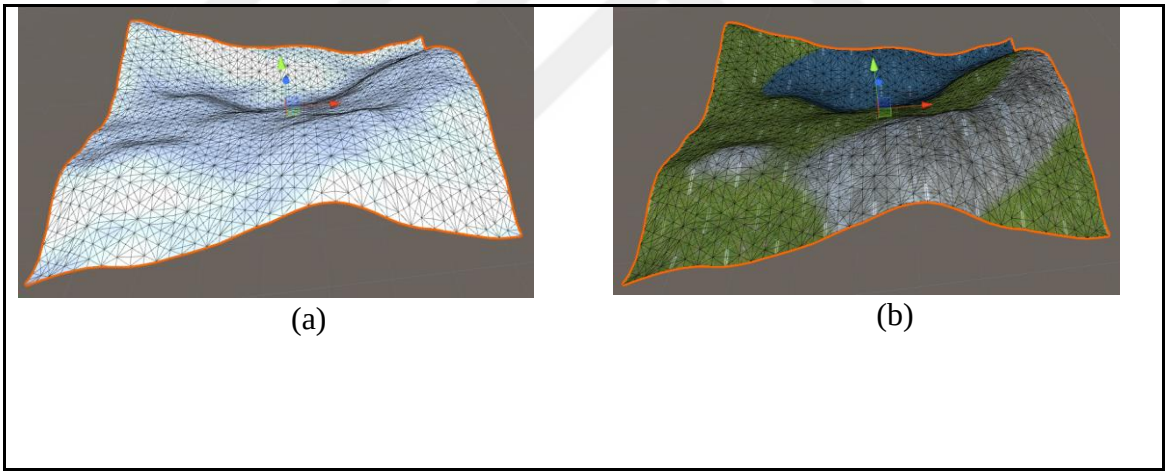
Figure 41. Map Blending (a) 0 blending, (b) 0.1 blending, (c) 0.2 blending

In Figure 42, non-textured and textured meshes are presented. While empty mesh is presented in Figure 42a, Figure 42b is a realistic map. In Figure 43, 3D mesh can be observed with a different camera angle. 3D model of mesh can be observed better in Figure 43.



(a) (b)

Figure 42. Mesh Texturing (a) Mesh, (b) Textured Mesh



(a) (b)

Figure 43. Different Camera view (a) Mesh, (b) Textured Mesh

6.4. VERTEX MANIPULATION

In this part vertices are manipulated for easy recognition of tiles. As high sharpness disturb appearance of map, sharpness level should not be increased too much in vertex manipulation

6.4.1. Hex Unity

Vertex height in the middle of the hex defines the tile type. Height of the vertices belonging to that hex are increased or decreased, to the height interval (defined in Section 6.2) of that tile type. This gives us easy recognition of the tiles.

Map in Figure 44a is non-manipulated raw map. Figure 44b shows the vertices that are manipulated. Figure 44c is the manipulated map. Blend value is decreased to 0.02 for easy detection of manipulation affect.

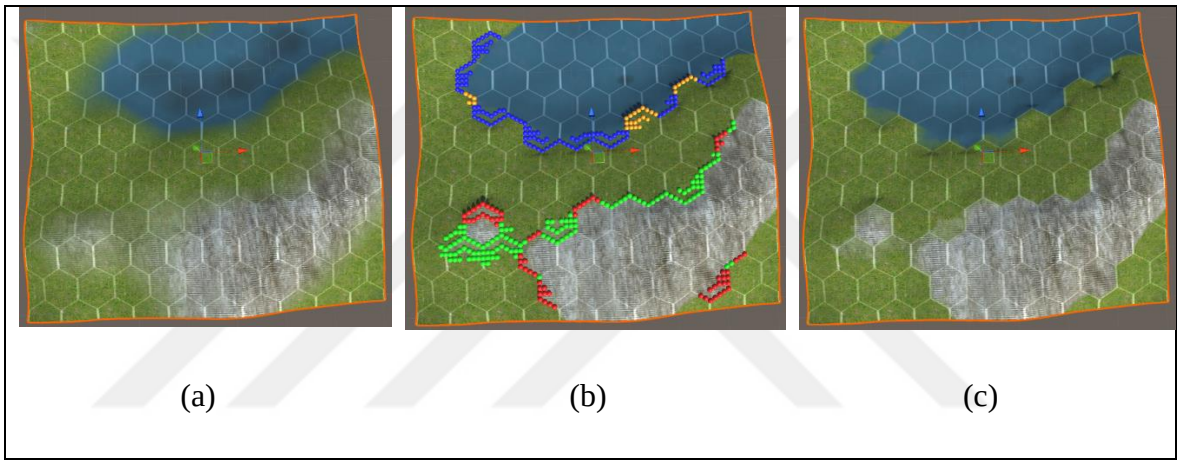


Figure 44. Hex Unity (a) raw map, (b) manipulated vertices, (c) manipulated map

6.4.2. Linear Interpolation

For linear interpolation, algorithm 3 is applied. The blending value is increased from 0.04 to 0.1 to show the effect of linear interpolation more clearly. In Section 6.3.1, blending value is around 0.1 for a better appearance.

In Figure 45a, tiles can not be easily recognizable because of blending. In Figure 45b linear interpolation is applied and all tiles are easily recognizable. Interpolation tolerance (equation 5.6) is 0.03.

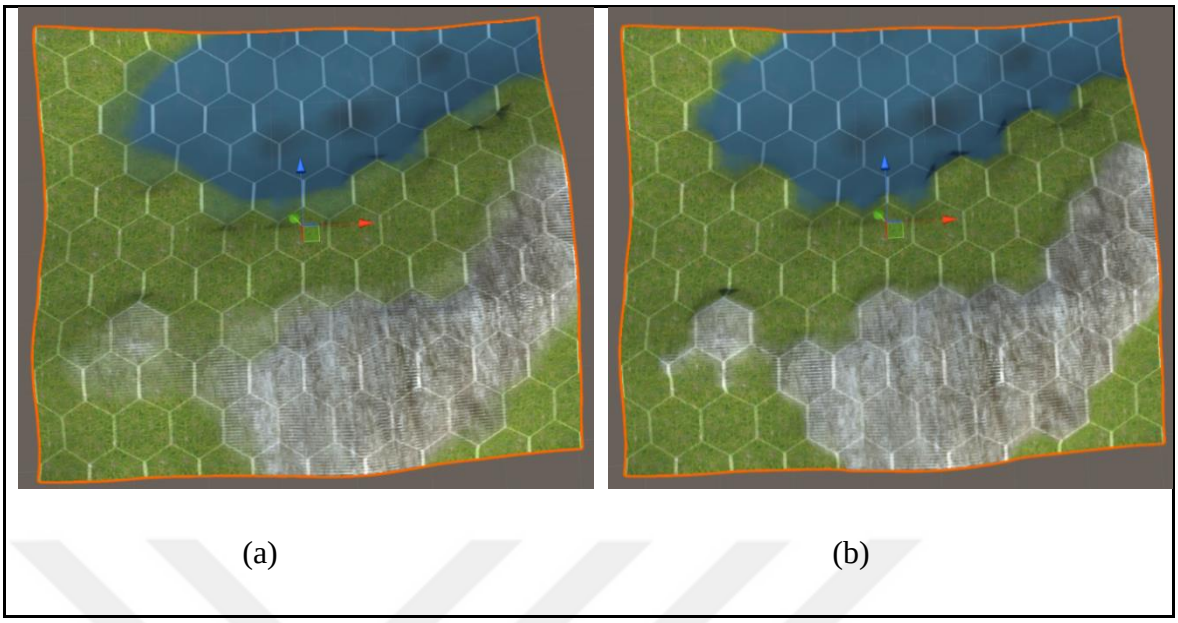
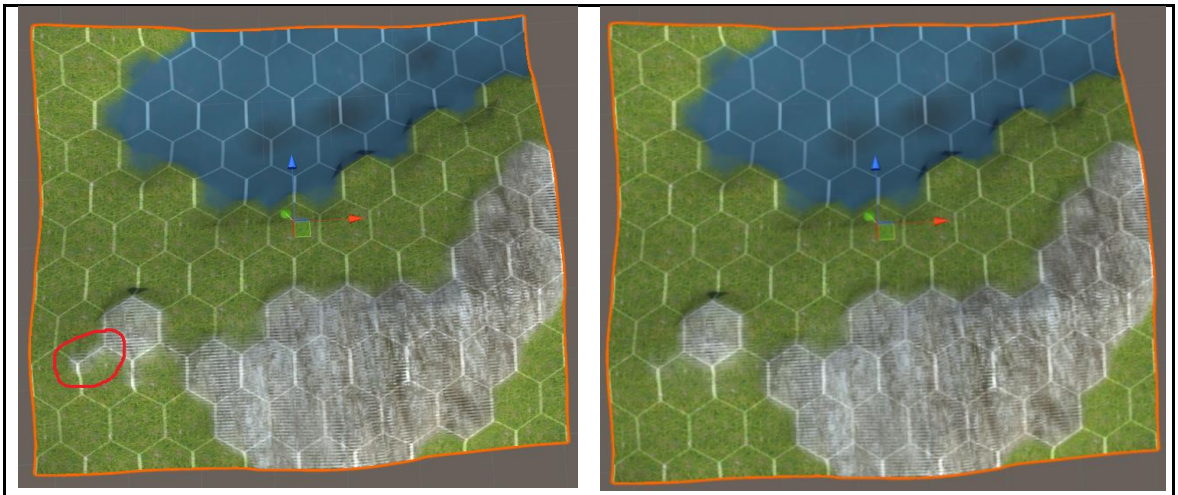


Figure 45. Linear Interpolation implementation (a) no Linear Interpolation, (b) Linear Interpolation applied

6.4.3. Edge/Corner Correction

In some regions; although all tile centers belong to the same tile type, some vertices belong to different tile type. In this case, these vertices are not manipulated correctly when vertex manipulation is done if tile types are different. So, a new algorithm is added for such conditions.

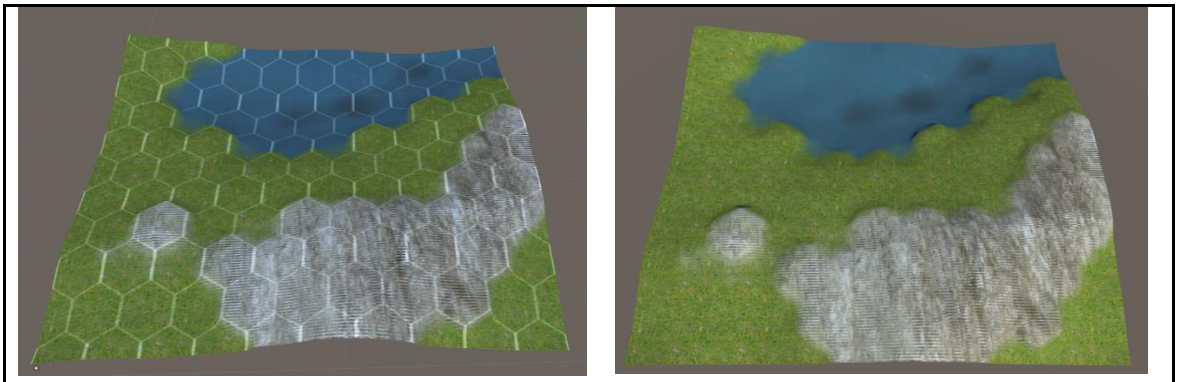
In Figure 46a, edge vertices of neighbor hexes belong to mountain interval level although both hexes are land. Because in the raw map this region is mountain and inner vertices of hexes are converted to land. So, edge/corner correction must be done to correct such vertices. In Figure 46b height of these vertices are decreased to below mountain level according to Figure 46a.



(a) (b)

Figure 46. Edge/Corner Correction (a) No Edge/Corner Correction, (b) Edge/Corner Correction applied

When we removed tiling, it is hard to detect tiles as seen in Figure 47. There is sufficient soft transition between tiles.



(a) (b)

Figure 47. Tiles (a) Marker enabled, (b) Marker disabled

7. RESULTS

The system is tested in terms of run-time performance and visual quality that is measured with the sharpness of the mesh. Basic PTG and our approach defined in Chapter 6 are compared. Initial test results have been presented in ISMSIT 2022 [25]. In this chapter, detailed test results are given.

There are three types of variables; Perlin noise parameters, general noise parameters and map parameters. Maps in the previous chapters are generated using the parameters in Tables 2, 3, 4. Each map is given a map number such as Map0, Map1, etc.

Table 2. Perlin Parameters

Perlin Parameters	
seed	1000
z offset	0
x offset	0
map size	10x10

Table 3. Noise Parameters

Noise Parameters	
octaves	10
persistence	0,3
lacunarity	2
noise Scale	40

Table 4. Map Parameters

Map Parameters	
max. Height	10
mountain threshold	0.7
sea threshold	0.3

In the following sections, the effect of each parameter is modelled and analysed. Table 2 parameters are modelled in Section 7.1. Table 3 parameters are modelled in Section 7.2. Table 4 parameters are modelled in Section 7.3

7.1. PERLIN PARAMETER

Seed is the randomness of the map. Changing seed changes pseudo-random gradient vectors explained in Section 5.1. 2D Perlin noise is infinite in both x and z direction. While creating a finite map, a finite 2D window is taken from this infinite noise. Offset values represent the distance of this window from reference center. Map size represent the size of the window. 2D Perlin plane is presented in Figure 48.

Vertex shader is used in the application. Fragment shaders are not used. Because appearance of a mesh can be shown different than its geometry by using a fragment shader. So, pixels or normal vectors are not manipulated with Perlin noise. Screenshots taken from 3D meshes are 2D. In order to show the 3D mesh, just a basic linear vertex shader is used. Vertices are shaded depending on their height values. Perlin function is used from the library of .NET.

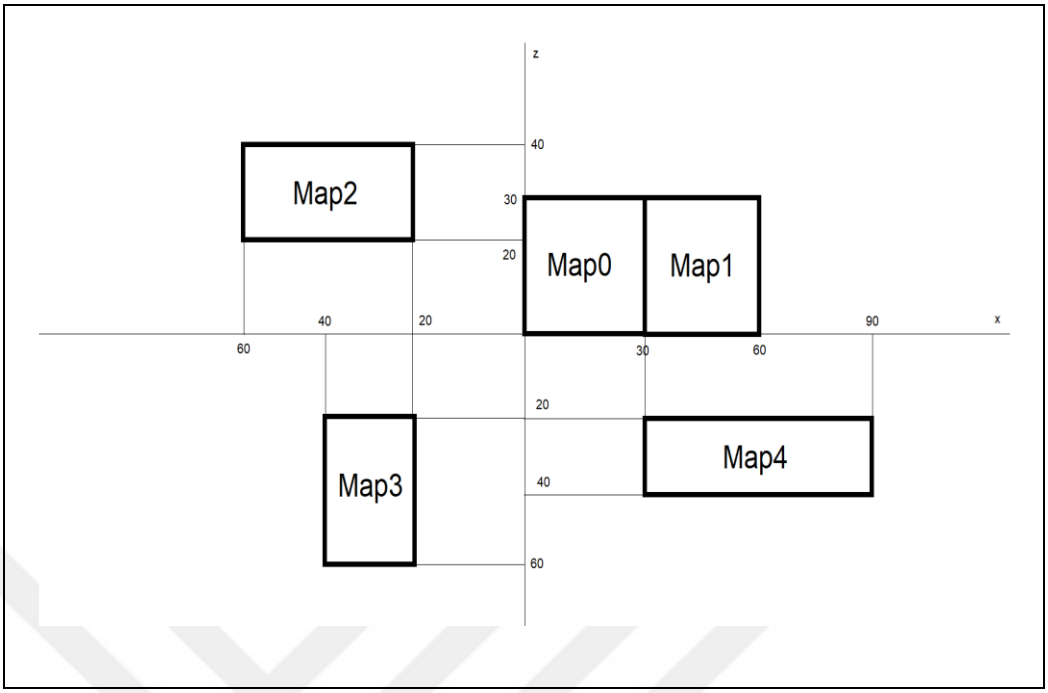


Figure 48. Perlin Plane with seed=1000

Map0 is the region used in this chapters for the results. The effects of changing seed value, shift values and map sizes are presented in the following sections.

7.1.1. Map1-Island

In Map1 an island is generated. In order to convert the map to an island, falloff is used. Map1 has the same dimensions as Map0. It is shifted 30 in x direction. It can be seen in Figure 49. So Map1 (in Figure 49b) should be continuous with Map0 (in Figure 49a). In Figure 49c, a falloff map is applied to map; converting map to an island. In Figure 49d, manipulation functions are enabled and map is converted to a tiled form. It can be observed that the map is tiled correctly. In the application a = 3, b=6 (falloff parameters defined in Section 7.1.1.1). Map1' has the same size and the same shift values as Map1, but Map1' is not converted to island (Figure 49b). Applying falloff converts the edge tiles to sea tiles.

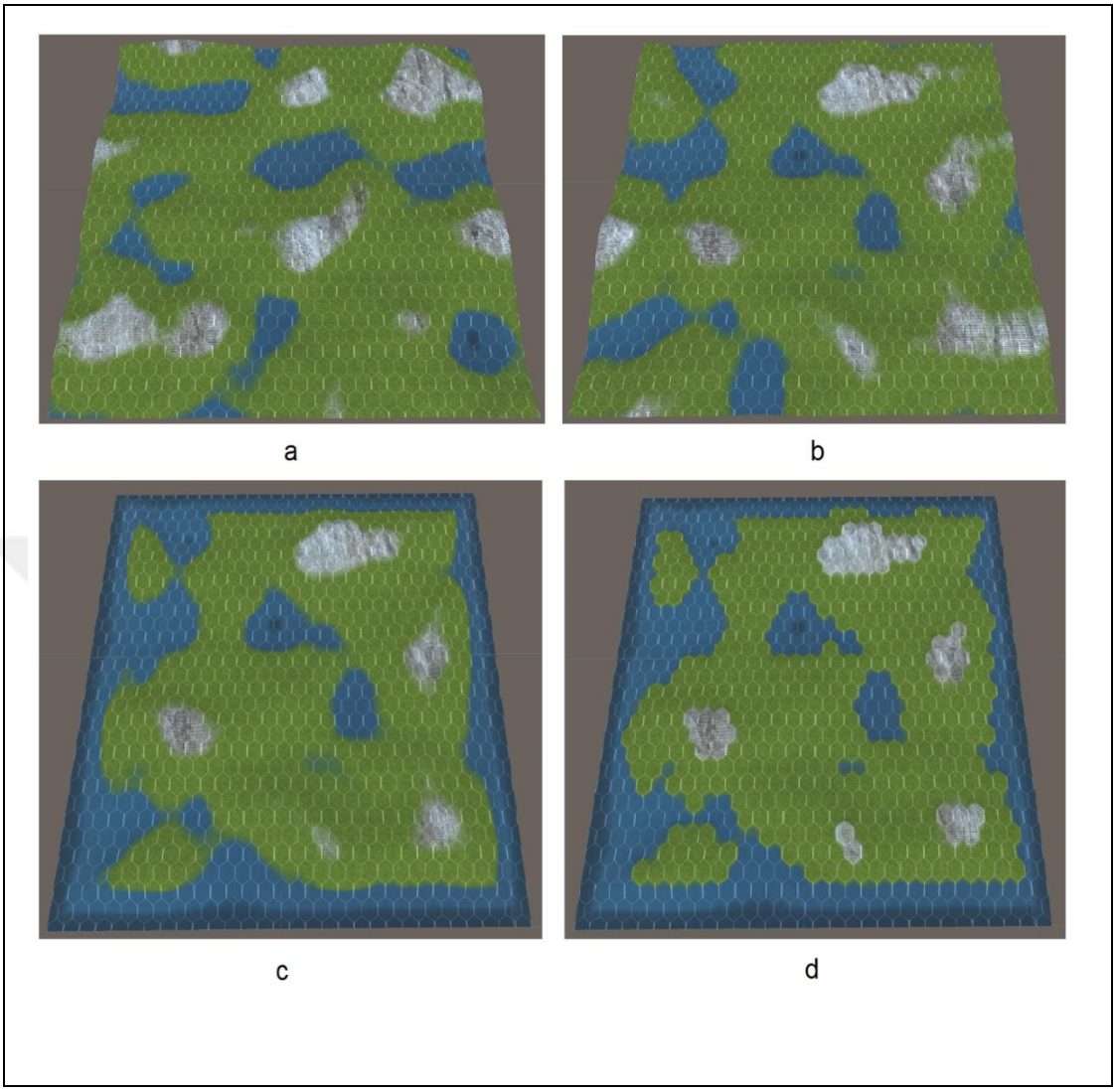


Figure 49. Map1 (a) Raw map, (b) Following map, (c)Island map, (d)Manipulated tiled map

7.1.1.1. Falloff Map

Algorithm 6 creates the map in Figure 50. sizeX is number of vertices in x direction and sizeZ is the number of vertices in z direction. White is 1 and black is 0 in Figure 50 and Figure 51. In the algorithm center is the reference point. At the center x=2*i/sizeX -1 = 0. The value increases as point goes to edges. At the edges x=2*i /sizeX -1 = 1.

Algorithm 6. Falloff Map Algorithm

```
fallofmap = new float[sizeX, sizeZ]
  for (int i=0; i< sizeX; i++){
    for (int j=0; j< sizeZ; j++){
        x = i / (float) sizeX * 2 -1;
        z = j / (float) sizeZ * 2 -1;
        value =Max (Abs (x),Abs (z));
        map [i, j] = value
return map;
```

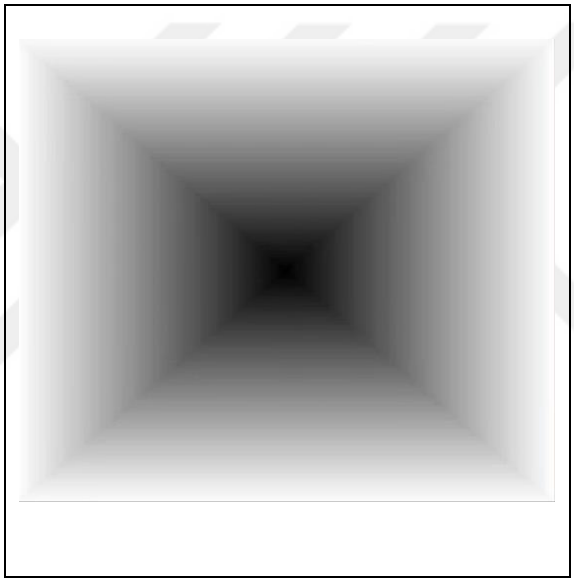


Figure 50. Falloff map

Map in the Figure 50 should be faded because center dark area is too small. Equation 7.1 is used for fading. Fading equation is applied to "value" parameter in Algorithm 6. Figure 51 is the resulting falloff map. Increasing "a" shortens transition region between sea and mountain (or between 0-1). Increasing "b" shortens the sea region in the edges. So, for small maps, smaller "a" values and larger "b" values should be used. Then the map generated by noise is multiplied by 1-f(x) to convert it into an island map. As black regions suppress the area, instead of f(x) , (1-f(x)) is used.

$$f(x) = \frac{x^a}{x^a + (b - bx)^a} \quad 0 < x < 1 \tag{7.1}$$

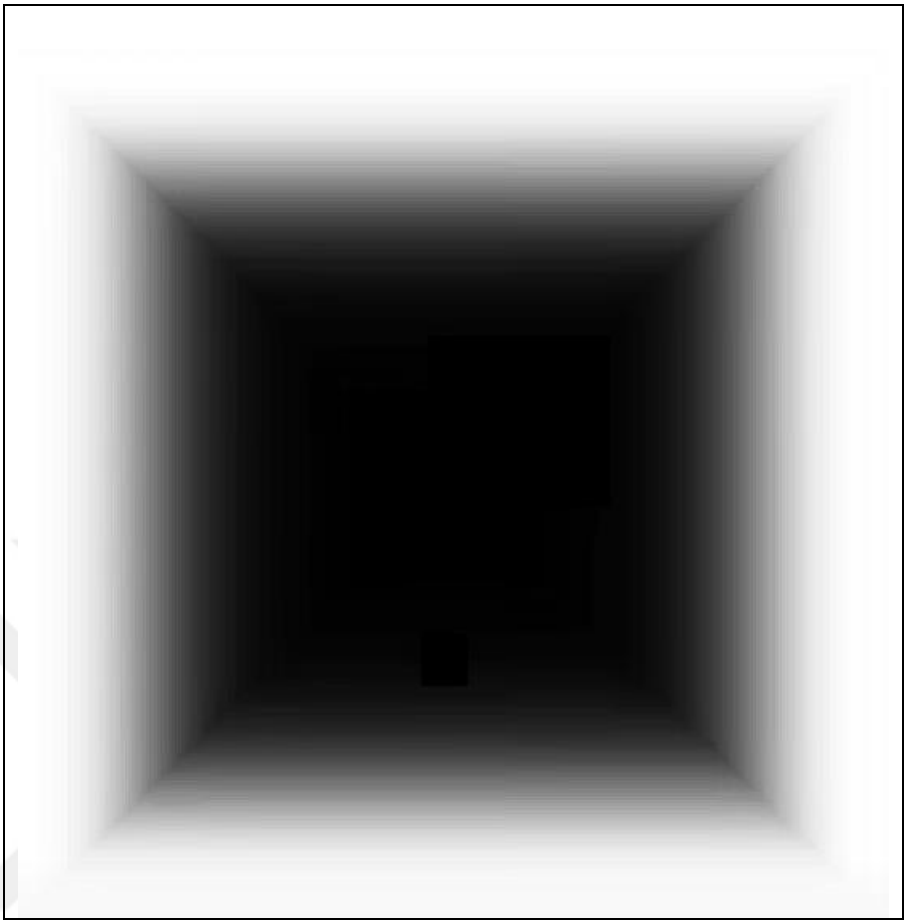


Figure 51. Fading Effect added

7.1.2. Map2-Mediterian

Mediterian is a map type where it has land in the edges and sea in the center.Map2 has size 40x20. It is shifted -60 in x direction and 20 in z direction according to Map0 and the map in Figure52a is generated. Then reverse filter (1-f(t)) is applied in Figure 52b to generate a Mediterian style map. Mediterian style maps have lands in borders and have seas in center. In Figure 52c, manipulation functions are enabled and map is converted to tiled form. It can be observed that map is tiled correctly. Map2' has the same size and same shift values as Map2, but Map2' is not converted to Mediterian map (Figure 52b). In the application a = 1, b=1 (falloff parameters) because we need more region than Map1 as land at the edges.

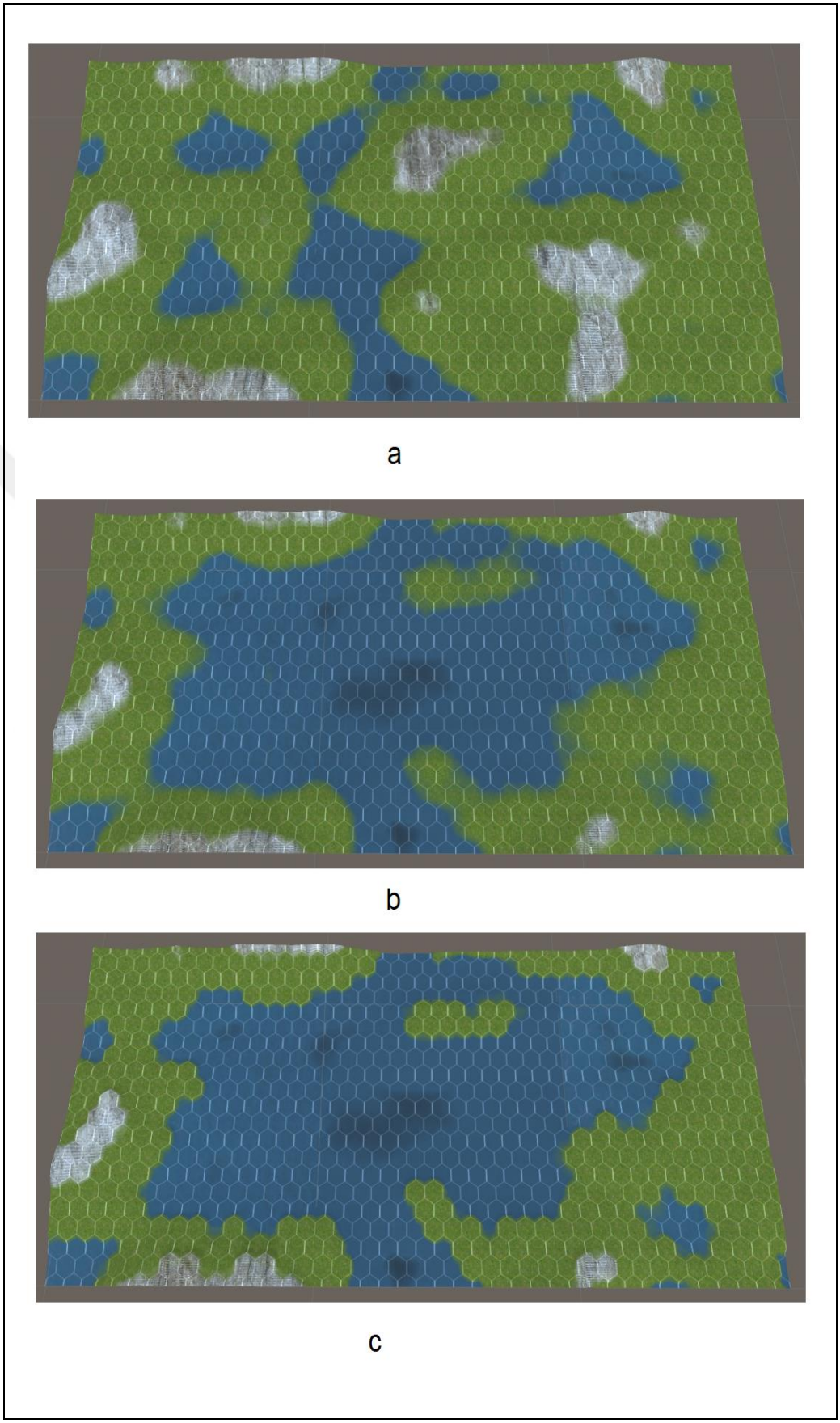


Figure 52. Map2 (a) Raw map, (b) Mediterian map, (c) Manipulated tiled map

7.1.3. Map3-Increased/Decreased Effect

Map3 is generated in order to test increased and decreased effect of Perlin Noise. Map3 has size 20x40. It is shifted -40 in x direction and -60 in z direction according to Map0 and the map in Figure 53a is generated. In Figure 53b, height of map is multiplied by z * mapX / mapZ / x. x and z are the coordinates of vertices. mapX and mapZ are size of map. This process decreases the effect of Perlin Noise. Because z / mapZ decreases at small z values. Perlin Noise has a normal effect in the North part of the map and has a very small effect in the South part of the map. This process also increases the effect of Perlin Noise. Because mapX / x increases at small x values. Perlin Noise has a normal effect in East part of the map and increased effect in West part of the map. Map in Figure 53b is generated. In Figure 53c, manipulation functions are enabled and map is converted to tiled form. It can be observed that map is tiled correctly. Map3' has the same dimension and same shift values with Map3, but Map3' is not multiplied by z * mapX / mapZ / x. It is presented in Equation 7.2.

New height value = old height value * mapX / x * z/ mapZ (7.2)

In north-west side of the map there are sea-mountain transition in neighboring tiles instead of sea-land-mountain transition (which should continue at least 3 tiles). This type of transitions increases sharpness.

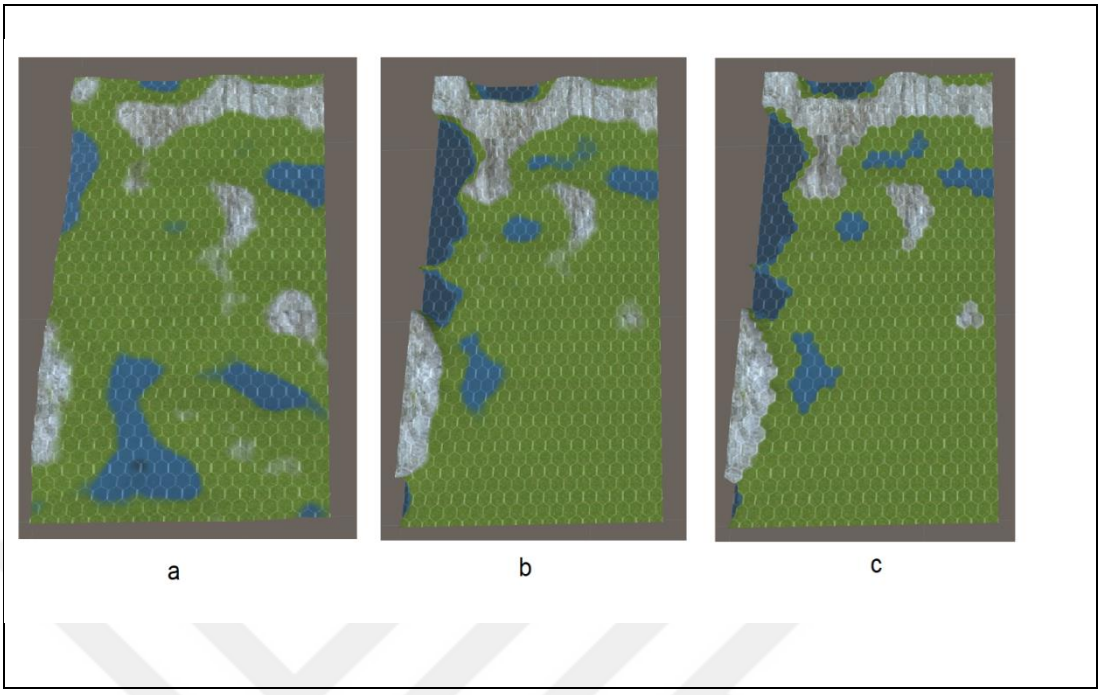


Figure 53. Map3 (a) Raw map, (b) Increased/Decreased effect, (c) Manipulated tiled map

7.1.4. Map4-Inland

Map4 is an inland map. Map4 has size 60x20. It is shifted 30 in x direction and -40 in z direction according to Map0. Maximum Elevation is increased from 10 to 20 to see the map better in Figure 54a. Linear interpolation is applied in Figure 54b. Values between 0-1 range are interpolated to 0.35-1 range to create an inland map. Map in Figure 54b is created. In Figure 54c, manipulation functions are enabled and map is converted to tiled form. It can be observed that map is tiled correctly. Map4' has the same size and the same shift values as Map4, but Map4' is not converted to inland map (Figure 52b).

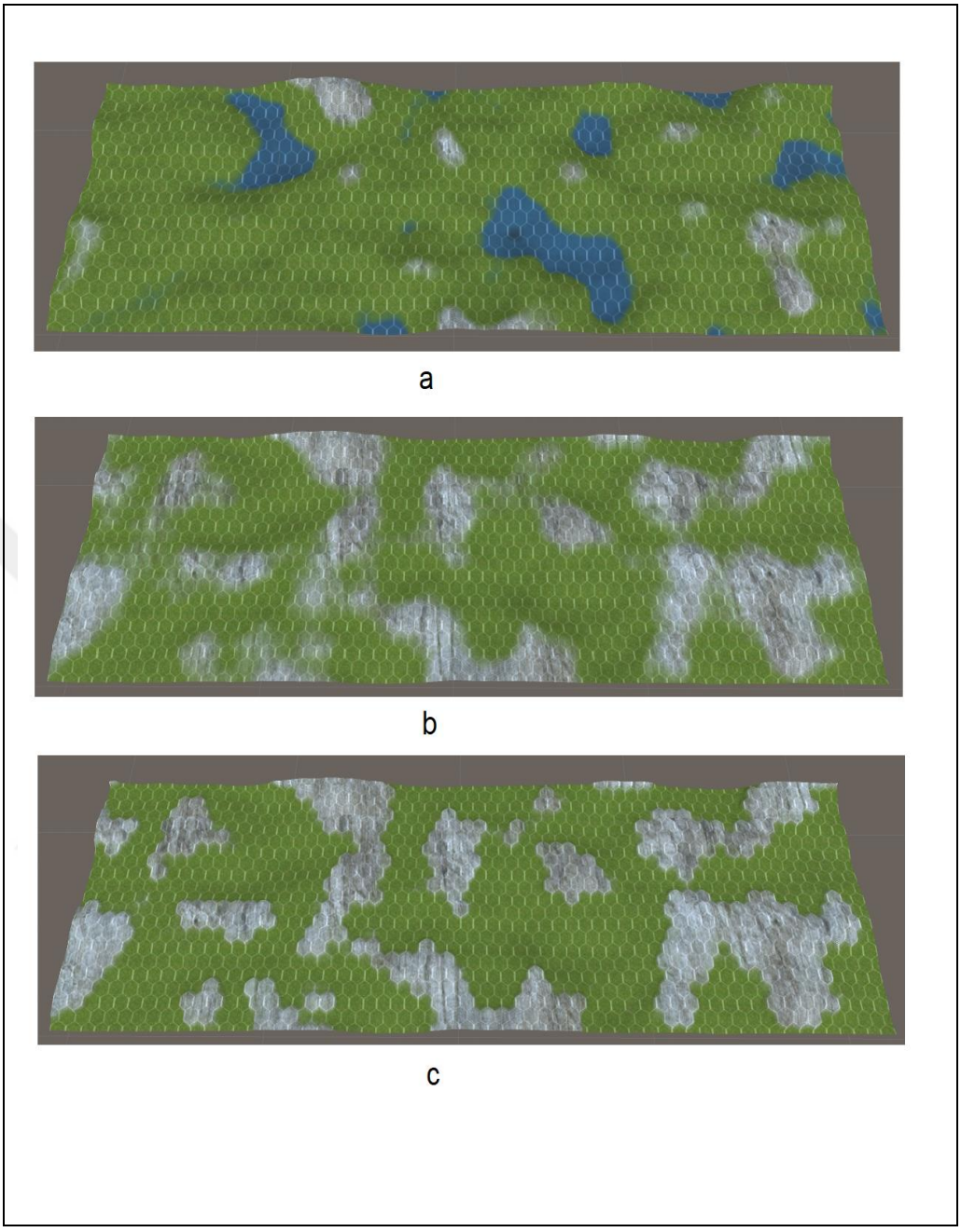


Figure 54. Map4 (a) Raw map, (b) Inland map, (c) Manipulated

7.1.5. Map5-Changing seed

In order to see the effect of pseudo-random gradient vectors (defined in Section 5.1), Map0 is used with no shift value with a seed of 1000 in Figure 55a. In Figure 55b, manipulation functions are enabled and map is converted to tiled form. In Figure 55c, seed value of 2401

is used. In Figure 55d, manipulation functions are enabled and the map in Figure 55c is converted to tiled form.

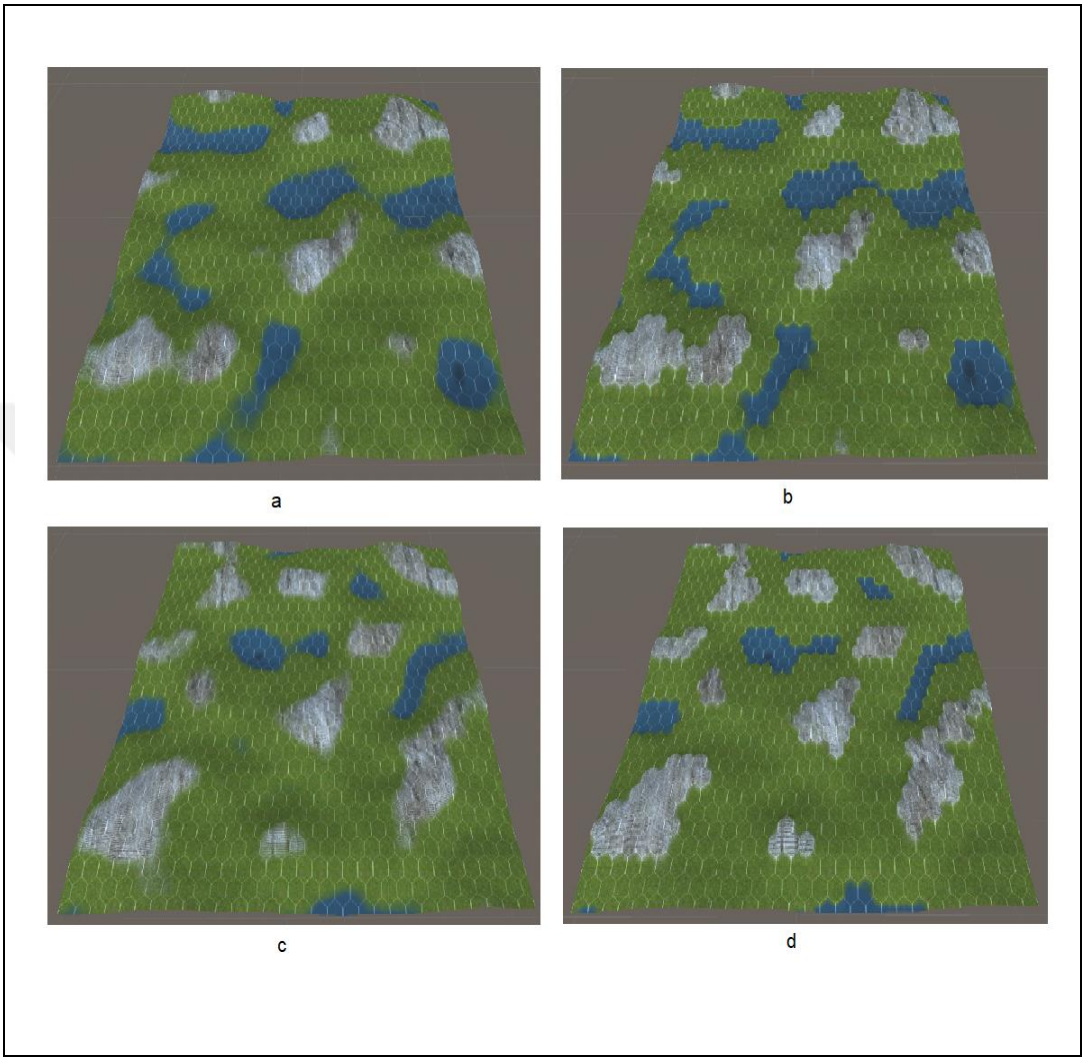


Figure 55. Seed (a) raw map with seed=1000, (b) manipulated map with seed=1000, (c) raw map with seed=2401, (d) manipulated map with seed=2401

7.2. NOISE PARAMETERS

Seed is the randomness, noiseScale is the zoom level and offset values are the shift values of Perlin function. In the noise function all values are between 0 and 1. These values are multiplied by a constant (maximumHeight) value to increase the height of the map.

7.2.1. Map6, Map7-Changing Frequency

Noise Scale represent the frequency of Perlin Noise. Different noise frequencies can be observed in Figure 56. Low frequencies (high noise Scale) behave as if it is zoomed into the map (Figure 56a), high frequencies (low noise Scale) behave as if it is zoomed out of the map (Figure56b). If mountain peaks or sea dips are smaller than the size of one tile, those mountain or sea regions will be lost after vertex manipulation. This is presented in Figure 57. If frequency is too low, mesh will be seen like a terrain rather than a map. Map6 is presented in Figure 56a. Map7 is presented in Figure 56b.

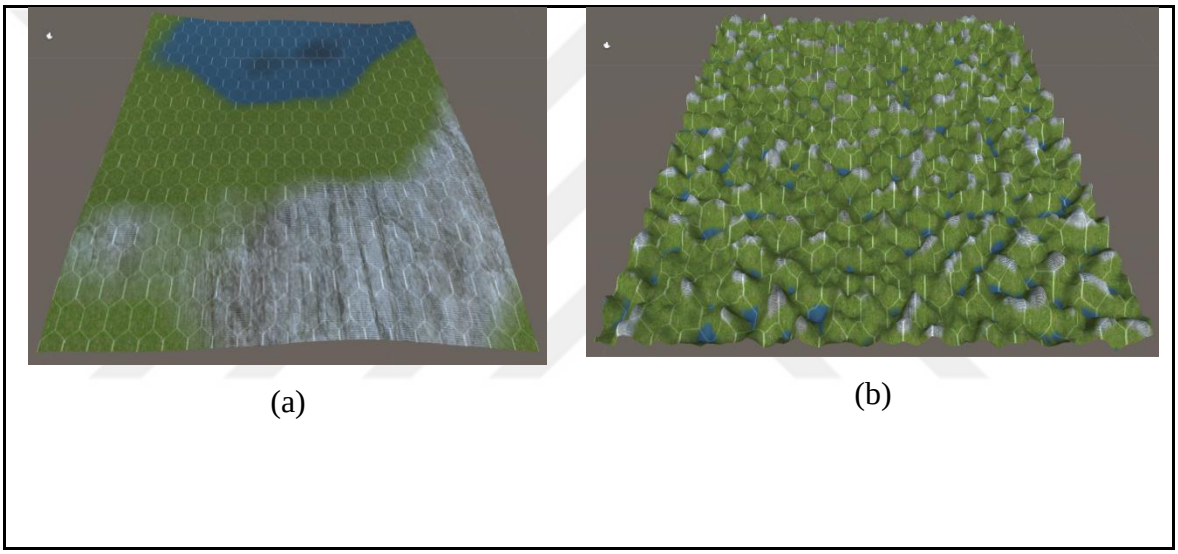


Figure 56. Noise Scale (a) 100, (b) 5

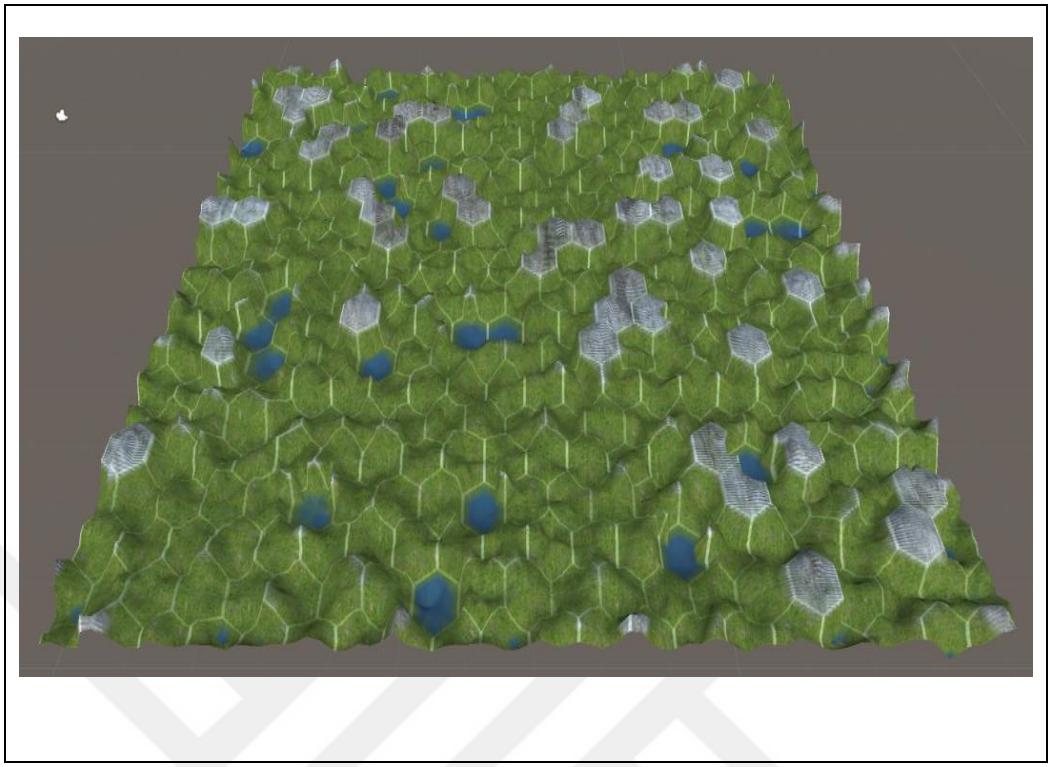
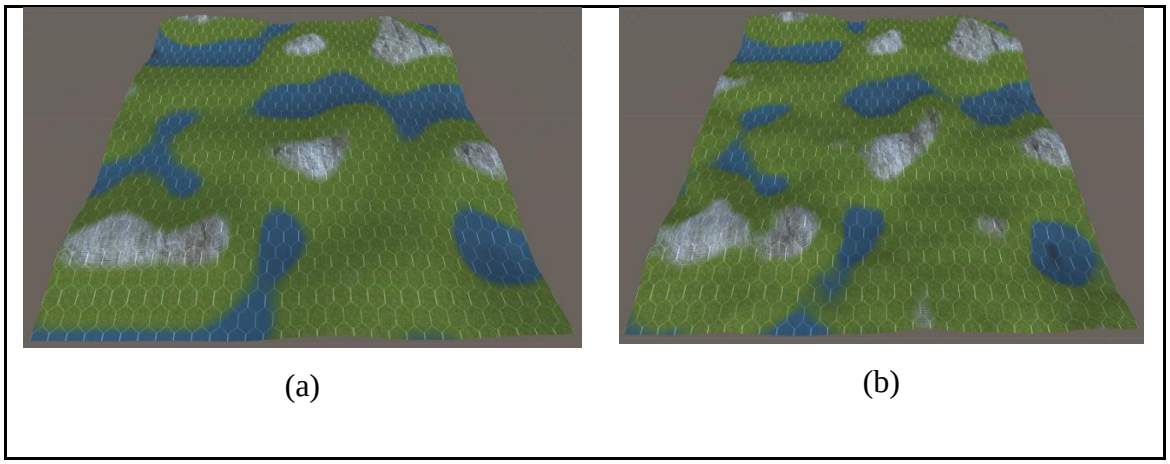


Figure 57. After Manipulation of Figure 44b

7.2.2. Map8-Changing Octaves

Octaves represent the number of different signals. High octaves increase the variety of noise. A larger map will be used in order to see the variety better. 30x30 dimensions are used in Figure 58. In Figure 58a, there are 5 sea regions, but in Figure 58b, there are 9 sea regions.

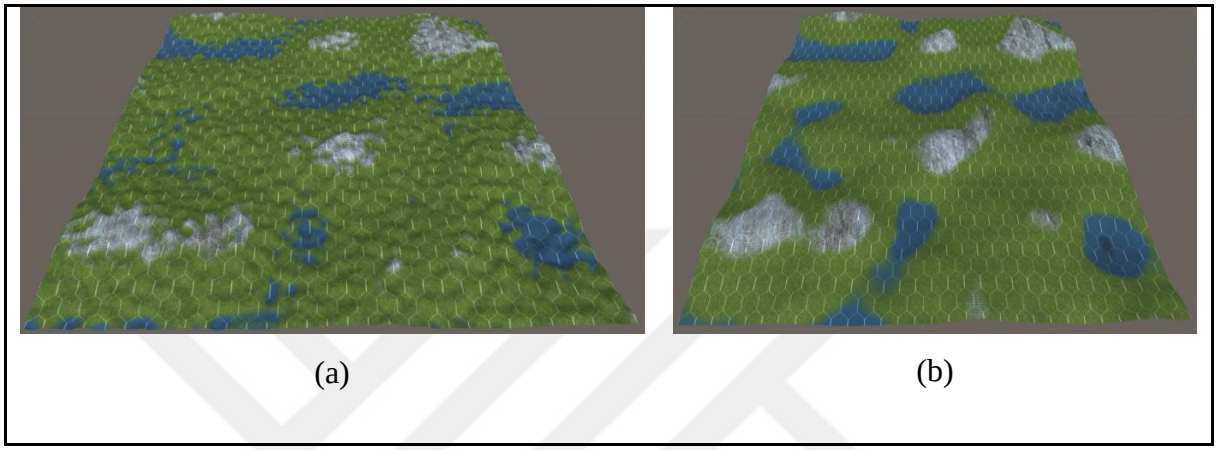


(a) (b)

Figure 58. Octaves (a) 1, (b) 10

7.2.3. Map9-Changing Lacunarity

Lacunarity is the frequency multiplier between the noise signals. As lacunarity increase, oscillation of varieties increases. Additional octaves have oscillational effects on the main octave as seen in Figure 59. Advised value for lacunarity is 2 as in Figure 59b.

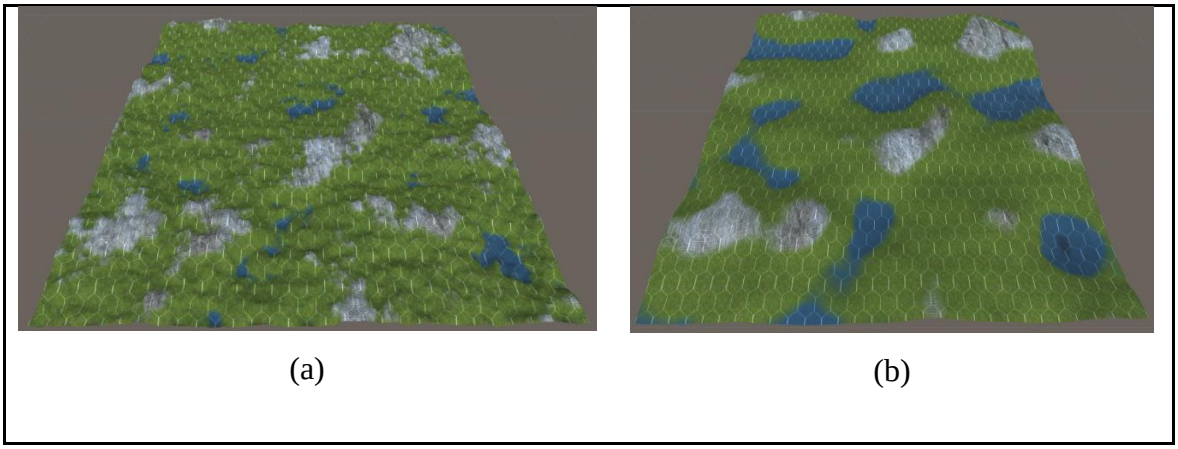


(a) (b)

Figure 59. Lacunarity (a) 8, (b) 2

7.2.4. Map10-Changing Persistence

Persistence is the effect of additional octaves. If persistence increases, details will be more effective. The soft appearance of Perlin is completely lost with 0.7 persistence as seen in Figure 60a. Map0 is presented in Figure 60b for comparison.



(a) (b)

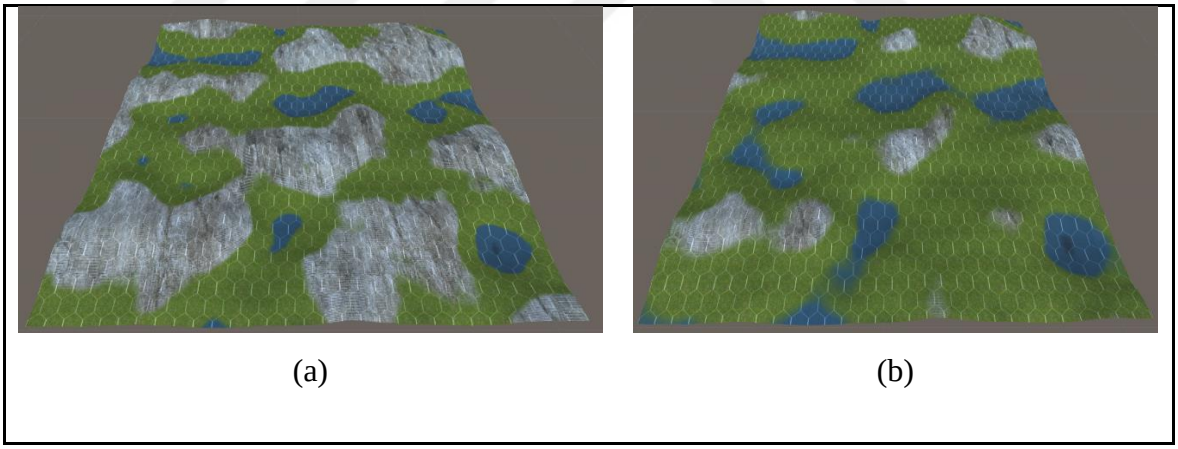
Figure 60. Persistence (a) 0.7, (b) 0.3

7.3. MAP PARAMETERS

Map can be re-generated with the same Perlin noise parameters but with different map parameters. For the previous maps, mountain threshold was 0.7, sea threshold was 0.3, map dimensions were 10x10 and maximum elevation was 10. Here 30x30 maps are used to observe the affects better.

7.3.1. Map11-Mountain Threshold

If mountain threshold is increased, the number of mountain tiles will decrease. There would be less tiles having higher height value than mountain threshold value. If mountain threshold is decreased, the number of mountain tiles will increase. There would be more tiles having higher height value than mountain threshold value. The effect of mountain threshold can be observed in Figures 61a and 61b.

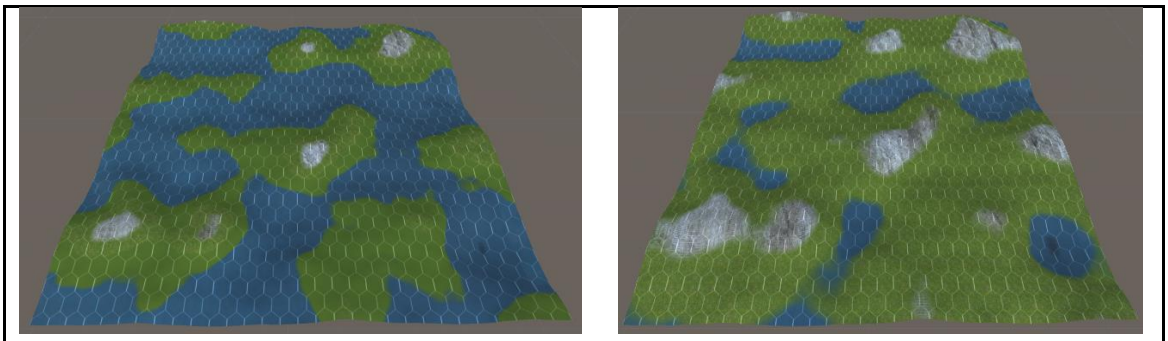


(a) (b)

Figure 61. Mountain Threshold (a) 0.5, (b) 0.7

7.3.2. Map12-Sea Threshold

If sea threshold is increased, the number of mountain tiles will increase. There would be less tiles having lower height value than sea threshold value. If sea threshold is decreased, the number of mountain tiles will decrease. There would be less tiles having lower height value than mountain threshold. Effect of sea threshold can be observed in Figures 62a and 62b.



(a) (b)

Figure 62. Sea Threshold (a) 0.5, (b) 0.3

7.3.3. Map13-Map Size

Maps can be generated in any size. A different map size can be observed in Figure 63. As the map in Figure 63 is larger than the previous maps, camera is zoomed out to see the complete map. When the camera is so far, it is hard to detect the height differences. Performance according to maps size is explained in detail in Section 7.4.4.

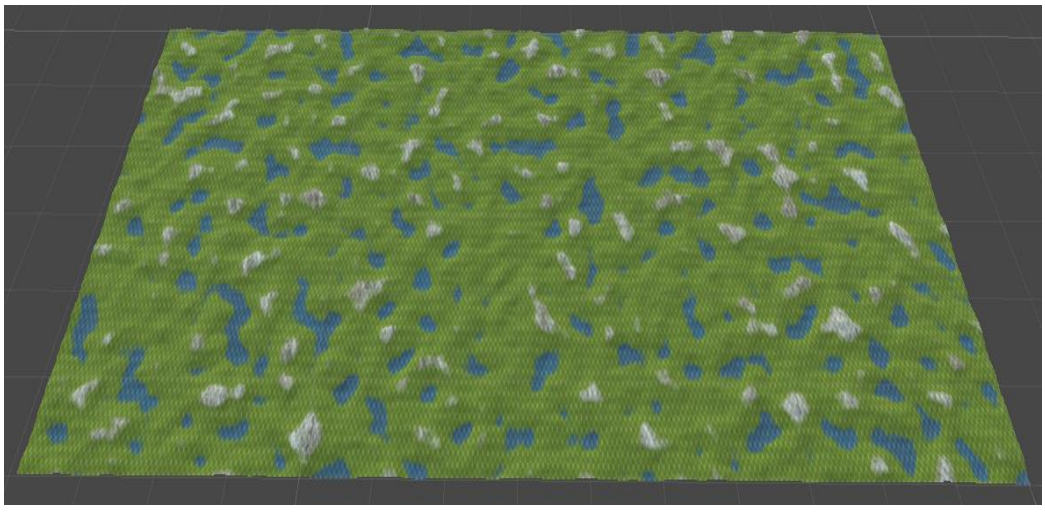
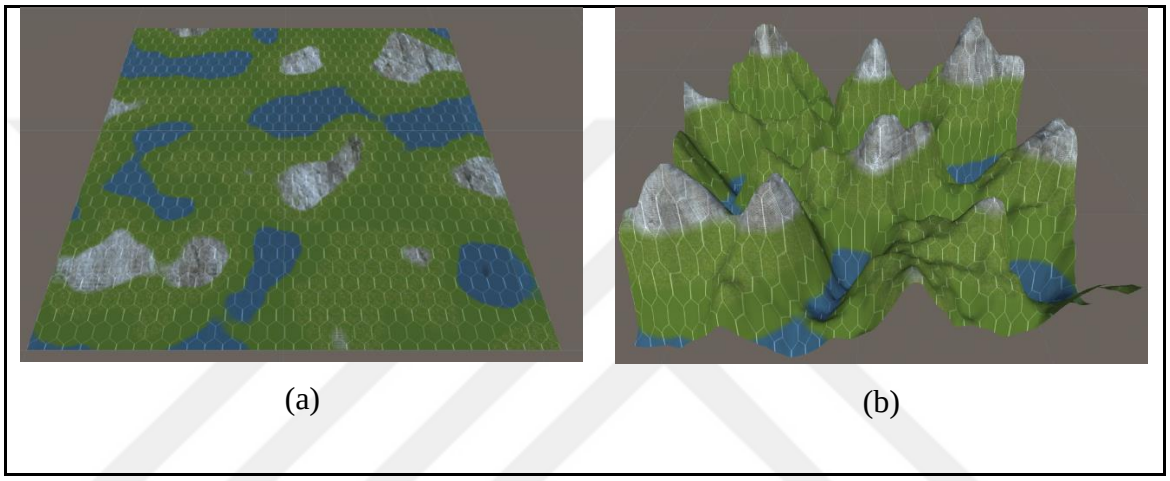


Figure 63. 200x100 map

7.3.4. Map14, Map15-Maximum Elevation

Maximum elevation shows the largest height value in the map. All elevation values between 0 and 1, are multiplied by maximum elevation. If maximum elevation value is 0, map will be 2D, because all height values would be 0. It does not affect tile definitions because thresholds are also multiplied by maximum elevation. Effect of max elevation can be observed in Figures 64a and 64b.



(a) (b)

Figure 64. Maximum Elevation (a) 0.1, (b) 100

7.4. EVALUATION OF MAPS 1-15

Evaluation of changing Perlin noise parameters, general noise parameters and map parameters are presented in this section.

7.4.1. Run-Time Performance

Run-time performance of all maps are presented in Table 5. Time does not change significantly while map parameters are changing. Map 13 has more processing time than other maps as it is larger than other maps. Time performance according to map size is analyzed in Section 7.5.4.

Table 5. Run-Time Performance of Maps1-15

	Geometric Values				Basic PTG		Manipulation Function		
TIME	x	z	Hex	Verti-ces	Time (ms)	Time per hex (ms)	Time (ms)	Time per hex (ms)	Func-tion Over-head (%)
Map0	30	30	900	29161	161	0,18	19	0,02	11,8
Map1'	30	30	900	29161	162	0,18	19	0,02	11,7
Map1	30	30	900	29161	170	0,19	21	0,02	12,4
Map2'	40	20	800	26001	156	0,2	21	0,03	13,5
Map2	40	20	800	26001	162	0,2	21	0,03	13
Map3'	20	40	800	25921	159	0,2	20	0,03	12,6
Map3	20	40	800	25921	162	0,2	20	0,03	12,3
Map4'	60	20	1200	38961	192	0,16	30	0,03	15,6
Map4	60	20	1200	38961	200	0,17	26	0,02	13
Map5	30	30	900	29161	163	0,18	19	0,02	11,7
Map6	30	30	900	29161	166	0,18	22	0,02	13,3
Map7	30	30	900	29161	172	0,19	26	0,03	15,1
Map8	30	30	900	29161	141	0,16	23	0,03	16,3
Map9	30	30	900	29161	183	0,2	25	0,03	13,7
Map10	30	30	900	29161	167	0,19	23	0,03	13,8
Map11	30	30	900	29161	165	0,18	16	0,02	9,7
Map12	30	30	900	29161	160	0,18	21	0,02	13,1
Map13	200	100	20000	642001	2200	0,11	622	0,03	28,3
Map14	30	30	900	29161	167	0,19	22	0,02	13,2
Map15	30	30	900	29161	162	0,18	21	0,02	13

7.4.2. Sharpness

Sharpness values of all maps are presented in Table 6. Aim of these tests are to observe the effects of the manipulation functions on the sharpness of the total mesh. So, sharpness is measured for both non-manipulated map (raw map) and manipulated map. Both maximum sharpness and average sharpness at manipulated vertices are measured. Number of manipulated vertices are also measured. But this measured average sharpness value does not show the total sharpness of the mesh. Because average sharpness is the average sharpness value of just manipulated vertices. In order to calculate mesh average sharpness, weighted average formula should be used. In a weighted average, each data point is multiplied by the

assigned weight, which is then summed and divided by the number of data points. Average sharpness can be calculated according to equation in 7.3. s is sharpness of mesh, *v* is number of total vertices, *v1* is number of manipulated vertices, *s2* is the measured average sharpness of the raw map, *s1* is the measured average sharpness of manipulated map.

$$s = \frac{v1 * s1 + (v - v1) * s2}{v} \tag{7.3}$$

Table 6. Sharpness of Maps1-15

				Raw Map		Manipulated map			
CURVA-TURE	**x**	**z**	**Verti-ces (v)**	**Max-imum (%)**	**Aver-age (s2) (%)**	**Max-imum (%)**	**Aver-age (s1) (%)**	**Number of manipu-lated vertices (v1)**	**Sharp-ness of mesh (%) (s)**
Map0	30	30	29161	0,55	0,12	22,5	4,92	2770	0,58
Map1'	30	30	29161	0,52	0,1	19,8	4,33	2394	0,45
Map1	30	30	29161	0,64	0,09	25,2	4,5	2900	0,53
Map2'	40	20	26001	0,54	0,18	24,4	4,64	2659	0,64
Map2	40	20	26001	0,43	0,09	21,4	4,92	1800	0,42
Map3'	20	40	25921	0,59	0,1	23,5	4,38	2357	0,49
Map3	20	40	25921	5,25	0,25	71,5	6,55	2034	0,74
Map4'	60	20	38961	0,47	0,1	19,8	4,32	2282	0,35
Map4	60	20	38961	0,29	0,05	15,5	3,71	4168	0,44
Map5	30	30	29161	0,52	0,1	20,5	4,6	2858	0,54
Map6	30	30	29161	0,23	0,05	13,9	3,87	1431	0,24
Map7	30	30	29161	6,44	1,57	82,9	12,4	7516	4,37
Map8	30	30	29161	0,31	0,08	18,7	4,52	2528	0,46
Map9	30	30	29161	2,26	0,5	30,3	4,65	3904	1,06
Map10	30	30	29161	3,52	0,8	37,2	5,94	4709	1,63
Map11	30	30	29161	0,5	0,11	20,4	4,3	3613	0,63
Map12	30	30	29161	0,55	0,1	22,9	4,75	3110	0,60
Map13	200	100	642001	0,54	0,09	20,9	4,13	50565	0,41
Map14	30	30	29161	0,55	0,11	22,5	4,92	2770	0,57
Map15	30	30	29161	0,55	0,11	22,5	4,92	2770	0,57

Map3, Map9, Map10 have average sharpness values greater than 0.2. Map3 is sharper because, it has increased Perlin noise effect on west side of the map. Map9 and Map10 have

more details because of increased lacunarity and persistence. Map6 has low sharpness value because of low frequency of map, Map7 has high sharpness value because of high frequency of the map.

Result 1: More detailed maps have higher sharpness values.

There is a peak value in Map7; 82.9%. Frequency of the map is so high that, there is both mountain vertices and sea vertices in the same tile. Hex unity algorithm increases sharpness here. If there are both sea vertices (vertices in the sea tile type range) and mountain vertices (vertices in the mountain tile type range), there is fast transition in the tile. Mountain neighbor of the tile convert the border vertices to mountain height range and sea neighbor of the tile convert the border vertices to sea height range. This creates high sharpness in the tile.

In quantization, data smaller than quantization amount can be lost. In this application, tile size is the quantization step. If a region belonging to any tile type is smaller than the tile size, that region can be lost due to hex unity algorithm, because according to hex unity algorithm, all vertices in a tile must belong to same tile type range. This is also defined in Section 7.2.1

Result 2: Regions defined by the noise must not be smaller than the quantization step size; tile size.

Effect of linear interpolation algorithm is analyzed in Section 7.5.5.

7.5. EVALUATION OF THE REQUIREMENTS

7.5.1. Real 3D

In Section 4.1 five requirements are defined. The first requirement is to generate a 3D mesh. As our approach uses PTG, the mesh is formed with different vertex heights. This is presented in Figure 41. All mountain (or land or sea) tiles have different shapes, as each tile consist of 48 vertices which have different heights. Therefore, variations of tile types increase (as described in Section 4.1.1).

7.5.2. Hexagonal Tiles

All tiles are hexagonal. Hexagonal tiles can be observed in Figure 65. Vertices of hexes are presented in Figure 65a. Tiles are presented in Figure 65b. All hexes are uniform.

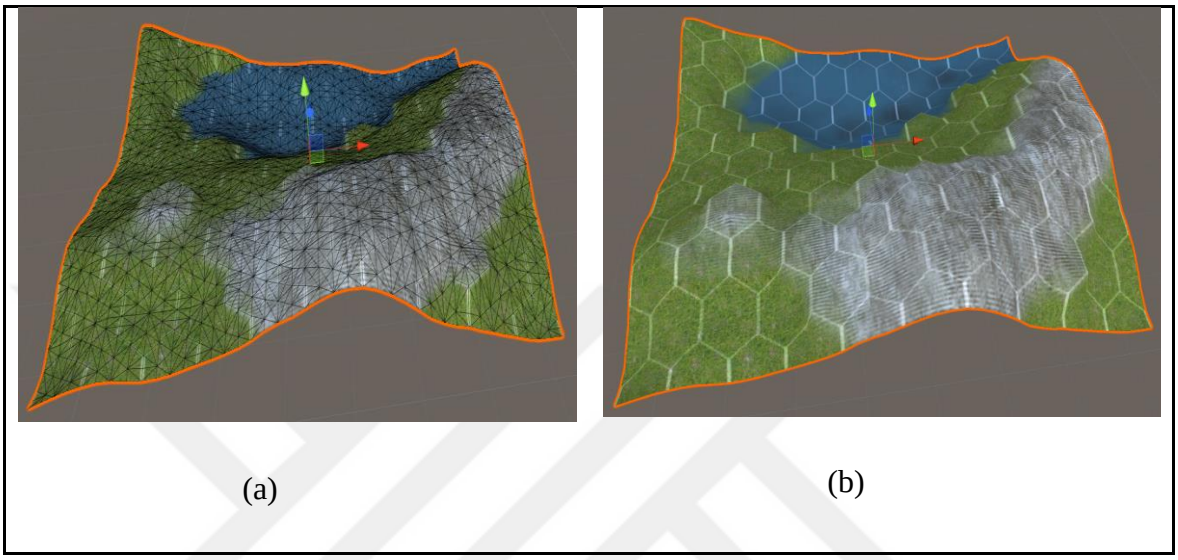
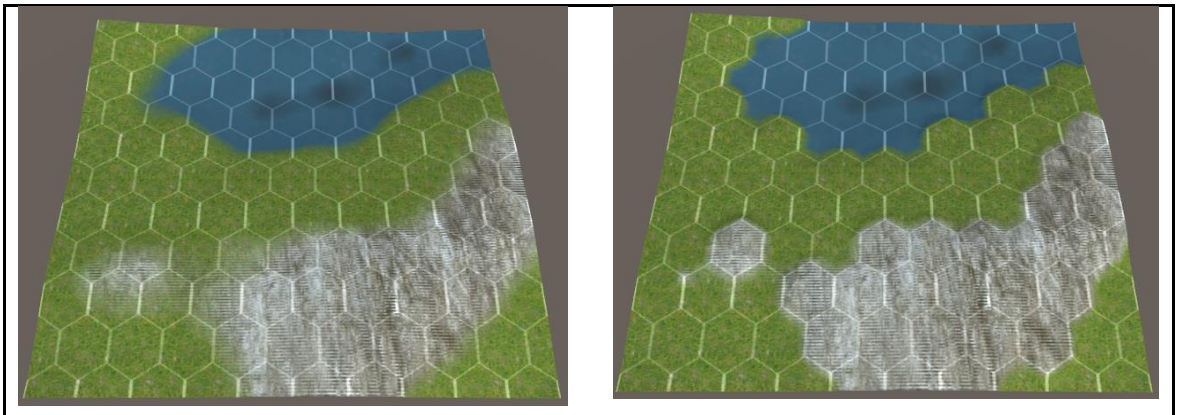


Figure 65. 3D Map (a) Vertices, (b) Tiles

7.5.3. Easily Recognizable

In Section 4.1.3, it was defined that in order to make the tiles easily recognizable, all vertices except border vertices of the tile must belong to the same terrain type. Tiles are easily recognizable in Figure 66b compared to Figure 66a. As all inner vertices of a tile belong to the defined height range, ER is succeeded. In Figure 67, it is presented that all inner vertices of a hex belong to the same height range.



(a) (b)

Figure 66. (a) Raw Map, (b) Tiled Map

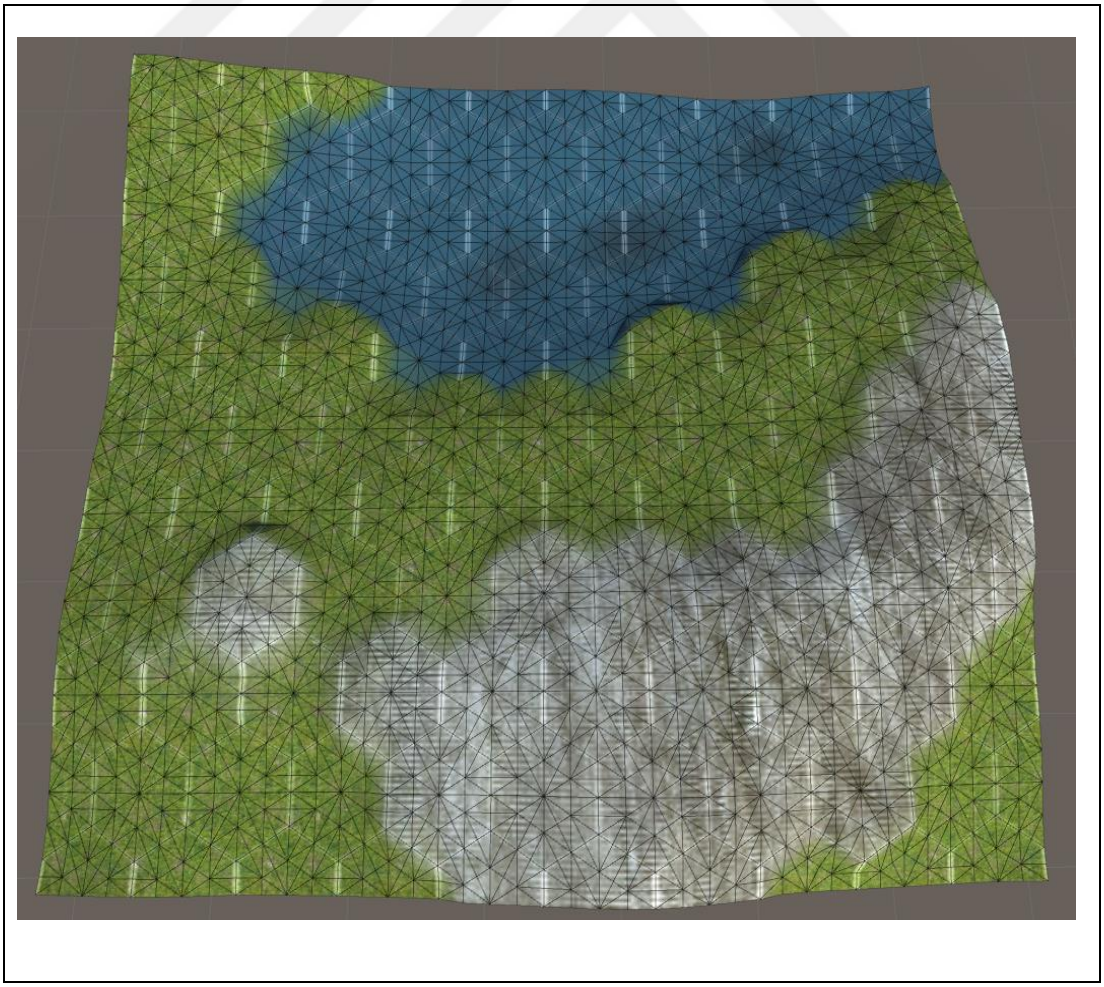


Figure 67. Vertices of the map

7.5.4. Run-time Performance

The time value is read at four different level. In order to calculate total processing time, time is taken at the beginning and at the end of the whole process. In order to calculate manipulation functions time, time is taken at the beginning and at the end of the manipulation functions. “System.DateTime.Now” function is used for time measurements. When measuring time, 10 measurements are taken. Average of 10 values are used in the table for the resulting number. Basic PTG is calculated by subtracting manipulation time from the whole process time.

The program runs on a PC with Intel i7 processor and 16GB RAM and NVIDIA GTX960m graphic card. Manipulation functions consist of vertex manipulation, linear interpolation and edge/corner correction functions. The results are given in Table 3. x and z are the number of tiles in x and z directions. Total number of vertices and triangles are also shown in the Table 7 and Table 8.

Table 7 shows run time performance for smaller maps. “Hex” is the total number of hexes in the mesh. It is the product of x and z. “Vertices” are the total number of vertices used in the mesh. “Triangles” are number of triangles generated (as defined in Section 3.6) in the mesh. “Time” is the calculated value of basic PTG and measured value of manipulation function. “Time per hex” is calculated by dividing time to hex number. “Function overhead” is the percentage of manipulation function time to basic PTG function time.

Table 7. Run-Time Performance for Small Maps

Geometric Values					Basic PTG		Manipulation Function		
x	**z**	**Hex**	**Vertices**	**Triangles**	**Time (ms)**	**Time per hex (ms)**	**Time (ms)**	**Time per hex (ms)**	**Function Overhead (%)**
2	2	4	153	384	57	14,25	2	0.50	3.51
6	6	36	1225	3456	59	1,64	3	0.08	5.08
10	6	60	2025	5760	63	1,05	3	0.05	4.76
16	8	128	4257	12288	70	0.55	4	0.03	5.71
20	10	200	6601	19200	77	0.39	5	0.03	6.49
24	12	288	9457	27648	88	0.31	6	0.02	6.82
30	16	480	15665	46080	105	0.22	9	0.02	8.57
36	20	720	23409	69120	130	0.18	14	0.02	10.77
40	20	800	26001	76800	139	0.17	13	0.02	9.35
48	24	1152	37345	110592	173	0.15	19	0.02	10.98
60	30	1800	58201	172800	228	0.13	35	0.02	15.35

Table 8 shows time performance for larger maps. 32-bit vertex buffer used instead 16-bit vertex buffer for larger maps which is a new function in Unity. Second is used instead of ms as larger maps takes more time to generate.

Table 8. Run-Time Performance for Larger Maps

Geometric Values					Basic PTG		Our Approach		
x	**z**	**Hex**	**Vertices**	**Triangles**	**Time (s)**	**Time per hex (ms)**	**Time (s)**	**Time per hex (ms)**	**Function Overhead (%)**
80	40	3,200	103,201	307,200	0.37	0.12	0.10	0.03	25.79
100	50	5,000	161,001	480,000	0.56	0.11	0.14	0.03	25.72
200	100	20,000	642,001	1,920,000	2.01	0.10	0.57	0.03	28.51
400	200	80,000	2,564,001	7,680,000	7.94	0.10	2.34	0.03	29.46
500	250	125,000	4,005,001	12,000,000	12.24	0.10	3.56	0.03	29.09
600	300	180,000	5,766,001	17,280,000	20.80	0.12	6.30	0.04	30.30
800	400	320,000	10,248,001	30,720,000	33.41	0.10	10.10	0.03	30.25
1.000	500	500,000	16,010,001	48,000,000	50.19	0.10	15.06	0.03	30.02

Function overhead vs Hex number graphics can be observed in Figure 68 where Figure 68a is the graph for smaller maps and Figure 68b is the graph for larger maps. Function convergence can be observed better in larger maps.

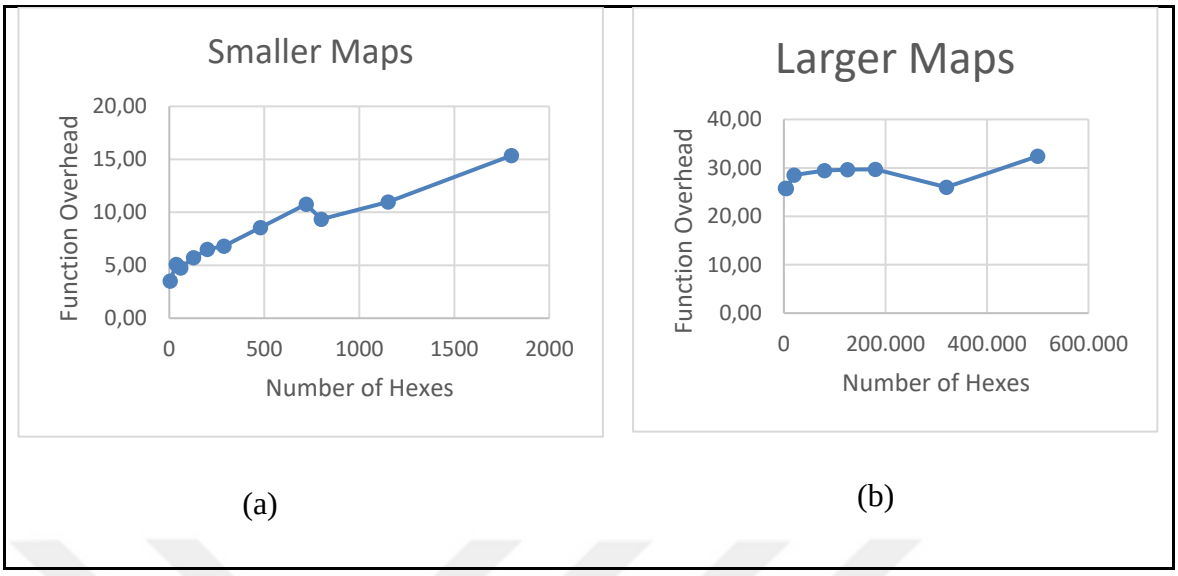


Figure 68. Function Overhead vs Hex Number (a) Smaller Maps, (b) Larger Maps

Time per hex of basic PTG is almost constant (0.1) after 20,000 vertices. Time per hex of manipulation functions are almost constant (0.03) after 4,000 vertices. It can be said that function overhead converges to a number around 30% (100*0.03/0.1).

Result 1: Manipulation function time does not increase so much compared to basic PTG time even at large number of vertices.

By looking at "time per hex" column, it is seen that "time per hex" decreases as the number of hexes increase. Total processing time is equal to "time per hex" multiplied by the number of hexes. When the tiles of a map is generated one by one, the total time of this process is smaller than the total time of a map generated as a whole. Because "time per hex" value of larger maps is smaller than "time per hex" value of smaller maps.

Result 2: It is proven that as tiles are generated together, total processing time decreases.

7.5.4.1. Performance using map sizes of TBS games

Performance of our approach is also measured using the dimensions of the Civilization maps which is the most popular games explained in Chapter 2. Civilization 6 is the last version of civilization series. Map dimensions of Civilization 6 are presented in Table 9 [24].

Table 9. Civilization 6 Map Dimensions

	Geometric Values					Basic PTG		Our Approach		
	x	z	Hex	Vertices	Triangles	Time (ms)	Time per hex (ms)	Time (ms)	Time per hex (ms)	Function Overhead (%)
Duel	44	26	1,144	37,065	109,824	159.0	0.14	32.10	0.03	20.19
Tiny	60	38	2,280	73,593	218,880	299.4	0.13	63.50	0.03	21.21
Small	74	46	3,404	109,705	326,784	390.6	0.11	102.40	0.03	26.22
Standard	84	54	4,536	146,041	435,456	515.8	0.11	123.40	0.03	23.92
Large	96	60	5,760	185,329	552,960	633.3	0.11	175.40	0.03	27.70
Huge	106	66	6,996	224,985	671,616	749.9	0.11	202.00	0.03	26.94

According to Table 9, largest map dimension of civilization 6 can be generated around 1 second. As loading time of map in civilization is in range of minutes, this time is an extremely short time. The map editor of civilization 6 is opened, generated the first map. Then map generation time is measured to prevent time loss for the first opening. A small map takes 10sec to be generated and a huge map takes 30sec to be generated. Even the generated maps are 2D, it takes much more time than our application.

Result 3: Using PTG for generating tiled-maps is a fast method.

7.5.5. Sharpness

In Figure 27, map has very high sharp values in the shores. At the shores, color is gray which is 128 where 255 is the highest point (white) and 0 with the lowest point (black). Linear interpolation tolerance is 128 as there is difference level of 128 between 2 vertices at the shoreline instead more smaller numbers. If the sharpness of a vertex at the shore line is calculated according to equation (4.1), it would be 128 (height of shore line land) – 0 (height of sea) – [128 (height of shore line land) – 128 (height of shore line land)] = 128. With a maximum height of 255, shore line vertices have 50% sharpness.

Result 1: If the linear interpolation tolerance is increased, sharpness increases.

If the linear interpolation is decreased, blending value should be also decreased. Linear interpolation increases border (sea-land or land-mountain) detection while increasing

blending decreases border detection. So, different sharpness values should be tested with different blend and linear interpolation values.

Figure 69 shows our manipulated vertices that are considered for sharpness measurement. Sharpness is just calculated at manipulated vertices. As height of other vertices do not change, their sharpness values do not change. There are 322 sharpness vertices.

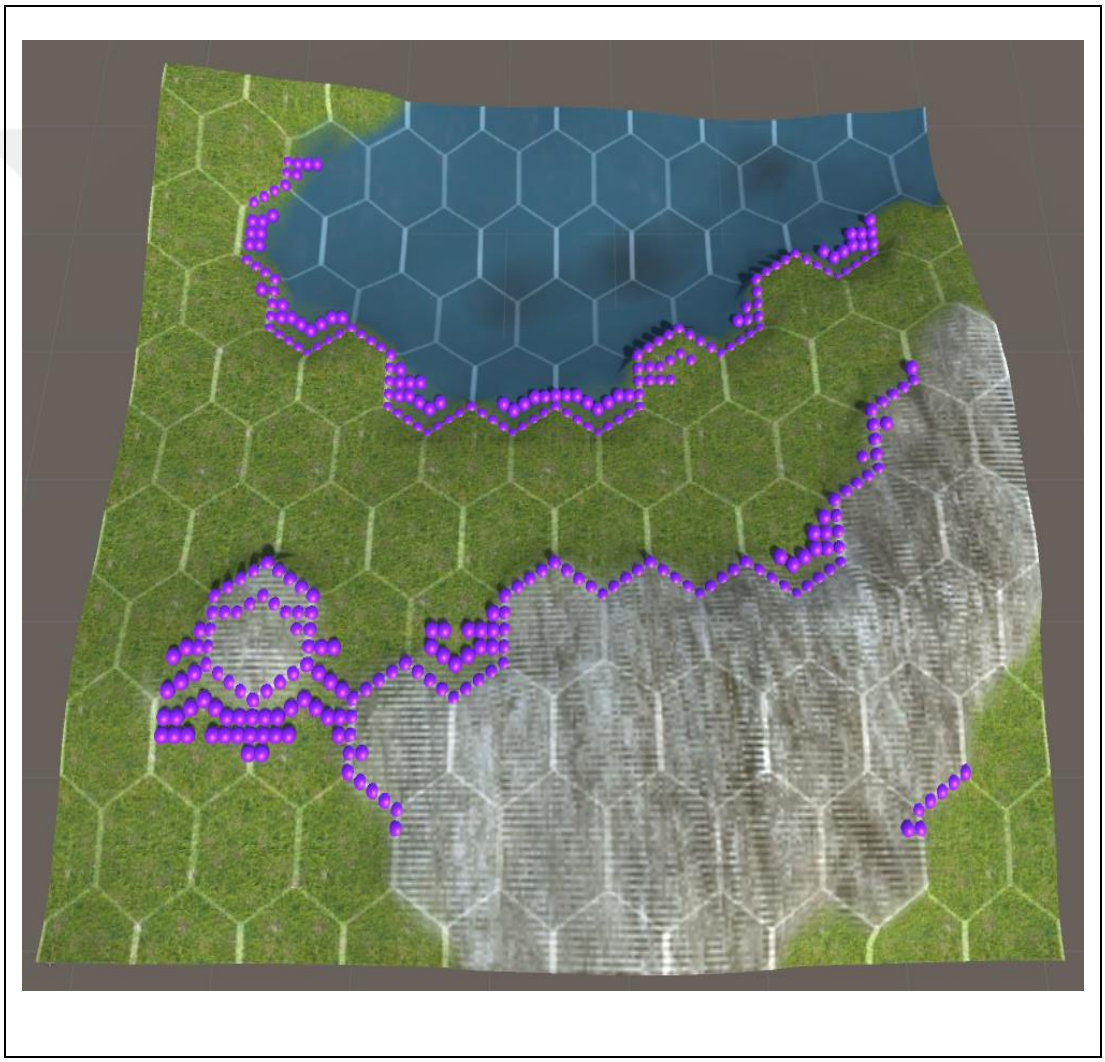


Figure 69. Vertices that sharpness is measured

First sharpness measurement is done with 0.1 blending value and 0.03 linear interpolation tolerance constant. Map generated with these parameters is presented in Figure 70.

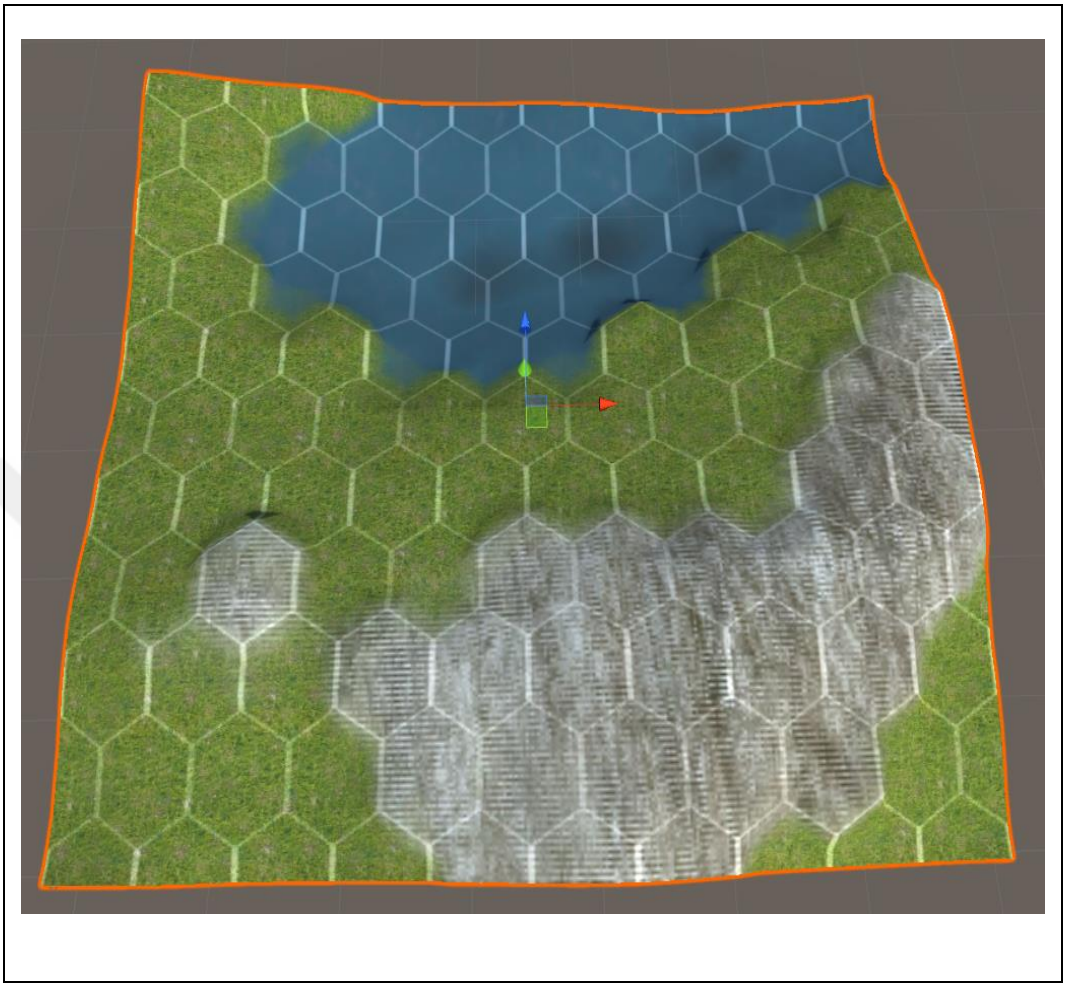


Figure 70. Map with 0.03 linear interpolation, 0.1 blending

In table 10 maximum sharpness of manipulated vertices is 19,8%. Average sharpness of manipulated vertices is 4,7%.

Table 10. Maximum and average sharpness values of 0.03 linear interpolation, 0.1 blending

	Max Height = 10		Percentage	
	Average Sharpness	Maximum Sharpness	Average Sharpness	Maximum Sharpness
Before manipulation	0.0136	0.0485	0.136	0.485
After manipulation	0.4664	1.9776	4.664	19.776

Second sharpness measurement is done 0.005 blending value and 0.03 linear interpolation tolerance constant. Map generated with these parameters is presented in Figure 71.

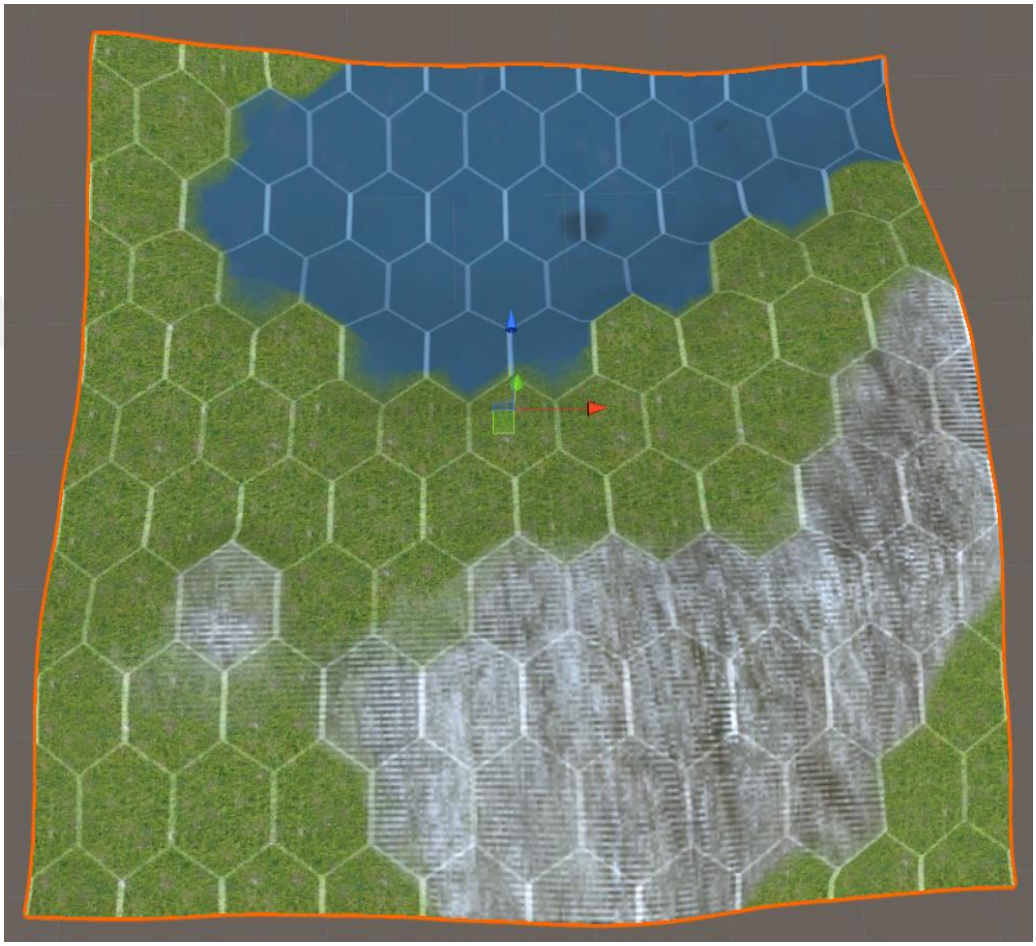


Figure 71. Map with 0.005 linear interpolation, 0.03 blending

Table 11. Maximum and average sharpness values of 0.005 linear interpolation, 0.03 blending

	Max Height = 10		Percentage	
	Average Sharpness	Maximum Sharpness	Average Sharpness	Maximum Sharpness
Before manipulation	0.0136	0.0485	0.136	0.485
After manipulation	0.3915	1.6761	3.915	16.761

In table 11, maximum sharpness of manipulated vertices is 16,8%. Average sharpness of manipulated vertices is 3,9%.

Although average sharpness value decreases from 4,7% to 3,9% some appearance problems occurred. Easy recognition of tile has some appearance problem as seen in Figure 60. Decreasing linear interpolation, decreases its affect. If linear interpolation tolerance decreases too much, it is hard to distinguish sea land transitions or land mountain transitions. This can be observed in Figure 72.

Result 2: Changing linear interpolation value did not affect sharpness value much. It can be decreased until tile unity loses clearness.

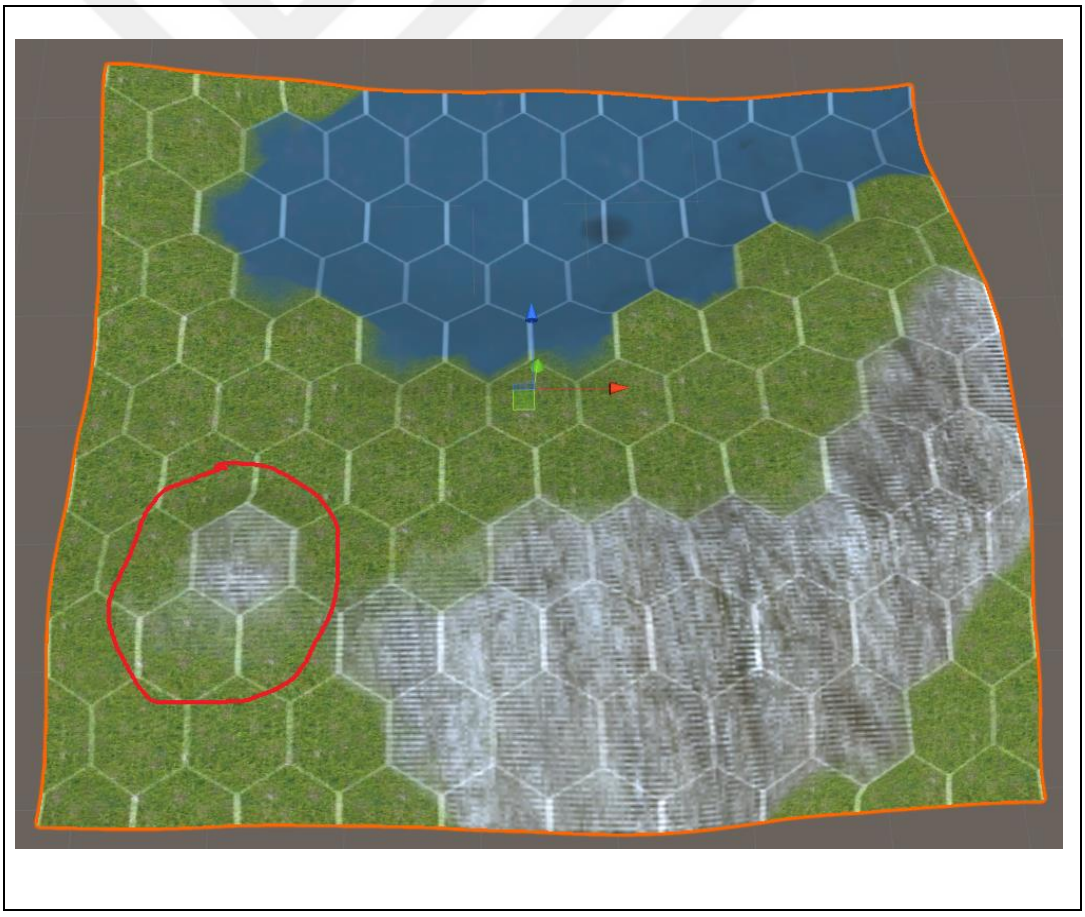


Figure 72. Tile unity start to lose clearness.

7.5.5.1. Sharpness comparison

In Chapter 2, four games are taken into consideration. Civilization uses predefined tiles. Predefined tiles are not generated by using procedural terrain generation methods. Anno series are designed as 2D; and 3D predefined mountains are used. Sharpness is defined for 3D maps, as sharpness of a 2D map is always zero. In Thea 2, tiles are not modified for easy recognition. Therefore, they cannot be considered for sharpness comparison.

In Endless Legend, tiles are 3D and tiles are modified for easy recognition. Two neighboring tiles are marked with blue circles in Figure 73. Three vertices are marked with red following each other. Left tile has a height of 0.5 and right tile has a height of 1. Tile centers are flat. If curvature at middle vertex is calculated by using the equation (4.1), it is (1(height of second vertex)-0.5(height of first vertex)) – (1(height of third vertex)-1(height of second vertex)) = 0.5. There is 50% curvature in the middle vertex. This value is a larger value than the maximum curvature (19%) in our application.

Curvature value does not make a map acceptable or unacceptable. It just shows the sharpness of the map.

Figure 73. Sharpness in Endless Legend

7.5.6. Evaluation

All 5 requirements are done. Real 3D, hexagonal, easily recognizable requirements are done depending on their definitions. Detailed tests are done for run-time performance and sharpness. Some results are developed according to tests. Target of “a procedure for PTG to support tiled maps” is succeeded.

8. CONCLUSION

A method to generate 3D hexagonal tiled terrains is defined using procedural terrain generation methods using Perlin noise. All our requirements for our approach are considered and compared with basic PTG method. Fast process time of PTG is not increased in multiplies of total process time. As 1 hexagon inside outer hexagon is used, total and peak sharpness values are increased in significant amount. But this sharpness increase amount can be decreased by using more hexagons inside the outer hexagon.

The application is developed in Unity but no unity-specific tool (such as unity terrain tool) is used, application can be converted to OpenGL or other game engines.

This tiling process can be applied to any generated map as a filter after Cartesian vertices are converted to hexagonal vertices. As all real-world maps are drawn by pixels, cartesian form of real-world maps should be converted to our frame style. Vertices of our frame do not coincide with pixels of a real game worlds.

REFERENCES

1. Burdziej J. Using hexagonal grids and network analysis for spatial accessibility in urban environments-a case study of public amenities in Torun. *Miscellanea Geographica*. 2019; 23(2): 99-110.

2. Civilization V: Homepage. [cited 2023 January 22]. Available from: https://civilization.com/civilization-5/

3. Home - Anno Union. [cited 2023 January 22]. Available from: https://anno-union.com/

4. Amplitude Studios. [cited 2023 January 22]. Available from: https://www.amplitude-studios.com/#GamesEndlessDungeon

5. Thea 2: The Shattering – MuHa Games [cited 2023 January 22]. Available from: http://muhagames.com/projects/thea-2-the-shattering/

6. Togelius J, Kastbjerg E, Yannakis GN. What is procedural content generation? Mario on the borderline. *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. 2011; 1-6.

7. 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 (TOMM)*. 2013; 9(1): 1-22.

8. Freiknecht J, Effelsberg W. A survey on the procedural generation of virtual worlds. *Multimodal Technologies and Interaction*. 2017; 1(4): 27.

9. Smelik R. M, De Kraker K. J, Tutenel T, Bidarra R, Groenewegen SA. A survey of procedural methods for terrain modelling. *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*. 2009; 25-34.

10. Fournier A, Fussell D, Carpenter L. Computer rendering of stochastic models. *Communications of the ACM*. 1982; 25(6): 371-384.

11. Fortune S. Voronoi diagrams and Delaunay triangulations. *Computing in Euclidean Geometry*. 1995; 225-265.

12. Lagae A, Lefebvre S, Cook R, DeRose T, Drettakis G, Ebert D.S, Zwicker M. A survey of procedural noise functions. *Computer Graphics Forum*. Oxford, UK: Blackwell Publishing Ltd. 2010; 29(8): 2579-2600.

13. Perlin K. An image synthesizer. *ACM Siggraph Computer Graphics*. 1985; 19(3): 287-296.

14. Musgrave FK, Kolb CE, Mace RS. The synthesis and rendering of eroded fractal terrains. ACM Siggraph Computer Graphics. 1989; 23(3): 41-50.

15. Veenaadeeve NV, Chithra R, Manju M, Soman KP. Efficiency of GPU over CPU in geometric processing of 3D objects. *International Journal of Engineering & Technology*. 2013; 2(7): 34-45.

16. Kelly C. In Programming 2D games: CRC press; 2012. p. 303-305.

17. Hardy A, Mc Roberts D. A. K. Blend maps: enhanced terrain texturing. *Proceedings of the 2006 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*. 2006;(61-70).

18. Wang H. Games, logic and computers. *Computation, Logic, Philosophy*. 1990; 195-217.

19. Maung D, Shi Y, Crawfis R. Procedural textures using tilings with Perlin Noise. *2012 17th International Conference on Computer Games*. 2012; 60-65.

20. Maung D. Tile-based method for procedural content generation.Doctoral Thesis. The Ohio State University. 2016.

21. Grelsson D. Tile based procedural terrain generation in realtime. Bachelor Thesis, Blekinge Institute of Technology. 2014.

22. Yap P. Grid-based path-finding. *Conference of the Canadian Society for Computational Studies of Intelligence*.Springer Berlin HeidelBerg. 2002; 44-55.

23. Perlin K. Improving noise. *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniqu*es. 2002; 681-682.

24. Map (Civ6) – Civilization Wiki. [cited 2023 January 22]. Available from: https://civilization.fandom.com/wiki/Map_(Civ6).

25. Gürler F, Onbaşıoğlu E. Applying Perlin Noise on 3D hexagonal tiled maps. *2022 International Symposium on Multidisciplinary Studies and Innovatie Technologies.* 2022; 670-673.

APPENDIX A: SHARPNESS TEST VALUES

Data in the below tables are generated from our application. Values in column 2 and column 3 are coordinates of vertices. Values in column 4 and column 5 are generated with our application. Values are 100.000 multiples of real values. In column 6 and column 7, absolute values of data are divided by 100.000. In the last 2 lines maximum and average values of data are calculated. Column 2, 4, 6 are used for non-manipulated old values. Column 3, 5, 7 are used for manipulated new values.250th data has the largest sharpness value and line is marked in yellow.

Table 12. Sharpness values with 0.005 linear interpolation, 0.03 blending

Sequence	XCoordinate	ZCoordinate	Old_Curvature	New_Curvature	Absolute Value old	Absolute Value new
1	28	5	-188	0	0,00188	0
2	67	5	-1445	-8601	0,01445	0,08601
3	68	5	-600	0	0,006	0
4	24	6	2423	66746	0,02423	0,66746
5	25	6	2128	50501	0,02128	0,50501
6	26	6	2246	50293	0,02246	0,50293
7	27	6	1254	32492	0,01254	0,32492
8	28	6	-258	-21134	0,00258	0,21134
9	68	6	-2062	-51411	0,02062	0,51411
10	69	6	-1384	2459	0,01384	0,02459
11	70	6	85	24508	0,00085	0,24508
12	71	6	1092	48683	0,01092	0,48683
13	72	6	2504	84746	0,02504	0,84746
14	16	7	142	23335	0,00142	0,23335
15	17	7	1448	30148	0,01448	0,30148
16	24	7	-129	0	0,00129	0
17	25	7	398	18815	0,00398	0,18815
18	9	8	206	25782	0,00206	0,25782
19	10	8	-300	19579	0,003	0,19579
20	11	8	-462	22355	0,00462	0,22355
21	13	8	-50	21671	0,0005	0,21671
22	14	8	217	13348	0,00217	0,13348
23	15	8	-135	8504	0,00135	0,08504
24	16	8	-320	0	0,0032	0

25	17	8	-141	0	0,00141	0
26	18	8	-71	6959	0,00071	0,06959
27	19	8	411	16840	0,00411	0,1684
28	23	8	614	11893	0,00614	0,11893
29	24	8	493	0	0,00493	0
30	9	9	532	-2036	0,00532	0,02036
31	10	9	925	-2036	0,00925	0,02036
32	11	9	2029	-2036	0,02029	0,02036
33	12	9	2512	-20000	0,02512	0,2
34	13	9	1244	-22036	0,01244	0,22036
35	14	9	-391	-22036	0,00391	0,22036
36	15	9	298	-22036	0,00298	0,22036
37	16	9	705	-19999	0,00705	0,19999
38	17	9	58	-24998	0,00058	0,24998
39	18	9	-487	-24998	0,00487	0,24998
40	19	9	-24	-24998	0,00024	0,24998
41	20	9	535	-19999	0,00535	0,19999
42	21	9	325	2556	0,00325	0,02556
43	22	9	127	2628	0,00127	0,02628
44	23	9	198	-4999	0,00198	0,04999
45	24	9	266	0	0,00266	0
46	9	10	3316	19620	0,03316	0,1962
47	10	10	2807	19620	0,02807	0,1962
48	11	10	1267	19620	0,01267	0,1962
49	12	10	1117	22036	0,01117	0,22036
50	13	10	2166	-1885	0,02166	0,01885
51	14	10	3560	-5330	0,0356	0,0533
52	15	10	1894	-14238	0,01894	0,14238
53	16	10	138	-20349	0,00138	0,20349
54	17	10	1101	-8116	0,01101	0,08116
55	18	10	2252	4202	0,02252	0,04202
56	19	10	2133	13660	0,02133	0,1366
57	20	10	1330	24998	0,0133	0,24998
58	21	10	1102	18802	0,01102	0,18802
59	22	10	1159	18802	0,01159	0,18802
60	23	10	1371	14881	0,01371	0,14881
61	24	10	1688	34977	0,01688	0,34977
62	25	10	1371	39384	0,01371	0,39384
63	26	10	89	39814	0,00089	0,39814
64	27	10	-1011	38406	0,01011	0,38406
65	28	10	-1567	34600	0,01567	0,346
66	29	10	-910	17916	0,0091	0,17916

67	30	10	-369	-5770	0,00369	0,0577
68	31	10	517	-10121	0,00517	0,10121
69	32	10	1455	-14412	0,01455	0,14412
70	33	10	1215	-18726	0,01215	0,18726
71	34	10	130	-26136	0,0013	0,26136
72	35	10	-206	-29278	0,00206	0,29278
73	36	10	-665	-55431	0,00665	0,55431
74	9	11	172	-11416	0,00172	0,11416
75	10	11	-361	-21430	0,00361	0,2143
76	11	11	573	-17583	0,00573	0,17583
77	12	11	783	0	0,00783	0
78	20	11	827	0	0,00827	0
79	21	11	762	-15963	0,00762	0,15963
80	22	11	535	-10397	0,00535	0,10397
81	30	11	-1319	-21125	0,01319	0,21125
82	31	11	-1220	-21125	0,0122	0,21125
83	32	11	-754	-19999	0,00754	0,19999
84	33	11	-1174	-21125	0,01174	0,21125
85	34	11	-1139	-21125	0,01139	0,21125
86	35	11	-478	-21125	0,00478	0,21125
87	36	11	-222	0	0,00222	0
88	11	12	1261	11676	0,01261	0,11676
89	12	12	1562	0	0,01562	0
90	19	12	765	-6714	0,00765	0,06714
91	20	12	711	0	0,00711	0
92	30	12	307	13527	0,00307	0,13527
93	31	12	629	13192	0,00629	0,13192
94	33	12	991	20175	0,00991	0,20175
95	34	12	490	11439	0,0049	0,11439
96	35	12	305	0	0,00305	0
97	36	12	564	0	0,00564	0
98	12	13	87	0	0,00087	0
99	13	13	2340	2480	0,0234	0,0248
100	14	13	4334	-830	0,04334	0,0083
101	15	13	2885	-1131	0,02885	0,01131
102	16	13	190	0	0,0019	0
103	17	13	1856	2625	0,01856	0,02625
104	18	13	3550	7220	0,0355	0,0722
105	19	13	2569	4624	0,02569	0,04624
106	20	13	960	0	0,0096	0
107	35	13	1407	35253	0,01407	0,35253
108	36	13	464	0	0,00464	0

109	12	14	3133	134817	0,03133	1,34817
110	13	14	1005	141629	0,01005	1,41629
111	14	14	-1424	150324	0,01424	1,50324
112	15	14	-3365	148350	0,03365	1,4835
113	16	14	-4347	146573	0,04347	1,46573
114	17	14	-2922	147462	0,02922	1,47462
115	18	14	-1623	147189	0,01623	1,47189
116	19	14	450	138079	0,0045	1,38079
117	20	14	2292	127255	0,02292	1,27255
118	36	14	2170	84466	0,0217	0,84466
119	37	14	892	82586	0,00892	0,82586
120	38	14	383	80899	0,00383	0,80899
121	39	14	-16	77785	0,00016	0,77785
122	40	14	-1597	71892	0,01597	0,71892
123	41	14	-290	44675	0,0029	0,44675
124	42	14	394	16421	0,00394	0,16421
125	43	14	1501	-4158	0,01501	0,04158
126	44	14	2950	-28987	0,0295	0,28987
127	45	14	1698	-29995	0,01698	0,29995
128	46	14	1096	-24076	0,01096	0,24076
129	47	14	-903	-15407	0,00903	0,15407
130	48	14	-3326	-243	0,03326	0,00243
131	49	14	-225	-12226	0,00225	0,12226
132	50	14	2185	-22920	0,02185	0,2292
133	51	14	3528	-43361	0,03528	0,43361
134	52	14	4854	-71847	0,04854	0,71847
135	53	14	2795	-65115	0,02795	0,65115
136	54	14	791	-61186	0,00791	0,61186
137	55	14	-1663	-54301	0,01663	0,54301
138	56	14	-4512	-37346	0,04512	0,37346
139	57	14	-1695	-55802	0,01695	0,55802
140	58	14	-13	-72450	0,00013	0,7245
141	59	14	1742	-82595	0,01742	0,82595
142	60	14	4075	-85174	0,04075	0,85174
143	61	14	2288	-89285	0,02288	0,89285
144	62	14	-264	-91164	0,00264	0,91164
145	63	14	-1204	-81860	0,01204	0,8186
146	64	14	-1605	-109550	0,01605	1,0955
147	59	15	-3400	-8621	0,034	0,08621
148	60	15	-2243	-19999	0,02243	0,19999
149	61	15	-3803	-33853	0,03803	0,33853
150	62	15	-3416	-35233	0,03416	0,35233

151	63	15	-1588	-35233	0,01588	0,35233
152	64	15	-70	0	0,0007	0
153	62	16	-623	-1676	0,00623	0,01676
154	63	16	-360	0	0,0036	0
155	64	16	-258	0	0,00258	0
156	63	17	1219	12409	0,01219	0,12409
157	64	17	82	0	0,00082	0
158	64	18	2675	68811	0,02675	0,68811
159	65	18	951	33748	0,00951	0,33748
160	66	18	-736	3389	0,00736	0,03389
161	67	18	-1417	13311	0,01417	0,13311
162	68	18	-1547	-54804	0,01547	0,54804
163	67	19	-545	-83158	0,00545	0,83158
164	68	19	179	0	0,00179	0
165	68	20	-60	0	0,0006	0
166	68	21	-169	0	0,00169	0
167	69	21	704	33996	0,00704	0,33996
168	24	22	-1882	-114853	0,01882	1,14853
169	25	22	-1790	-97446	0,0179	0,97446
170	26	22	-1092	-109647	0,01092	1,09647
171	27	22	601	-111307	0,00601	1,11307
172	28	22	3132	-112467	0,03132	1,12467
173	29	22	1940	-111101	0,0194	1,11101
174	30	22	-251	-103995	0,00251	1,03995
175	31	22	-2142	-81939	0,02142	0,81939
176	32	22	-3984	-59985	0,03984	0,59985
177	33	22	-574	-80773	0,00574	0,80773
178	34	22	2089	-98590	0,02089	0,9859
179	35	22	3637	-81600	0,03637	0,816
180	36	22	3954	-83256	0,03954	0,83256
181	37	22	3243	-83588	0,03243	0,83588
182	38	22	2736	-84944	0,02736	0,84944
183	39	22	1222	-73678	0,01222	0,73678
184	40	22	-1130	-89153	0,0113	0,89153
185	41	22	1143	-95339	0,01143	0,95339
186	42	22	1516	-114111	0,01516	1,14111
187	43	22	1835	-113592	0,01835	1,13592
188	44	22	3404	-107041	0,03404	1,07041
189	45	22	882	-99188	0,00882	0,99188
190	46	22	-913	-90240	0,00913	0,9024
191	47	22	-2095	-70925	0,02095	0,70925
192	48	22	-3183	-80615	0,03183	0,80615

193	68	22	1266	78539	0,01266	0,78539
194	69	22	23	38836	0,00023	0,38836
195	70	22	-535	36266	0,00535	0,36266
196	71	22	-413	14556	0,00413	0,14556
197	72	22	-332	-6315	0,00332	0,06315
198	24	23	371	0	0,00371	0
199	25	23	-791	-26307	0,00791	0,26307
200	26	23	-2445	-26307	0,02445	0,26307
201	27	23	-2537	-26307	0,02537	0,26307
202	28	23	-2106	-19999	0,02106	0,19999
203	29	23	-3374	-944	0,03374	0,00944
204	35	23	-537	-21058	0,00537	0,21058
205	36	23	3232	-19999	0,03232	0,19999
206	37	23	170	-22364	0,0017	0,22364
207	38	23	-3214	-25935	0,03214	0,25935
208	39	23	-1780	-29558	0,0178	0,29558
209	40	23	241	20700	0,00241	0,207
210	41	23	-1629	1244	0,01629	0,01244
211	42	23	-2950	-10181	0,0295	0,10181
212	43	23	-479	-17902	0,00479	0,17902
213	44	23	434	-19999	0,00434	0,19999
214	45	23	-360	-18871	0,0036	0,18871
215	46	23	-1842	-16944	0,01842	0,16944
216	47	23	-683	-14317	0,00683	0,14317
217	48	23	160	0	0,0016	0
218	72	23	-55	-9368	0,00055	0,09368
219	24	24	23	0	0,00023	0
220	25	24	241	0	0,00241	0
221	26	24	421	-1166	0,00421	0,01166
222	27	24	980	31094	0,0098	0,31094
223	48	24	11	0	0,00011	0
224	49	24	-219	1158	0,00219	0,01158
225	50	24	1	-6781	0,00001	0,06781
226	51	24	20	-31829	0,0002	0,31829
227	24	25	839	0	0,00839	0
228	25	25	2179	24468	0,02179	0,24468
229	48	25	377	0	0,00377	0
230	49	25	1575	8963	0,01575	0,08963
231	50	25	2687	8963	0,02687	0,08963
232	51	25	1456	8963	0,01456	0,08963
233	52	25	-198	4345	0,00198	0,04345
234	53	25	1254	-9911	0,01254	0,09911

235	12	26	-1888	-96937	0,01888	0,96937
236	13	26	-52	-80521	0,00052	0,80521
237	14	26	2236	-84906	0,02236	0,84906
238	15	26	3898	-84056	0,03898	0,84056
239	16	26	4023	-82030	0,04023	0,8203
240	17	26	3724	-82532	0,03724	0,82532
241	18	26	1912	-82135	0,01912	0,82135
242	19	26	-1231	-66857	0,01231	0,66857
243	20	26	-2621	-33798	0,02621	0,33798
244	21	26	-1367	-15991	0,01367	0,15991
245	22	26	105	7959	0,00105	0,07959
246	23	26	1734	23040	0,01734	0,2304
247	24	26	3492	77737	0,03492	0,77737
248	48	26	4006	160407	0,04006	1,60407
249	49	26	2645	158294	0,02645	1,58294
250	50	26	1107	167610	0,01107	1,6761
251	51	26	-958	159243	0,00958	1,59243
252	52	26	-2970	144742	0,0297	1,44742
253	53	26	-2007	132350	0,02007	1,3235
254	54	26	-971	109411	0,00971	1,09411
255	55	26	1664	59690	0,01664	0,5969
256	56	26	3168	-1232	0,03168	0,01232
257	57	26	2173	-36692	0,02173	0,36692
258	58	26	1372	-27979	0,01372	0,27979
259	59	26	172	-46296	0,00172	0,46296
260	60	26	-1558	-128030	0,01558	1,2803
261	12	27	-246	0	0,00246	0
262	13	27	-1659	-24025	0,01659	0,24025
263	14	27	-3303	-20949	0,03303	0,20949
264	15	27	-2167	-18751	0,02167	0,18751
265	16	27	321	-20000	0,00321	0,2
266	17	27	-1883	-17003	0,01883	0,17003
267	18	27	-2973	-12489	0,02973	0,12489
268	19	27	-1134	-4380	0,01134	0,0438
269	58	27	-2208	-77209	0,02208	0,77209
270	59	27	-608	-63872	0,00608	0,63872
271	60	27	787	0	0,00787	0
272	12	28	-363	0	0,00363	0
273	13	28	-681	12208	0,00681	0,12208
274	59	28	1012	-5292	0,01012	0,05292
275	60	28	1125	0	0,01125	0
276	11	29	-452	59412	0,00452	0,59412

277	12	29	-601	0	0,00601	0
278	60	29	54	0	0,00054	0
279	9	30	-1591	6086	0,01591	0,06086
280	10	30	-2553	15703	0,02553	0,15703
281	11	30	-1813	-3655	0,01813	0,03655
282	12	30	-250	54495	0,0025	0,54495
283	60	30	2747	142178	0,02747	1,42178
284	61	30	1084	119346	0,01084	1,19346
285	62	30	-1328	88810	0,01328	0,8881
286	63	30	-3074	77483	0,03074	0,77483
287	64	30	-4570	60308	0,0457	0,60308
288	65	30	-3256	646	0,03256	0,00646
289	66	30	-1083	-59884	0,01083	0,59884
290	67	30	1871	-97752	0,01871	0,97752
291	68	30	4165	-111484	0,04165	1,11484
292	69	30	2405	-130105	0,02405	1,30105
293	70	30	443	-147668	0,00443	1,47668
294	71	30	-2187	-149309	0,02187	1,49309
295	72	30	-3603	-163391	0,03603	1,63391
296	9	31	-17	-29271	0,00017	0,29271
297	10	31	124	-29271	0,00124	0,29271
298	67	31	-2284	-3398	0,02284	0,03398
299	68	31	115	-19999	0,00115	0,19999
300	69	31	-1480	-38468	0,0148	0,38468
301	70	31	-3310	-37157	0,0331	0,37157
302	71	31	-856	-37157	0,00856	0,37157
303	72	31	942	-19999	0,00942	0,19999
304	9	32	-688	0	0,00688	0
305	10	32	-832	0	0,00832	0
306	70	32	-327	17399	0,00327	0,17399
307	71	32	-647	-1934	0,00647	0,01934
308	72	32	-713	0	0,00713	0
309	9	33	-336	-29271	0,00336	0,29271
310	10	33	-402	-29271	0,00402	0,29271
311	11	33	-261	-33338	0,00261	0,33338
312	72	33	-670	28957	0,0067	0,28957
313	9	34	-2001	-68033	0,02001	0,68033
314	10	34	-2322	-53015	0,02322	0,53015
315	11	34	-1640	-28881	0,0164	0,28881
316	12	34	434	27191	0,00434	0,27191
317	12	35	185	0	0,00185	0
318	13	35	496	-4763	0,00496	0,04763

319	12	36	-227	0	0,00227	0
320	13	36	-112	0	0,00112	0
321	14	36	-209	-8412	0,00209	0,08412
322	15	36	-445	17456	0,00445	0,17456

MAKSİMUM	0,04854	1,6761
AVERAGE	0,01357	0,39153

Table 13. Sharpness values with 0.03 linear interpolation, 0.1 blending

Sequence	XCoordinate	ZCoordinate	Old_Curvature	New_Curvature	Mutlak değer old	Mutlak değer new
1	28	5	-188	0	0,0019	0
2	67	5	-1445	-8008	0,0145	0,0801
3	68	5	-600	0	0,006	0
4	24	6	2423	101825	0,0242	1,0183
5	25	6	2128	86933	0,0213	0,8693
6	26	6	2246	86911	0,0225	0,8691
7	27	6	1254	70734	0,0125	0,7073
8	28	6	-258	-19676	0,0026	0,1968
9	68	6	-2062	-47866	0,0206	0,4787
10	69	6	-1384	42482	0,0138	0,4248
11	70	6	85	62434	0,0009	0,6243
12	71	6	1092	84700	0,0109	0,847
13	72	6	2504	117827	0,025	1,1783
14	16	7	142	40878	0,0014	0,4088
15	17	7	1448	26974	0,0145	0,2697
16	24	7	-129	0	0,0013	0
17	25	7	398	17518	0,004	0,1752
18	9	8	206	23068	0,0021	0,2307
19	10	8	-300	17518	0,003	0,1752
20	11	8	-462	20002	0,0046	0,2
21	13	8	-50	19390	0,0005	0,1939
22	14	8	217	11943	0,0022	0,1194
23	15	8	-135	7609	0,0014	0,0761
24	16	8	-320	0	0,0032	0
25	17	8	-141	0	0,0014	0
26	18	8	-71	6226	0,0007	0,0623
27	19	8	411	15068	0,0041	0,1507
28	23	8	614	10641	0,0061	0,1064
29	24	8	493	0	0,0049	0

30	9	9	532	-21822	0,0053	0,2182
31	10	9	925	-21822	0,0093	0,2182
32	11	9	2029	-21822	0,0203	0,2182
33	12	9	2512	-59999	0,0251	0,6
34	13	9	1244	-61822	0,0124	0,6182
35	14	9	-391	-61822	0,0039	0,6182
36	15	9	298	-61822	0,003	0,6182
37	16	9	705	-39999	0,0071	0,4
38	17	9	58	-64472	0,0006	0,6447
39	18	9	-487	-64472	0,0049	0,6447
40	19	9	-24	-64472	0,0002	0,6447
41	20	9	535	-59999	0,0054	0,6
42	21	9	325	-17712	0,0033	0,1771
43	22	9	127	-17647	0,0013	0,1765
44	23	9	198	-24472	0,002	0,2447
45	24	9	266	0	0,0027	0
46	9	10	3316	57554	0,0332	0,5755
47	10	10	2807	57554	0,0281	0,5755
48	11	10	1267	57554	0,0127	0,5755
49	12	10	1117	61822	0,0112	0,6182
50	13	10	2166	39549	0,0217	0,3955
51	14	10	3560	36342	0,0356	0,3634
52	15	10	1894	28048	0,0189	0,2805
53	16	10	138	2433	0,0014	0,0243
54	17	10	1101	33641	0,011	0,3364
55	18	10	2252	45110	0,0225	0,4511
56	19	10	2133	53916	0,0213	0,5392
57	20	10	1330	64472	0,0133	0,6447
58	21	10	1102	56822	0,011	0,5682
59	22	10	1159	56822	0,0116	0,5682
60	23	10	1371	53315	0,0137	0,5332
61	24	10	1688	73401	0,0169	0,734
62	25	10	1371	77276	0,0137	0,7728
63	26	10	89	77539	0,0009	0,7754
64	27	10	-1011	76238	0,0101	0,7624
65	28	10	-1567	72776	0,0157	0,7278
66	29	10	-910	57463	0,0091	0,5746
67	30	10	-369	35966	0,0037	0,3597
68	31	10	517	31914	0,0052	0,3191
69	32	10	1455	27920	0,0146	0,2792
70	33	10	1215	23903	0,0122	0,239
71	34	10	130	17004	0,0013	0,17

72	35	10	-206	14078	0,0021	0,1408
73	36	10	-665	-51608	0,0067	0,5161
74	9	11	172	-30214	0,0017	0,3021
75	10	11	-361	-39174	0,0036	0,3917
76	11	11	573	-35732	0,0057	0,3573
77	12	11	783	0	0,0078	0
78	20	11	827	0	0,0083	0
79	21	11	762	-34283	0,0076	0,3428
80	22	11	535	-29302	0,0054	0,293
81	30	11	-1319	-61006	0,0132	0,6101
82	31	11	-1220	-61006	0,0122	0,6101
83	32	11	-754	-59999	0,0075	0,6
84	33	11	-1174	-61006	0,0117	0,6101
85	34	11	-1139	-61006	0,0114	0,6101
86	35	11	-478	-61006	0,0048	0,6101
87	36	11	-222	0	0,0022	0
88	11	12	1261	10447	0,0126	0,1045
89	12	12	1562	0	0,0156	0
90	19	12	765	-6251	0,0077	0,0625
91	20	12	711	0	0,0071	0
92	30	12	307	12103	0,0031	0,121
93	31	12	629	11803	0,0063	0,118
94	33	12	991	18051	0,0099	0,1805
95	34	12	490	10235	0,0049	0,1024
96	35	12	305	0	0,0031	0
97	36	12	564	0	0,0056	0
98	12	13	87	0	0,0009	0
99	13	13	2340	2309	0,0234	0,0231
100	14	13	4334	-773	0,0433	0,0077
101	15	13	2885	-1053	0,0289	0,0105
102	16	13	190	0	0,0019	0
103	17	13	1856	2444	0,0186	0,0244
104	18	13	3550	6722	0,0355	0,0672
105	19	13	2569	4305	0,0257	0,0431
106	20	13	960	0	0,0096	0
107	35	13	1407	31542	0,0141	0,3154
108	36	13	464	0	0,0046	0
109	12	14	3133	162731	0,0313	1,6273
110	13	14	1005	168658	0,0101	1,6866
111	14	14	-1424	176438	0,0142	1,7644
112	15	14	-3365	174672	0,0337	1,7467
113	16	14	-4347	173081	0,0435	1,7308

114	17	14	-2922	173877	0,0292	1,7388
115	18	14	-1623	173633	0,0162	1,7363
116	19	14	450	165482	0,0045	1,6548
117	20	14	2292	155965	0,0229	1,5597
118	36	14	2170	117680	0,0217	1,1768
119	37	14	892	115837	0,0089	1,1584
120	38	14	383	114058	0,0038	1,1406
121	39	14	-16	111354	0,0002	1,1135
122	40	14	-1597	106097	0,016	1,061
123	41	14	-290	81046	0,0029	0,8105
124	42	14	394	55023	0,0039	0,5502
125	43	14	1501	36398	0,015	0,364
126	44	14	2950	13831	0,0295	0,1383
127	45	14	1698	12465	0,017	0,1247
128	46	14	1096	17340	0,011	0,1734
129	47	14	-903	25327	0,009	0,2533
130	48	14	-3326	39368	0,0333	0,3937
131	49	14	-225	28061	0,0023	0,2806
132	50	14	2185	17945	0,0219	0,1795
133	51	14	3528	-387	0,0353	0,0039
134	52	14	4854	-26043	0,0485	0,2604
135	53	14	2795	-20278	0,028	0,2028
136	54	14	791	-17052	0,0079	0,1705
137	55	14	-1663	-10432	0,0166	0,1043
138	56	14	-4512	5431	0,0451	0,0543
139	57	14	-1695	-11657	0,017	0,1166
140	58	14	-13	-27057	0,0001	0,2706
141	59	14	1742	-36073	0,0174	0,3607
142	60	14	4075	-38473	0,0408	0,3847
143	61	14	2288	-42301	0,0229	0,423
144	62	14	-264	-44051	0,0026	0,4405
145	63	14	-1204	-35388	0,012	0,3539
146	64	14	-1605	-1E+05	0,0161	1,0199
147	59	15	-3400	-49819	0,034	0,4982
148	60	15	-2243	-59999	0,0224	0,6
149	61	15	-3803	-72394	0,038	0,7239
150	62	15	-3416	-73630	0,0342	0,7363
151	63	15	-1588	-73630	0,0159	0,7363
152	64	15	-70	0	0,0007	0
153	62	16	-623	-1499	0,0062	0,015
154	63	16	-360	0	0,0036	0
155	64	16	-258	0	0,0026	0

156	63	17	1219	11102	0,0122	0,111
157	64	17	82	0	0,0008	0
158	64	18	2675	103673	0,0268	1,0367
159	65	18	951	71243	0,0095	0,7124
160	66	18	-736	43525	0,0074	0,4353
161	67	18	-1417	52232	0,0142	0,5223
162	68	18	-1547	-51025	0,0155	0,5103
163	67	19	-545	-1E+05	0,0055	1,1651
164	68	19	179	0	0,0018	0
165	68	20	-60	0	0,0006	0
166	68	21	-169	0	0,0017	0
167	69	21	704	31651	0,007	0,3165
168	24	22	-1882	-1E+05	0,0188	1,0276
169	25	22	-1790	-44855	0,0179	0,4486
170	26	22	-1092	-55771	0,0109	0,5577
171	27	22	601	-57256	0,006	0,5726
172	28	22	3132	-58294	0,0313	0,5829
173	29	22	1940	-57072	0,0194	0,5707
174	30	22	-251	-50448	0,0025	0,5045
175	31	22	-2142	-30556	0,0214	0,3056
176	32	22	-3984	-10830	0,0398	0,1083
177	33	22	-574	-29556	0,0057	0,2956
178	34	22	2089	-45702	0,0209	0,457
179	35	22	3637	-29903	0,0364	0,299
180	36	22	3954	-31385	0,0395	0,3139
181	37	22	3243	-31683	0,0324	0,3168
182	38	22	2736	-32895	0,0274	0,329
183	39	22	1222	-22816	0,0122	0,2282
184	40	22	-1130	-57663	0,0113	0,5766
185	41	22	1143	-42974	0,0114	0,4297
186	42	22	1516	-59770	0,0152	0,5977
187	43	22	1835	-59306	0,0184	0,5931
188	44	22	3404	-53445	0,034	0,5345
189	45	22	882	-46418	0,0088	0,4642
190	46	22	-913	-38412	0,0091	0,3841
191	47	22	-2095	-21130	0,021	0,2113
192	48	22	-3183	-72129	0,0318	0,7213
193	68	22	1266	112377	0,0127	1,1238
194	69	22	23	76025	0,0002	0,7603
195	70	22	-535	74069	0,0054	0,7407
196	71	22	-413	54410	0,0041	0,5441
197	72	22	-332	15499	0,0033	0,155

198	24	23	371	0	0,0037	0
199	25	23	-791	-65872	0,0079	0,6587
200	26	23	-2445	-65872	0,0245	0,6587
201	27	23	-2537	-65872	0,0254	0,6587
202	28	23	-2106	-59999	0,0211	0,6
203	29	23	-3374	-42258	0,0337	0,4226
204	35	23	-537	-60985	0,0054	0,6099
205	36	23	3232	-60000	0,0323	0,6
206	37	23	170	-62201	0,0017	0,622
207	38	23	-3214	-65526	0,0321	0,6553
208	39	23	-1780	-68899	0,0178	0,689
209	40	23	241	17893	0,0024	0,1789
210	41	23	-1629	-40221	0,0163	0,4022
211	42	23	-2950	-50859	0,0295	0,5086
212	43	23	-479	-58046	0,0048	0,5805
213	44	23	434	-60000	0,0043	0,6
214	45	23	-360	-58949	0,0036	0,5895
215	46	23	-1842	-57155	0,0184	0,5716
216	47	23	-683	-54709	0,0068	0,5471
217	48	23	160	0	0,0016	0
218	72	23	-55	-10487	0,0006	0,1049
219	24	24	23	0	0,0002	0
220	25	24	241	0	0,0024	0
221	26	24	421	-1086	0,0042	0,0109
222	27	24	980	28949	0,0098	0,2895
223	48	24	11	0	0,0001	0
224	49	24	-219	1036	0,0022	0,0104
225	50	24	1	-6067	1E-05	0,0607
226	51	24	20	-28479	0,0002	0,2848
227	24	25	839	0	0,0084	0
228	25	25	2179	22780	0,0218	0,2278
229	48	25	377	0	0,0038	0
230	49	25	1575	8020	0,0158	0,0802
231	50	25	2687	8020	0,0269	0,0802
232	51	25	1456	8020	0,0146	0,0802
233	52	25	-198	3888	0,002	0,0389
234	53	25	1254	-8867	0,0125	0,0887
235	12	26	-1888	-86733	0,0189	0,8673
236	13	26	-52	-29793	0,0005	0,2979
237	14	26	2236	-33717	0,0224	0,3372
238	15	26	3898	-32956	0,039	0,3296
239	16	26	4023	-31144	0,0402	0,3114

240	17	26	3724	-31593	0,0372	0,3159
241	18	26	1912	-31238	0,0191	0,3124
242	19	26	-1231	-17568	0,0123	0,1757
243	20	26	-2621	12249	0,0262	0,1225
244	21	26	-1367	28796	0,0137	0,288
245	22	26	105	50947	0,0011	0,5095
246	23	26	1734	64528	0,0173	0,6453
247	24	26	3492	113755	0,0349	1,1376
248	48	26	4006	190724	0,0401	1,9072
249	49	26	2645	189082	0,0265	1,8908
250	50	26	1107	197755	0,0111	1,9776
251	51	26	-958	189966	0,0096	1,8997
252	52	26	-2970	176464	0,0297	1,7646
253	53	26	-2007	164927	0,0201	1,6493
254	54	26	-971	143789	0,0097	1,4379
255	55	26	1664	98158	0,0166	0,9816
256	56	26	3168	42136	0,0317	0,4214
257	57	26	2173	9846	0,0217	0,0985
258	58	26	1372	18663	0,0137	0,1866
259	59	26	172	2274	0,0017	0,0227
260	60	26	-1558	-1E+05	0,0156	1,1455
261	12	27	-246	0	0,0025	0
262	13	27	-1659	-63747	0,0166	0,6375
263	14	27	-3303	-60884	0,033	0,6088
264	15	27	-2167	-58837	0,0217	0,5884
265	16	27	321	-59999	0,0032	0,6
266	17	27	-1883	-57210	0,0188	0,5721
267	18	27	-2973	-53007	0,0297	0,5301
268	19	27	-1134	-45457	0,0113	0,4546
269	58	27	-2208	-1E+05	0,0221	1,1326
270	59	27	-608	-1E+05	0,0061	1,0085
271	60	27	787	0	0,0079	0
272	12	28	-363	0	0,0036	0
273	13	28	-681	11366	0,0068	0,1137
274	59	28	1012	-4927	0,0101	0,0493
275	60	28	1125	0	0,0113	0
276	11	29	-452	53158	0,0045	0,5316
277	12	29	-601	0	0,006	0
278	60	29	54	0	0,0005	0
279	9	30	-1591	47887	0,0159	0,4789
280	10	30	-2553	56492	0,0255	0,5649
281	11	30	-1813	39398	0,0181	0,394

282	12	30	-250	92116	0,0025	0,9212
283	60	30	2747	173752	0,0275	1,7375
284	61	30	1084	153233	0,0108	1,5323
285	62	30	-1328	125810	0,0133	1,2581
286	63	30	-3074	115002	0,0307	1,15
287	64	30	-4570	98837	0,0457	0,9884
288	65	30	-3256	44777	0,0326	0,4478
289	66	30	-1083	-10095	0,0108	0,101
290	67	30	1871	-44734	0,0187	0,4473
291	68	30	4165	-57021	0,0417	0,5702
292	69	30	2405	-73681	0,0241	0,7368
293	70	30	443	-89396	0,0044	0,894
294	71	30	-2187	-90864	0,0219	0,9086
295	72	30	-3603	-1E+05	0,036	1,2409
296	9	31	-17	-68631	0,0002	0,6863
297	10	31	124	-68631	0,0012	0,6863
298	67	31	-2284	-44543	0,0228	0,4454
299	68	31	115	-60000	0,0012	0,6
300	69	31	-1480	-77194	0,0148	0,7719
301	70	31	-3310	-75974	0,0331	0,7597
302	71	31	-856	-75974	0,0086	0,7597
303	72	31	942	-40000	0,0094	0,4
304	9	32	-688	0	0,0069	0
305	10	32	-832	0	0,0083	0
306	70	32	-327	16199	0,0033	0,162
307	71	32	-647	-1801	0,0065	0,018
308	72	32	-713	0	0,0071	0
309	9	33	-336	-68631	0,0034	0,6863
310	10	33	-402	-68631	0,004	0,6863
311	11	33	-261	-72418	0,0026	0,7242
312	72	33	-670	46960	0,0067	0,4696
313	9	34	-2001	-18430	0,02	0,1843
314	10	34	-2322	-4993	0,0232	0,0499
315	11	34	-1640	16600	0,0164	0,166
316	12	34	434	66695	0,0043	0,667
317	12	35	185	0	0,0019	0
318	13	35	496	-4434	0,005	0,0443
319	12	36	-227	0	0,0023	0
320	13	36	-112	0	0,0011	0
321	14	36	-209	-7832	0,0021	0,0783