Procedural Generation of Adjustable Terrain for Application in Computer Games Using 2D Maps

Izabella Antoniuk(✉) and Przemysław Rokita

Institute of Computer Science, Warsaw University of Technology,
Nowowiejska 15/19, 00-665 Warsaw, Poland
I.Antoniuk@stud.elka.pw.edu.pl,
P.Rokita@ii.pw.edu.pl

Abstract. This paper describes method for generating 3D terrain for usage in computer games by processing set of 2D maps and employing of user-specified parameters. Most of existing solutions don't allow for modifications during generation process, while introducing any changes usually requires complex activities, or is limited to adjusting input maps. We present our solution that allows not only for easy edition of created terrain, but also verification of its quality at each step of generation.

1 Introduction

Computer games, depending on their genre, can include numerous terrains with different properties and details. Some areas are represented schematically, while others are created with great level of detail, showing even the smallest elements.

Terrain designed for usage in computer games must posses series of different properties as well as allow easy introduction of various changes. Those concern not only terrain shape but also other game elements connected to them such as: story, quests, placement of objects, enemies and other constituents. Because of those requirements most of maps are modelled by hand, usually requiring considerable amount of time to create. Procedural content generation in it most common form is rarely used in those applications, because of low level of control over created content, as well as many difficulties with further edition of obtained objects. However it may provide interesting results.

While considering different terrains and arrangement of occurring elements, it is possible to notice that most of them can be divided by similar features and inclusive properties. Parts that contain distinctive similarities (considering that they are smaller and content inside them is mostly uniform) are much easier to generate and combine. It is also easier to decide precise set of constraints for single tile. That leads to the idea of generating 3D terrain using image processing approach based on simplified 2D maps.

While this concept is not new, existing methods still have a few drawbacks. They allow for creation of suitable and complex terrains, but their final product lacks the capability for further terrain adjustments. Usually procedure allows introducing modifications either by editing input maps (which greatly decreases

M. Kryszkiewicz et al. (Eds.): PReMI 2015, LNCS 9124, pp. 75–84, 2015.
DOI: 10.1007/978-3-319-19941-2_8

level of control over final shape of generated object) or requires conversion of generated terrain to 3D modelling environment (where results of such actions are not always satisfactory and often hard to predict). Another issue is complexity of obtained terrain, which not always meets computer games requirements.

In our approach we propose a method for creating 3D terrains by processing set of simplified maps. This allows for fast and precise generation, easy adjustment and regeneration of both entire terrain and its individual parts. As a test platform for our algorithm we chose Blender 2.73a application (for Blender documentation see [19]).

The rest of the paper is organized as follows. In Sect. 2 we review other works related to our area of research. Section 3 describes initial assumptions for our method. Section 4 contains procedure overview. Section 5 outlines some areas of future work. Finally we conclude our work in Sect. 6.

2 Related Work

Procedural generation for computer games is not a new topic similarly to generating terrain by processing 2D maps or other simplified input. Among existing algorithms some focus only on generating 2D maps from their more basic versions [6], while other generate complex terrains, containing various data and details [17,18]

One of many problems described in some of the existing works is building entire worlds in real time (which is not an easy case, considering amount of data that needs to be processed) [2]. Other interesting area of research is creating objects from input given by the users in form of pre-processed objects [5]. Some works focus on increasing level of control over generated output by defining terrain with various constraints and parameters such as set of actions that will be performed at generated map [4] or by restricting some of its properties to better fit desired results [13].

Simple and complex algorithms proposed for creation of various virtual worlds, range from basic methods [7] to entire frameworks and methodologies [11,16]. Among the interesting issues is representation of real world data through sets of height maps [10].

When it comes to generation itself, there are many different ways to realize it. Some solutions use sets of height maps to create complex objects [12], others introduce genetic algorithms, evolving terrain according to user preferences [8], or focus on efficient introduction of various changes to created map(i.e. adding roads) [9].

There are many methods for generation of objects and terrains. For detailed study of procedural content generation algorithms see [1,3,14,15].

3 Initial Assumptions

In this section we present assumptions that led to design of our algorithm in its current form.

While planning our procedure, we assumed that generated terrain must meet certain constraints and have a few significant properties, if it is to be used in computer games.

Most important features concern terrain itself. We assumed that any area must allow for introducing various changes, both during algorithm operations and after it terminates its work. At the same time, any created object should be as close to desired output as possible, providing high level of control over terrain properties.

Another crucial issue is generation process. Designing and modelling of complex 3D objects takes considerable amount of time. We decided to use image processing and base our procedure on 2D maps to simplify the process. In our approach each pixel represents different region inside generated object. Since terrain tiles contain different data, that would be difficult to represent on single image, also increasing its complexity. We decided to use set of maps, where each file contains different information. Currently we consider three type of maps: height map (containing information about basic terrain level, represented in greyscale colour value), dispersion map (describing dispersion range for height value in single region, stored in text file) and terrain map (showing different terrains distribution through the scene, represented as RGB colour value). Each map is independent from others therefore performing changes to one of them does not require updating others (for example, if we want to slightly adjust dispersion range, we don't need to change terrain level or type).

While creating 2D images is easier than modelling 3D environment, very complex maps can still take a lot of time to finish, especially if we consider large maps with great level of detail. Since each terrain can be divided into patches containing elements of the same type (like mountains, plains, forests etc.) we decided to simplify that input as much as possible. Such patches are represented by pixels in our maps and are further processed adding details during algorithm operations and merging terrains according to user-defined properties.

After generation is complete, each of the regions processed from pixels in input maps is represented as different object. Although its internal properties are to some point determined by adjacent regions, it can be further adjusted or regenerated independently, therefore allowing for easier management of terrain data. At the same time, one faulty region doesn't disqualify entire terrain - it can simply be regenerated (or manually adjusted), without influencing other, correct regions.

4 Algorithm Overview

In this section we describe procedure we use for generating 3D maps by processing set of 2D images and using chosen properties of emerging terrain. As a test platform for our algorithm we use newest Blender application (version 2.73a). We chose this 3D modelling environment because it contains complete python interpreter and also allows for easy access to program functionality.

Terrain generation algorithm consists of two main steps: processing terrain maps and building terrain according to input data and predefined properties.

4.1 Processing Terrain Maps

Storing different terrain data in separate files requires precise defining what type of information is taken from which map. In our method we currently consider three types of data, defining our output object.

First map contains data about basic altitude level of region represented by each pixel. Since we assume that RGB values in that map are equal we read only one colour value and represent terrain height with it (Fig. 1. left). This parameter defines basic terrain level, which is a foundation for any further calculations (like obtaining dispersion or performing terrain generation).

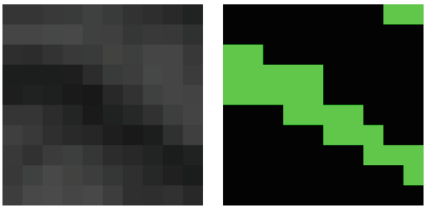

Fig. 1. Input Files: height map (left), and terrain map (right)

Second map, stored in text file, contains values of dispersion for each region. Total range of this parameter is described by two variables - one concerning dispersion value below terrain level, and one for terrain above it (depending on desired results, those values might differ rather significantly).

Final map stored in RGB image (Fig. 1. right), contains description of different terrains occurring in generated object. Each terrain type is assigned a colour, and is then recognized by simply comparing pixel RGB values to those stored in terrain dictionary (each item, apart from terrain ID also contains basic properties for 3D generation algorithms to work with).

4.2 3D Terrain Generation

After information contained in map files is processed and interpreted we obtain few sets of data that are used in next step, which is terrain generation. Our algorithm currently consist of three main parts: creating and placing basic regions, adding details to those regions and finally joining created terrain tiles and reducing visibility of borders between them, as presented in Algorithm 1.

In first step of our procedure we use values obtained by processing image containing height map to place basic grids across the scene, as shown at Fig. 2 (top). Each grid has predefined number of vertices. We also normalize height values for it to better fit terrain characteristic, using user-defined parameter (normalization parameter allows us to adjust height differences between regions, decreasing or increasing tiles dispersion along “z” axis).

Algorithm 1. Terrain Generation

```
Require: Size, Heights, Dispersions, Terrains, Regions, Normalization
  for all Regions do
    CreateGrid(Size)
    GetHeight(Heights, Region)
    PlaceGrid(Height, Normalization)
  end for
  for all Grids do
    AssignTerrainType(Region)
    ChooseAlgorithm(TerrainType)
    ReadDispersion(Dispersions, Region)
    GenerateTerrain(Region, Algorithm, Dispersion)
  end for
  for all Grids do
    CheckNeighbours()
    AlignBorders(Regions, TerrainType)
    SmoothTransition()
  end for
```

Although in this step we do not perform any complex operations, obtained results give quite good impression about overall terrain shape. It also provides the possibility to fix any defects that generated terrain can have at this point (like to high dispersion between regions) before main generation process begins.

After we create basic version of our terrain, we then use information from remaining maps, to add details. First we check type of terrain for each existing region (Fig. 2. bottom), since that information decides what kind of algorithm will be used for generation, as well as some of its input parameters. After obtaining required data, we run proper algorithm with given parameters and obtained dispersion range. At this point we gain set of separate regions, where each of them contains part of output terrain, generated according to data from map files. Results of such generation are shown at Fig. 3 (top).

Final step of our procedure connects created regions and blurs border between them. This part of algorithm works on two neighbouring tiles. It first checks placement of border vertices (since grids are generated next to each other, global x and y coordinates for those vertices will be the same) and moves them to new location. Final placement of each vertex is calculated according to set of constraints specified by user (i.e. they can be placed in the middle, or closer to one region location, depending from influence that each terrain will have at calculations outcome). Results of joining tiles are shown at Fig. 3 (middle).

Although at this point created regions are connected, borders between tiles are clearly visible. We use simple procedure to blur those transitions, by defining percentage of terrain that will be aligned (it corresponds directly to number of vertices in each region that will be modified). Subsequently we level that terrain to its border (already connected with neighbouring region), gradually decreasing alignment influence. Final terrain is shown in Fig. 3 (bottom).

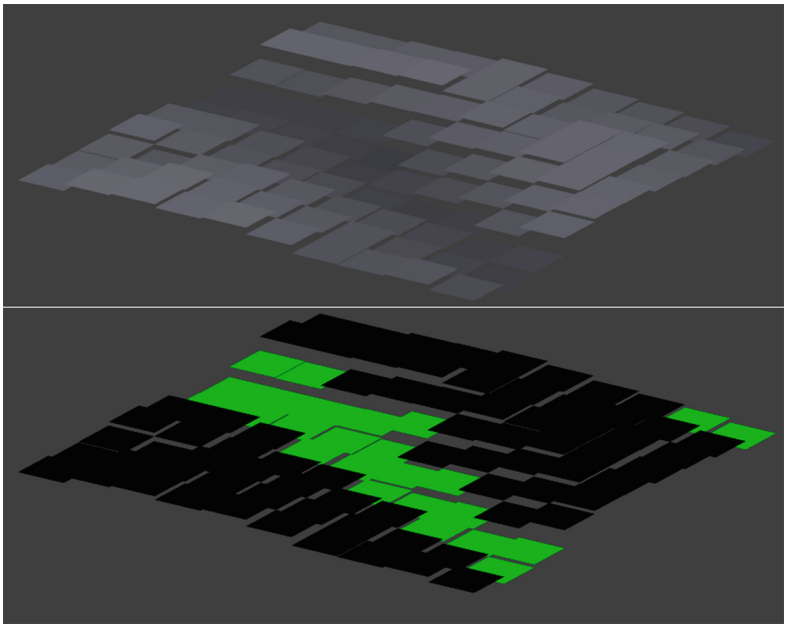

Fig. 2. Basic generation: placing regions (top), and assigning terrain type (bottom)

After this final step we obtain terrain made up from set of separate tiles, where each of elements can be processed and modified independently. Since computer games rarely store terrain as single object, and rather load small parts of it when they are needed (i.e. while player travels to another locations or approaches borders between them), such data representation helps to avoid unnecessary operations (like dividing terrain after it was created, to include it in computer game). At the same time we also allow possibility to connect regions into single object (in this operation we consider both joining all regions as well as restricting this action to chosen set of elements).

In its current form (see Fig. 4), our algorithm has both advantages and disadvantages. We still use rather simple procedures for generating terrain inside tiles and blurring borders between them, therefore the amount of available terrains as well as their variety is limited. At the same time terrains we can generate can be easily edited and adjusted because they are editable Bender objects. Unfortunately using Blender application as our test platform greatly limits our performance. Our algorithm can generate terrain (including complex maps, with many different terrain tiles), from relatively simple input, but without inserting any additional detail (like vegetation or buildings). One big advantage is that each step of our procedure can be verified, limiting number of errors occurring in final object. Also due to its modular structure, appending new functionality is an easy task.

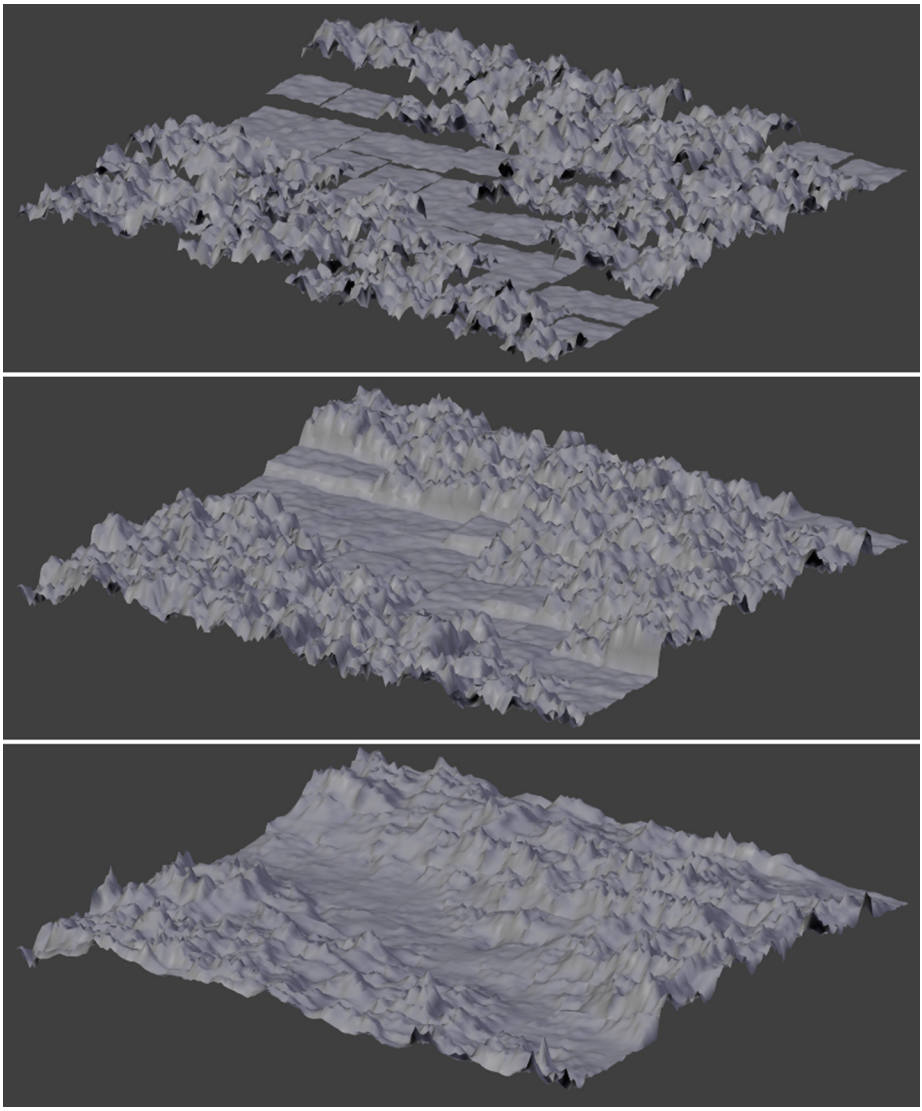

Fig. 3. Terrain generation: generating terrain details in regions (top), joining regions (middle) and blurring borders between regions (bottom)

5 Future Work

Our algorithm in its current form can create rather large set of different terrains and is only limited by generation procedure that we are using for different terrain types. Currently we use diamond-square algorithm for that purpose. In the

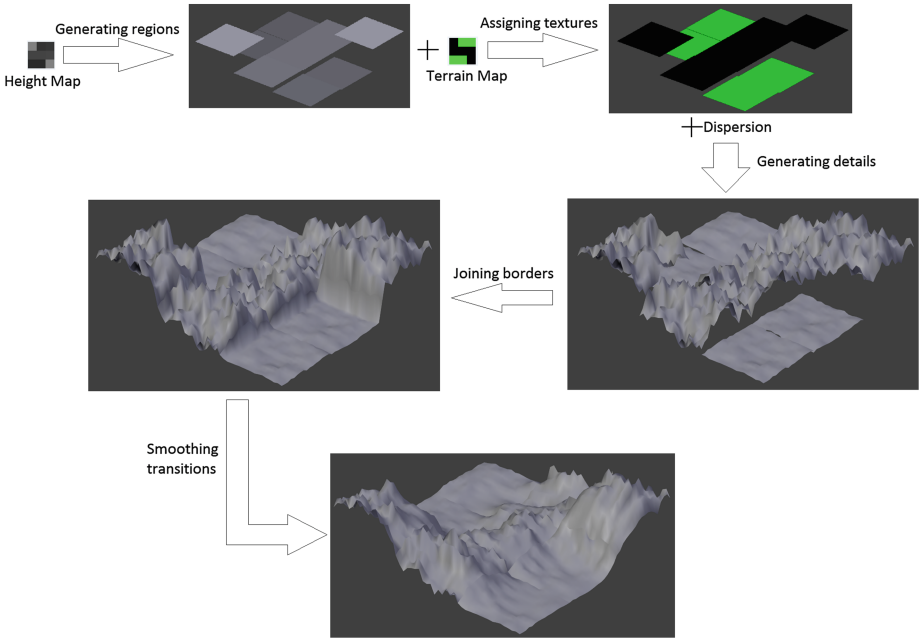

Fig. 4. Algorithm overview

future we would like to add different and more complex procedures for generating terrain.

Another issue is that our joining procedure does not provide satisfactory results, since from time to time, borders between regions are still visible. We would like to work on that problem and try some other solutions, like genetic algorithms, or more complex, mathematical functions than the one we are currently using.

Finally, we would like to add some procedures to further add details to created terrain (like vegetation, rocks, buildings and other).

6 Conclusions

In this work we presented procedure for creating 3D terrain by processing set of schematic maps, and user-defined properties. Terrains obtained during generation process could not only be easily adjusted at different parts of their creation, but also are suitable base for further modifications. At the same time any adjustments can be performed without further actions, either by editing maps, or modifying terrain itself, since our output areas are editable Blender objects.

Our algorithm is at early stage of development and still has some drawbacks, like to clear borders between terrain tiles, or insufficient base of generation algorithms. We plan to address those problems.

Even in this form our procedure creates interesting and playable terrains, that can also be easily used in computer games and further adjusted by designers, providing suitable base for future research.

References

1. Ebert, D.S., Kenton Musgrave, F., Peachey, D., Perlin, K., Worley, S.: Texturing and Modelling: A Procedural Approach, 3rd edn. Morgan Kaufmann Publishers Inc., San Francisco (2002)
2. Greuter, S., Parker, J., Stewart, N., Leach, G.: Real-time procedural generation of 'Pseudo Infinite' cities. In: Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia (2003)
3. Hendrix, M., Mejer, S., van der Velden, J., Iosup, A.: Procedural content generation for games: a survey. Commun. Appl. ACM Trans. Multimed. Comput. **9** (2013)
4. Linden, van der R., Lopes, R., Bidarra, R.: Designing procedurally generated levels. In: Proceedings of IDPv2 - Workshop on Artificial Intelligence in the Game Design Process (2013)
5. Merrell, P., Manocha, D.: Model synthesis: a general procedural modeling algorithm. IEEE Trans. Vis. Comput. Graph. **17**(6), 715–728 (2011)
6. Prachyabrued, M., Roden, T.E., Benton, R.G.: Procedural generation of stylized 2D maps. In: Proceedings of the International Conference on Advances in Computer Entertainment Technology (2007)
7. Prusinkiewicz, P., Hammel, M.: A fractal model of mountain with rivers. In: Proceedings of Graphics Interface, pp. 174–180 (1993)
8. Raffe, W.L., Zambetta, F., Li, X.: Evolving patch-based terrains for use in video games. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (2011)
9. Raman, S., Jianmin, Z.: Efficient terrain triangulation and modification algorithms for game applications. Int. J. Comput. Games Technol. **2008**, 5 (2008)
10. Roberts, G., Balakirsky, S., Foufou, S.: 3D reconstruction of rough terrain for USARSim using a height map method. In: Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems (2008)
11. Roden, T., Parberry, I.: From artistry to automation: a structured methodology for procedural content creation. In: Rauterberg, M. (ed.) ICEC 2004. LNCS, vol. 3166, pp. 151–156. Springer, Heidelberg (2004)
12. Santos, P., Toledo, de R., Gattas, M.: Solid height-maps sets: modelling and visualisation. In: Proceedings of the ACM Symposium on Solid and Physical Modelling (2008)
13. Smelik, R.M., Galka, K., Kraker, de K.J., Kujiper, F., Bidarra, R.: Semantic constraints for procedural generation of virtual worlds. In: Proceedings of the 2nd International Workshop on Procedural Content Generation in Games (2011)
14. Smelik, R.M., Kraker, de K.J., Groenewegen, S.A.: A Survey of procedural methods for terrain modelling. In: Proceedings of the Workshop on 3D Advanced Media in Gaming and Simulation, pp. 25–34 (2009)
15. Smelik, R.M., Tutenel, T., Bidarra, R., Benes, B.: A survey on procedural modelling for virtual worlds. Comput. Graph. Forum (2014)
16. Smelik, R.M., Tutenel, T., Kraker, de K.J., Bidarra, R.: A proposal for a procedural terrain modelling framework. In: Proceedings of the 14th Eurographics Symposium on Virtual Environments (2008)

17. Smelik, R.M., Tutenel, T., Kraker, K.J., Bidarra, R.: Declarative terrain modelling for military training games. Int. J. Comput. Games Technol. **2010**, 11 (2010). Article no. 2
18. Smelik, R.M., Tutenel, T., de Kraker, K.J., Bidarra, R.: A declarative approach to procedural modelling of virtual worlds. Comput. Graph. **35**(2), 352–363 (2010)
19. Blender application web page (2015). http://www.blender.org/. Accessed: 2 February 2015