Feature-Based Procedural Generation of Adjustable Game Content

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

This paper describes a method, for procedural content generation, allowing alteration of object during generation process, as well as further modifications of object after it was created. Most of existing algorithms either require adjustment of unintuitive data and parameters, or don't allow for further alteration of created object. Such outcomes are unsatisfactory for most of game designers. We offer our solution for this problem and identify some possible future directions of our research.

Key words: computer games, procedural content generation.

Introduction

Game worlds have grown tremendously in detail and visual realism. Some of them are vast, others fascinate with incredible level of detail and realism, presenting beautiful landscapes and believable settlements.

At the same time, all of those game worlds require considerable amount of time to create and adjust. Most of game objects, including landscapes, are created by hand, taking days if not weeks to complete. More and more often game designers need to either reduce level of detail for particular objects, or decrease amount of those objects.

Procedural content generation is a vast topic, containing series of procedures, allowing creation of different parts of game worlds. Those procedures allow for fast and detailed generation of specific content, in huge amounts, usually much faster and with grater amount of detail, than any human designer can maintain.

Unfortunately, most of those methods, has some major drawbacks, at least from designers point of view. While detailed and suitable for their defined application, those algorithms require specific, and usually unintuitive data. At the same time object, once generated, leaves very little place(if it does at all)for any modifications of its appearance or adjustment of details. Usually any modification requires regenerating entire object – an operation, that with complex models can take considerable amount of time, without guaranteeing better results.

In this paper we present an approach for procedural content generation, allowing not only easy maintaining of object data, but also adjustment of different parts of this object, or it's corresponding textures and additional details. In our work we consider both manual adjustments, as well as regenerating or automatically correcting selected parts of object. As a test platform for our algorithms we use Blender 2.68.

The rest of the paper is organized as follows. In section 2, we review other works, related to our research. Section 3 presents an overview of our method and describe obtained results. In section 4 we discuss some possible future adjustments and research directions. Finally we conclude our work in section 5.

Related work

Procedural generation of various elements has a significantly long tradition in the field of computer graphics. In this section we analyze previous works related to our research goals and methods, as well as most popular algorithms for procedural generation of different 3D virtual words elements. For detailed study of various procedural generation procedures see [1, 2].

Object generation and modification

Building terrain is one of most obvious applications of procedural content generation in many fields, computer games included. Therefore generating various areas with different properties and constraints is one of better examined subjects.

In article [3], authors describe an algorithm for generating various terrains, that meet specific, selected by the designer, properties, using semantic constraints. For example, one might decide, that two points on the map need to be connected by road, or stay in clear line of sight.

[4] presents an integration of procedural modeling and manual editing, allowing user high level of control over

generated content, and easy way of regenerating and adjusting created terrains(like regenerating a road, so that it will pass through nearby city).

[5] describes an application for creating military training simulations, allowing for easy and intuitive terrain modifications, as well as fast creation of maps and their properties.

Article [6] presents method of generating terrain, using Delaunay triangulation for maintaining mesh complexity appropriate to required level of detail. Algorithm also contains stenciling and stitching methods, for modifying created terrain by adding objects such as roads, tracks, etc.

While generating game content, it is equally important to maintain terrain correctness, especially if after producing terrain, player should be able to reach certain places. In [7] authors presents a method for evaluating level quality, based on genetic algorithm and some chosen properties.

There are also many other methods, for generating various terrains, based on height maps, L-systems, genetic algorithms or other methods, giving satisfactory and detailed results [1,2,8], as well as variety of combinations of above. For example, in [9] authors present a method, to generate pseudo-infinite cities, using seeds as a base for content generation. Another interesting issue is dungeon generation, presented in article [10]. This problem is rather specific and contains many issues, that other terrains do not posses. At the same time, the way most dungeons are built, allows for better control in generating them, giving interesting results.

Terrain generation is only one aspect of creating beautiful and believable terrains. Next step is generating various objects to place through the scene, such as rocks [11] or vegetation [13].

Another interesting issue presented in [13] is manipulating object mesh, both globally and locally, for animation purposes. Described method givers realistic results for presented examples, and compares them without other existing methods.

Geometry images and textures

Geometry images can be defined as a 2D representation of 3D model shape. There are various methods corresponding to generation of those images, as well as their application to object modeling.

In [14] authors describe fast and scalable method containing procedural geometry mapping and ray-dependant grammar development, for generating detailed geometries in render-time, in this case applied for building facades.

Another issue is generation and application of height maps. While elevation maps can be quite adequate for describing ground shape, they certainly have some drawbacks. In [15] authors describe method for representing complicated geometries, using solid height-map sets, which allow representation of object with many overlapping layers and multi-level terrains.

Textures are another, quite widely described problem, considering both their generation, and application to different objects. For further reference in this topic see [1, 8].

Constraints and layout solving

Constraints, in procedural content generation, can be tools, to define desired properties of generated objects. Based on object type, they can define some terrain properties [3, 4, 7], or characteristics of an object, that is later placed on scene [16].

While generating some simple geometry is not very complicated, modeling more complex and detailed objects is quite a different case.

One of problems in this type of generation is large amount of calculations required to obtain satisfactory results, especially if we want to produce results in real time. Other is way of representing and maintaining large amounts of data, required to such operations [14]. Some known methods of generating such objects involve fractals, or other mathematical functions [8], as well as L-systems or genetic algorithms [7].

Placement of those objects on generated terrain, both in building interiors, and open spaces is another issue.

In [16] authors describe a method for placing objects in the building interiors, based on some constraints assigned to objects, as well as their placement relatively to other objects on the scene, or such things as walls, doors or other elements.

There are also methods for creating and placing object, based on some example presented by the user [17]. Described algorithm examines different dependences and properties of given object, such as placement of certain parts, or their corresponding dimensions, and relying on such data, creates more complex structures.

Placing of some objects in game, i.e. enemies, can be modeled by collecting data on player experience, and reacting accordingly [18], or by defining some patterns and rules for placing those objects [19, 20].

Algorithm

In this section we present concepts concerning our methodology, as well as it's current implementation. As a test platform for our algorithms we use Blender 2.68. We chose this 3D modeling application firstly because it contains complete python interpreter, allowing us to easily add needed functionality. Second reason is that options already available are sufficient base for our research.

Method

Our method is meant for generation of objects with main application in computer games, that are both complex and adjustable by the game designer at every point of their generation, as well as after obtaining them (see Algorithm 1).

Our algorithm consist two main parts. First one describes division of a scene depending on predefined steps.

Currently we consider only equal intervals, due to simplify needed calculations. Since we are operating in 3D space, we divide our scene in cuboids of appropriate size (after calculating our intervals). This allows us for both easy generation, as well as fragmentation to smaller regions, when we want to edit part of it. Moreover, by dividing space in such a way, we are not restricting number of different kinds of objects, that our algorithm can generate. Example of mesh, containing generated vertices, already assigned to faces is shown in Fig. 1.

Algorithm 1. Object Generation

```
Require: Width, Length, Height, StepW, StepL, StepH, Type
  Generate Intervals(Width, Length, Height, StepW, StepL, StepH)
  for all Intervals do:
    Calculate Height Range(Type)
    Generate Vert
    Save Vert to text File
  end for
  Check user input
  for all Verts in File do:
    Assign Verts to Faces
    Draw Faces
  end for
  Check user input
```

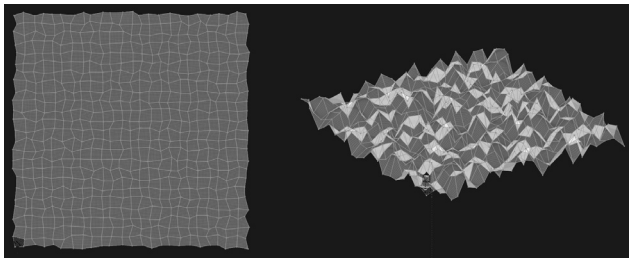

Fig. 1. Vertices and faces of generated object

Algorithm 2. Locating vertices

```
Require: Intervals
  if Type = Levelled then
    for all Intervals do:
      Select Intervals in Level
      Generate vertices at given Level with dispersion
    end for
  end if
  if Type = Rising then
    for all Intervals do:
      Select Intervals in Level
      for all Intervals in Level do:
        Generate vertices at given Level with dispersion
        Rise Level
      end for
    end for
  end if
  if Type = Island then
    if Location = Centre then
      Level = Max
      for all Intervals do:
        Select Intervals in Level
        Generate vertices at given Level with dispersion
      end for
    else
      Level = Max – LevelStep (where LevelStep=Max at borders)
      for all Intervals do:
        Select Intervals in Level
        Generate vertices at given Level with dispersion
      end for
    end for
  end if
```

Second part of our algorithm consist of generating vertices, in alignment described by object type. Depending from created content, different cuboids will contain information about placement of vertices inside considered model.

Algorithm 2 presents application of our method for generating few types of terrain: leveled plains or hills(with altitude at stable level), downward plains or hills(with altitude steadily rising or declining in one direction), and island type of terrain, with edges of object located at lowest level, and its middle at the highest point.

For each type of terrain, we first select cuboids from the scene, that will contain information about object vertices, and then we generate them. Depending from given requirements, such generation can be completely random, specified by some predefined properties, or even introduced by user.

At the end of procedure we save obtained results in text file. We chose such solution, because we wanted to be able to easily move our results between computers, without the need to copy entire blender file. Another reason is that „.txt" format allows not only simple file relocation, but also leaves possibility for future adaptations of our method for other platforms. Another advantage is that such files are considerably small in size.

Our approach is based on fact, that while procedural content generation is getting more and more accurate, designers still might want to modify some elements of created objects, either by regenerating fragments of model, or by manual adjusting them. Therefore our approach focuses exactly on that.

At every key point of object generation we leave place for some user adjustments, i.e. after generating points or faces, also after generating and assigning texture or adding any other element. Such fragmentation allows for better control of obtained output, as well as speeds up entire generation process - it's much faster to move few vertices, than to regenerate entire object, until we get satisfactory output. Another advantage is that, when results are not satisfactory, we can easily determine which step of generation is at fault and adjust it accordingly.

Implementation

Object generation algorithm presented in previous section is rather general and can be applied to any object, by simply adjusting appropriate functions, but after generating points and faces we use some of built in Blender functionality, to proceed. Currently our implementation contains generation of basic terrain with some randomly placed elements (rocks and grass), as presented in Algorithm 3.

Algorithm 3. Object adjustment

```
Require: Faces
  for all Faces do
    use subdivision modifier
    use smooth modifier
    use displacement modifier(displacement texture)
    Check user input
    Assign texture(object texture)
    Check user input
  end for
  Add objects to scene(list of objects, list of rules)
  Check user input
```

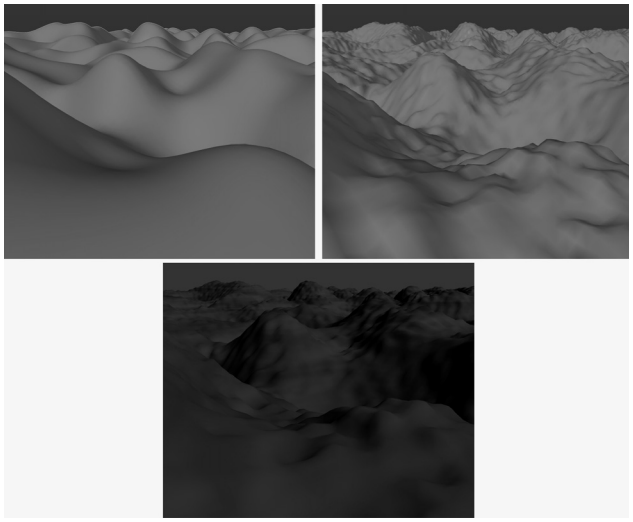

Fig. 2. Created object(top left), after appending displacement map through appropriate modifier(top right) and with assigned texture(bottom).

After a set of vertices is created, we then use some of built-in Blender functionality, documentation of which can be found in [21] in support section, to assign created vertices to faces and generate entire terrain from those faces.

Representation of displacement at given terrain can be resolved in many ways. In our approach we decided to use another of Blender tools: displacement modifier. Since Blender scripting language allows easy access to most of applications functionality, we just need to call appropriate procedure for created object

In our case, we are not using displacement map as representation of altitude through entire object, but rather want to modify some of places across the objects, without affecting general shape. Therefore using height maps in their basic form may be insufficient. We then decided to use displacement modifier, to achieve exactly that. Since that method may operate on user defined texture, it allows both simple generating of some believable shapes, as well as easy separation of parts of object with different properties, by simply assigning different texture. Together with smooth modifier, we achieved some interesting results. Example terrain, both before (top left) and after (top right) appending displacement modifier is presented at Figure 2.

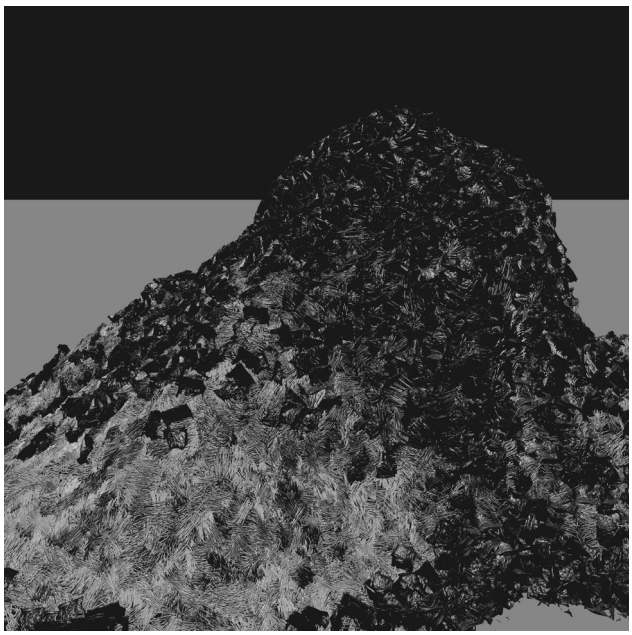

Fig. 3. Simple terrain with grass and rocks generated using our method

After creating and applying appropriate displacement for object, we then assign actual texture. Images used in this application can be both, generated procedurally or created by user, as long as it is a file, saved in format supported by Blender application. Therefore we can generate displacement map with any chosen properties. Currently we are using some of procedural textures available in application. Since they are generated directly into application, they also allow to follow any introduced changes in real time, without the need to load and reload newly generated texture. Fig. 2 (bottom) presents fully generated terrain with assigned simple texture(created using Blender tools).

After creating object with texture, we then place some simple objects across the scene, using Blender particle system. For this functionality to work we need to assign weights to our object, deciding placement of different types of objects we add to the scene.

At this point we generated whole, fully modifiable object. Although we predefined some of it's properties, all of them can be changed during any step of generation process, as well as at the end of algorithms operations. Example terrain is presented at Fig. 3.

Future work

The combination of procedural generation and manual editing can greatly improve workflow of game designers. First attempts have already been made (see [5]), but they still require a lot of work. Although our approach is still in early stage of development and requires considerable amount of work it still promises great results in the future.

We plan to incorporate more available terrains, as well as append generation of other objects, like buildings or some of more characteristic, in-game elements.

Another of our goals is to append few sets of various constraints, to allow better adjustment of created models. One of functionalities we want to incorporate are rules for connecting different terrains, and constraints for placing different objects (plants, rocks, buildings), through the scene, other than using weight maps.

In this case we were using some built-in Blender tools, for generating textures and modifying object but we also consider appending some external methods for this tasks, to achieve grater level of control, as well as to improve overall performance. We would like to at least add a method for creating texture and displacement map, that would be based on basic terrain shape, or other properties, specified by user.

We also plan to work on our data management algorithm. Currently it allows only for even intervals along each axis, and requires that each cuboids contains only one vert. We would like to explore other possibilities and compare them with our current solution.

Another of our goals is to adjust our method for other 3D modeling environments.

Conclusions

In this paper we presented a method for procedural generation of objects, that are both complex and adjustable at every step of their creation.

Out program was created in Blender application and uses some of it functionality.

Our program should be considered mainly as a base for further research. Even in such basic shape it still allows for creation of elements, that can be used in computer games. Currently number of available object is rather limited, but our data representation and generation algorithm do not restrict types of objects, that can be created. Another big advantage is that our method allows for great amount of object adjustments, both by user and algorithmic regeneration of its parts.

There are still very few procedural algorithms, that allow for modification of created content, that does not contain re-generating an object and presented method is sufficient for this application.

Bibliography

[1] Hendrikx M., Mejer S., Velden van der J., Iosup A.: Procedural content generation for games: a survey. ACM Trans. Multimedia Comput. Commun. Appl., Volume 9 Issue 1(2013)

[2] Smelik R. M., Tutenel T., Bidarra R., Benes B.: A survey on procedural modelling for virtual worlds. Computer Graphics Forum(2014)

[3] Smelik R. M., Galka K., Kraker de K. J., Kujiper F., Bidarra R.: Semantic constraints for procedural generation of virtual worlds. Proceedings of the 2nd International Workshop on Procedural Content Generation in Games(2011)

[4] Smelik R. M., Tutenel T., Kraker de K. J., Bidarra R.: A declarative approach to procedural modelling of virtual worlds. Computers and Graphics, Volume 35, Issue 2, 352--363(2010)

[5] Smelik R. M., Tutenel T., Kraker de K. J., Bidarra R.: Declarative Terrain modeling for Military Training Games. International Journal of Computer Games Technology(2010)

[6] Raman S., Jianmin Z.: Efficient Terrain Triangulation and Modification Algorithms for Game Applications. International Journal of Computer Games Technology(2008)

[7] Mourato F., Santos dos M. P., Bidarra R.: Automatic level generation for platform videogames using genetic algorithms. Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology(2011)

[8] Ebert D. S., Kenton Musgrave F., Peachey D., Perlin K., Worley S.: Texturing and Modelling: A Procedural Approach, (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2002)

[9] Greuter S., Parker J., Stewart N., Leach G.: Real-time procedural generation of 'pseudo infinite' cities. Proceedings of GRAPHITE (2003)

[10] Linden van der R., Lopes R., Bidarra R.: Procedural generation of dungeons. Accepted for publication in IEEE Transactions on Computational Intelligence and AI in Games TCIAIG(2014)

[11] Dart I. M., Rossi De G., Togelius J.: SpeedRock: procedural rocks through grammars and evolution 11. Proceedings of the 2nd International Workshop on Procedural Content Generation in Games(2011).

[12] Prusinkiewicz P., Lindenmayer A.: The algorithmic beauty of plants. New York, NY, USAL Springer-Verlag, (1990).

[13] Yoshiyasu Y., Yamazaki N.: Pose space surface manipulation. International Journal of Computer Games Technology(2012)

[14] Marvie J. E., Gautron P., Hirtzlin P., Sourimant G.: Render-time procedural per-pixel geometry generation. Proceedings of Graphic Interface(2011)

[15] Santos P., Toledo de R., Gattas M.: Solid Height-maps Sets: modelling and visualisation. Proceedings of the ACM Symposium on Solid and Physical Modeling,(2008)

[16] Tutenel T., Bidarra R., Smelik R. M., Kraker de K. J.: Rule based layout solving and its application to procedural interior generation. Proceedings of the Workshop on 3D Advanced Media in Gaming and Simulation(2009)

[17] Merrell P., Manocha D.: Model Synthesis: A General Procedural Modeling Algorithm. IEEE Transactions on Visualization and Computer Graphics, vol.17(2011)

[18] Lopes R., Hilf K., Jayapalan L., Bidarra R.: Mobile adaptive procedural content generation. Proceedings of Workshop on Procedural Content Generation for Games(2013)

[19] Dahlskog S., Togelius J.: Patterns and procedural content generation: revisiting Mario in world 1 level 1. Proceedings of the First Workshop on Design Patterns in Games(2012)

[20] Smith G., Othenin-Girard A., Whitehead J., Wardrip-Fruin N.: PCG-based game design: creating Endless Web. Proceedings of the International Conference on the Foundations of Digital Games(2012)

[21] Blender home page: http://www.blender.org/ (30.09.2014)

MSc Izabella ANTONIUK *– MSc (2012), PhD student at Institute of Computer Science, Warsaw University of Technology. Research interests: computer science and information technology, coputer graphics, procedural generation and artificial intelligence for application in computer games.*

Prof. Przemysław Rokita *– MSc (1985), PhD (1993), DSc (2000); Professor at the Institute of Computer Science of the Warsaw University of Technology. Member of SPIE, ACM, IEEE. Recipient of the Minister of Education Award (1995); Rector's Award in Science (2001); Laureate of the Golden Chalk Distinction for excellence in teaching – WUT (2005), (2006); Head of Computer Graphics Division (2013-); Member of the Faculty Council (2000-); Member of the Faculty Council Committee for Scientific Research (2008-); Research interests: computer science and information technology, digital image processing, computer graphics, image perception; Previously affiliated as visiting scientist and professor at: the Max-Planck-Institut für Informatik – Computer Graphics Department (Germany), the University of Aizu (Japan), Hiroshima Institute of Technology (Japan), Hiroshima Prefectural University (Japan), Imperial College of Science, Technology and Medicine (United Kingdom); Member of Program Committees and reviewer for many international conferences and journals, including: IEEE Computer Graphics and Applications, The Visual Computer, Real-Time Imaging, Opto-Electronics Review, Journal of Imaging Science and Technology, IEEE Transactions on Circuits and Systems for Video Technology, IEEE Transactions on Multimedia, ACM Siggraph, Eurographics, High Performance Graphics; Expert and consultant at National Centre for Research and Development, National Science Centre, Ministry of Science and Higher Education.*