From Artistry to Automation: A Structured Methodology for Procedural Content Creation

Timothy Roden, Ian Parberry

Department of Computer Science & Engineering, University of North Texas
P.O. Box 311366
Denton, Texas 76203-1366 USA
roden@cs.unt.edu, ian@cs.unt.edu
http://www.cs.unt.edu/~roden/esrg/researchgroup.htm

Abstract. Procedural techniques will soon automate many aspects of content creation for computer games. We describe an efficient, deterministic, methodology for procedurally generating 3D game content of arbitrary size and complexity. The technique progressively amplifies simple dynamically generated data structures into complex geometry. We use a procedural pipeline with a minimum set of controls at each stage to facilitate authoring. We show two examples from our research. Our terrain generator can synthesize massive 3D terrains in real-time while our level generator can be used to create indoor environments offline or in real-time.

1 Introduction

In the short history of computer games the primary technique used in the development of computer game content has been artistry. Game content, including 3D models, bitmapped graphics, levels and audio, have all been, for the most part, handcrafted by artists and designers working with a variety of software, often custom created. This approach has enabled game developers precise control over their creations in a manner similar to the way in which feature films are created. In fact, many in the game industry have long predicted that game development would increasingly mirror film development. Due to several compelling factors we believe this trend is at an end. Procedural content creation will soon become the dominant form of content creation for games software.

1.1 The Limits of Art

Handcrafted game content currently has at least four drawbacks. First, advances in technology mean artists need increasingly more time to create content. Secondly, handcrafted content is often not easy to modify once created. In a typical game development environment, game content is created at the same time that programmers are working on the game engine (rendering, networking code, etc.). Changes in specification or design of the game engine can dramatically alter technical

requirements for content, making content already produced obsolete. Thirdly, many widely used content creation authoring tools output content in a different format than that used by proprietary game engines. Developers typically build conversion utilities that can suffer from incompatibilities between what artists view in an authoring tool and how the content appears in the game engine. This difference can lead to repeated create-convert-test cycles which can be costly when the conversion process is time-intensive, such as creating a BSP tree from the output of a level editor. Finally, interactive games have the potential to become much more expansive than even the most epic film. If games are to reach this potential then humans can no longer continue to function as the predominant content creators.

1.2 Related Work

The use of procedural techniques in 3D graphic applications, including games, is not new. Particle systems have been used to model a variety of effects including smoke and fire. Procedural textures have also gained in popularity due to the work of Ken Perlin and others [1,7]. Procedural game textures have primarily been created offline using tools such as Darkling Simulation's Darktree, which allows the author to connect together a network of small procedural algorithms to generate the final texture [2]. With the advent of programmable graphics cards procedural textures will increasingly be synthesized dynamically. Researchers at MIT have experimented with systems to synthesize architectural 3D geometry from 2D floorplans [3] as well as methods to automatically populate buildings with furniture [4]. Inspired by studies into predictable random number generation and infinite universe techniques [5] procedurally generated virtual cities have been explored [6]. Fractal terrain generation and synthesis of vegetation has been the subject of much research, although most work has focused on offline generation [1,8].

2 Procedural Content Creation

Procedural techniques hold great promise yet harnessing that power can be difficult. Procedural methods typically use a set of parametric controls that enable a procedure to generate many different outputs. To make a procedure more useful, additional controls can be added. While the power of a procedure may be enhanced in this way, the resulting interface can become overly complex. In the case of a human using the interface, coming up with good results from a powerful procedure often degenerates into an authoring processing of trial and error. The challenge that is shaping the evolution of procedural content creation is to apply procedural techniques efficiently to meet demands for game content that is increasing both in quality and quantity.

An important premise of our procedural content creation systems is that we want to begin by generating very simple data. The data is then amplified in one or more steps, making use of predictable random number generators. Our terrain generator begins by synthesizing a relatively small 2D map while our level generator begins by creating a simple undirected graph. By using simple data as the foundation for the

content we wish to generate, we achieve several goals at once including, 1) we can quickly generate a simple version of the content which allows our algorithms to run in real-time, 2) we achieve a significant savings in storage requirements for the content since we only amplify that portion of the data needed at any given time, 3) the set of amplified data acts as a hierarchical representation that can be useful for a variety of processing such as level of detail, visibility determination and path finding, and, 4) data sharing becomes more efficient since only top-level data need be exchanged since more detailed data is available via the amplification process.

We use procedural pipelining as an efficient, controllable methodology for building procedural content creation systems. A primary goal is to structure the pipeline so that individual procedures can be constructed with the minimum number of controls necessary and each procedure can be authored somewhat separately from other procedures. Such systems exhibit a high degree of reusability and enable fast authoring.

3 Terrain Generation

The motivation behind our terrain generator is to dynamically synthesize massive 3D maps in real-time. For example we want the ability to create a 100 square mile map where individual vertices are 10 feet apart. Given an estimated storage size of 20 bytes per vertex the resulting data would occupy over 67 terabytes for the textured height mesh alone. Even more space would be needed to store terrain features such as trees. This requirement is prohibitive for a personal computer. However, we observe that in many games with large maps a player often moves about in a small geographical area and typically explores a large map incrementally.

Given these observations and the limitation of incremental exploration, our terrain generator first creates a high level representation of the map and creates detail as needed according to the player's location. Data at the finest level of detail, including low-level terrain features such as vegetation, are only created in the immediate vicinity of the player. As exploration proceeds, detail is generated as needed using a caching scheme that purges data unlikely to be needed in the near future.

Our terrain generator uses a rule-based system to first create a high-level 2D representation of the map (Fig. 1). In contrast to techniques that directly create 3D content we begin with 2D data for several reasons. First, we want to synthesize our terrain at the highest level of abstraction possible for efficiency reasons. Major terrain features on the 2D map are easily encoded using a minimum set of rules and include ocean, river, swamp, grass, forest, desert, hills and mountains. Second, since many of these terrain features naturally imply elevation, height data is essentially created for free. Third, our 2D map forms the basis for both synthesized elevation data and low-level terrain features such as vegetation.

Parametric controls in the 2D map generator include number of continents, ratio of land to ocean and percentage of various terrain types (river, swamp, forest, and hills)

to amount of total land. Placement of terrain is governed via separate rules for each terrain type that attempt to mimic terrain patterns found on earth. For example, hills are placed along randomly generated fault lines with dense groupings of hills transformed into mountains. Currently, we are experimenting with scale of 2560 square feet per 2D map square (approximately one half square mile). A 100 square mile map thus requires a 2D map approximately 200 x 200 squares.

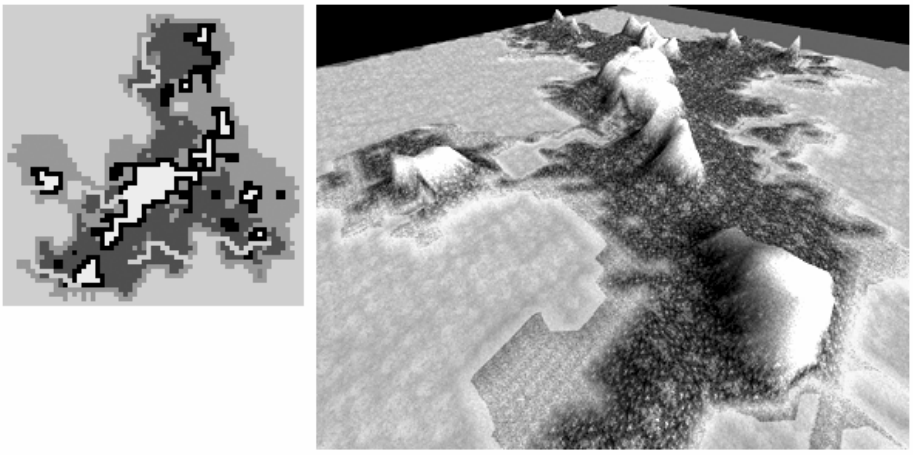

Fig. 1 'Non-viewable' top-level coarse 3D height field (right) amplified directly from auto-generated 64 x 64 terrain map (left). Note: 3D textures have been applied for illustration only.

The second procedure in our terrain generation pipeline is a rule-based system that extrudes the 2D map into a high-level coarse 3D representation (Fig. 1). Parametric controls include an elevation scaling factor as well as controls for setting approximate elevation ranges for low-level terrain features such as different varieties of trees, grasses and rocks, altitude for snow, etc. A set of internal rules attempts to mimic earth-like elevation patterns. The coarse 3D map forms the root of a sparse quadtree with as many levels in the tree as needed to synthesize the required detail. The bottom level of the tree and several levels above form the viewable levels, enabling continuous-level-of-detail rendering.

The third procedure in the terrain pipeline is a recursive algorithm to subdivide a coarse 3D mesh into a series of higher detail meshes. The procedure uses a simple fractal method called midpoint displacement that subdivides an edge into two edges where the elevation of the midpoint is randomly chosen between the elevations of the endpoints of the original edge. Multiple subdivisions of the coarse map also serve to break up what would otherwise be unnatural looking contours.

Additional procedures add low-level terrain features to the viewable layers of the 3D mesh. As in previous procedures a set of rules attempts to mimic earth. For example a distribution of trees is based on several pieces of data. The 2D map provides a general description of the density of trees found in a larger square area of the map. Adjacent squares on the 2D map can also affect tree density including the presence of

adjacent forest or river squares. Elevation data can be used to select a suitable variety of tree. A set of parametric controls provides customization in tree type, size, density and growth patterns.

4 Level Generation

A game level is generally an area of Euclidean space with corresponding 3D geometry and associated objects contained within the space including geometric objects and sounds. It is not unusual for game levels to represent space enclosed entirely indoors such as a building or an underground maze. For our initial experiments we chose to generate indoor environments. Example environments our level generator can synthesize are underground mazes or interior compartments and rooms of a spaceship, among other things. Our motivation is to dynamically create levels of size and complexity typically found in current games.

The level generator begins by creating a simple undirected graph in three dimensions. This first stage in the level pipeline can be controlled with several parameters including topology and a set of constraints. Some of the basic topologies are tree, ring and star. Additionally, multiple sub-graphs, with different topologies, can be connected together to form hybrid topologies. Each node in the graph represents a portion of the actual level geometry. Constraints on graph generation are given as rules relating to one or more fixed nodes with parameters such as min/max distance from terminal nodes (entry/exits), fixed connections to adjacent nodes and prefabricated geometry to be used for rendering the nodes. In addition, constraints can enable sequences of nodes that can only be visited by the player in a certain order.

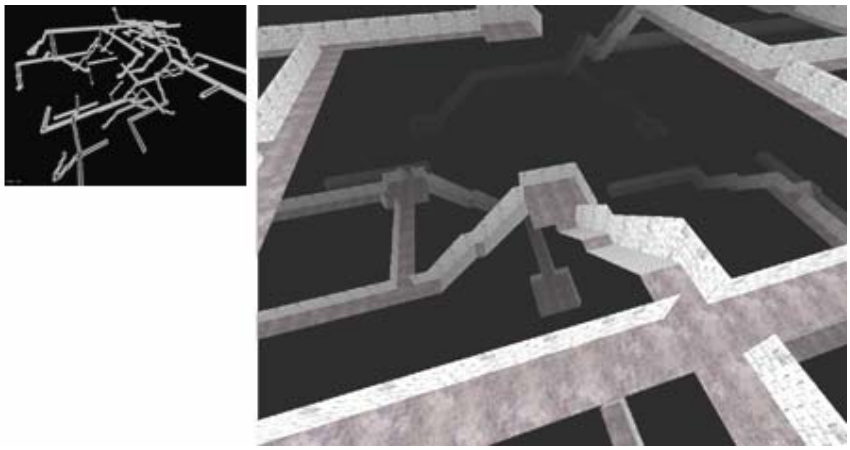

Fig. 2 Graph (left) amplified into full 3D geometry (right).

The next stage in the pipeline constructs basic geometry for each node (Fig. 2). We used a modeling program to build a limited set of prefabricated geometry sections that can be interconnected to produce a larger level. A similar method was used by Epic

Games with level designers manually fitting together the prefabs [9]. To construct the geometry for a node in the graph we gather the appropriate set of prefabs and combine them to create the final geometry for the node. We found that it was much easier to construct a very basic set of prefabs that could be combined to produce more detailed geometry than to create larger prefabs for every possible geometric node configuration. For example, our 'basic-dungeon' set consists of ten prefabs including corridors, stairs and rooms. Each prefab is textured in the modeling program. Completed node geometries are placed in a list that is instanced by nodes in the graph since multiple nodes in the graph may be structurally identical which allows them to share geometry. Additionally, at this stage of the generation, nodes may have been flagged with a constraint that forces their geometry to be assigned from a dedicated set of prefab enabling the level author to insert special rooms into the level.

The final stage in the pipeline involves adding objects to the level. Objects can include static 3D geometry in addition to movable 3D objects such as furniture. Sounds can also be added and would typically correspond to geometric features.

5 Conclusion

We believe the way forward for game developers to successfully incorporate procedural content into their games will begin by following a structured approach. Amplification of simple data structures in a pipelined architecture is a structured methodology that will enable procedural content creation systems to be built in a minimum amount of time, facilitate testing and debugging, and most importantly enhance authoring.

References

1. Ebert, David, et al: Texturing and Modeling: A Procedural Approach, 3rd edition. Morgan Kaufmann Publishers, San Francisco (2003).
2. Theodore, Steve: "Product Reviews: Darkling Simulations' Darktree 2," Game Developer Magazine, pp. 6-7 (March 2002).
3. Wehowsky, Andreas: Procedural Generation of a 3D model of MIT campus, Research Project Report, Massachusetts Institute of Technology Graphics Laboratory (May 2001). http://graphics.lcs.mit.edu/~seth/pubs/WehowskyBMG.pdf
4. Kjolass, Kari Anne Hoier: Automatic Furniture Population of Large Architectural Models, Master Thesis, Department of Electrical Engineering and Computer Science, MIT (2000).
5. Lecky-Thompson, Guy: Infinite Game Universe: Mathematical Techniques, Charles River Media (2001).
6. Greuter, Stefan, et al: Real-time Procedural Generation of 'Pseudo Infinite' Cities, in Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Austalasia and South East Asia, pp. 87 (2003).
7. Perlin, K: An Image Synthesizer, in Proceedings ACM SIGGRAPH, pp. 287-296 (1985).
8. DeLoura, Mark, ed.: Game Programming Gems, Charles River Media (2000).
9. Perry, L: "Modular Level and Component Design," Game Developer, pp. 30-35 (Nov 2002).