Procedural City Generation and Techniques for Game Designers

A Thesis Submitted to the Faculty of the Visual Effects Department
in Partial Fulfillment of the Requirements for the
Degree of Master of Fine Arts in Visual Effects
at
The Savannah College of Art and Design

Matthew John Fournier

Savannah, GA

© November 2014

Deborah R. Fowler, Committee Chair
Charles Shami, Committee Member
Stuart Robertson, Committee Member
# Table of Contents

List of Figures .......................................................................................................................... 1  
Thesis Abstract ......................................................................................................................... 2  
1. Introduction .......................................................................................................................... 3  
   1.1 Procedural Modeling ....................................................................................................... 4  
   1.2 Games ............................................................................................................................ 5  
   1.3 Pitfalls in Procedural Techniques .................................................................................. 7  
   1.4 Houdini Engine ............................................................................................................ 9  
2. Procedural Design ............................................................................................................... 10  
   2.1 Procedural Techniques for Game Design ...................................................................... 10  
   2.2 Handling Large Data Sets ............................................................................................ 11  
   2.3 Why use a Procedural Approach? ................................................................................ 13  
3. Game Design ...................................................................................................................... 13  
   3.1 Immersion in Games ..................................................................................................... 14  
   3.2 Play ............................................................................................................................... 15  
   3.3 Asset Management ....................................................................................................... 16  
4. Solution – Procedural City Generator Genesis ................................................................ 18  
   4.1 Importing Structure – Building Creation ..................................................................... 18  
   4.2 Road System ................................................................................................................ 20  
   4.3 Procedural Customization ............................................................................................. 21  
   4.4 Experiments and Practical Implementation .................................................................. 23  
5. City Engine ......................................................................................................................... 25  
6. Conclusion ........................................................................................................................... 25  
   6.1 Future Development ...................................................................................................... 26  
Appendix A: Tutorial ............................................................................................................... 28  
Appendix B: Genesis Building Generator Parameters ......................................................... 28  
Appendix C: Genesis Road Parameters ................................................................................ 30  
Appendix D: Genesis Road Intersection Parameters .......................................................... 30
Appendix E: Sidewalk Parameters - Work in Progress ................................................................. 31
Appendix F: Naming and Folder Structure .................................................................................. 31
Appendix G: Artist Survey .......................................................................................................... 32
Appendix H: Artist Survey .......................................................................................................... 33
Bibliography ................................................................................................................................ 34
List of Figures

Fig 1.1 Linear (Waterfall) Workflow ........................................................................................................... 3
Fig 1.2 Edge Magazines September 2014 Issue featuring No Man’s Sky .............................................. 5
Fig 1.3 Houdini Engine Workflow ................................................................................................................ 9
Fig 2.1 Genesis Building Modules ............................................................................................................. 11
Fig 2.2 Genesis Window Modules .............................................................................................................. 12
Fig 3.1 Procedural Weapon Examples ....................................................................................................... 15
Fig 3.2 Level of Detail with Poly Count ..................................................................................................... 16
Fig 4.1 In-game Render of Genesis Buildings ............................................................................................. 17
Fig 4.2 Yahoo Maps Image, Starting Point for Genesis ........................................................................... 18
Fig 4.3 Desired 2D Map Input for Genesis ................................................................................................. 18
Fig. 4.4 Procedural Window Module w/ GUI ............................................................................................. 19
Fig. 4.5 Genesis Building Module User Interface ....................................................................................... 19
Fig. 4.6 Genesis City Block Module User Interface ..................................................................................... 20
Fig. 4.7 Render of Roads and Traffic Props ............................................................................................... 20
Fig 4.8 Optimized Low-Poly Render of Road System ............................................................................... 21
Fig 4.9 Houdini Render of Three City Blocks ............................................................................................ 22
Fig. 4.10 Painting Textures on Finished Window Module ......................................................................... 22
Fig. 4.11 In-Game Render of Dock a Freighter Game .............................................................................. 24
Fig. 4.12 Arcade Box for Dock a Freighter Game ....................................................................................... 25
Modern films and 3D games are often set within large urban environments. Current techniques to create these environments are time-consuming, require expensive tools and involve enormous amounts of geometry, including buildings, roads, terrain and all their details. One alternative is to employ a procedural approach; Genesis is one such tool for the Houdini API that automates modeling, texturing and object placement while giving explicit control over “Hero” objects that require more detail.

Generating a city is challenging work; urban environments are often unique and influenced by their surrounding area. In modern games, the look of a city and its layout can vary based on geography, time periods, culture and population, all under the interpretation of an art director. The Genesis tool and proposed workflow will enable artists to quickly create these environments with the ability to manipulate individual assets. These assets are constructed in a procedural manner allowing artist to modify models, textures, topology and modules efficiently.

Keywords: Procedural modeling, level design, game design, Houdini, Unity, game art, procedural cities, procedural roads, city layout, procedural game design, procedural generation
1. Introduction

As technology improves, the demand for enhanced detail and realism in games and film has increased. The current solution is to hire more artists to meet this demand. These resources could instead be spent on enhancing game play with new innovative concepts.

Looking at the cost of games over the last several years, budgets have ballooned to well over $20 million per title. Big name titles such as Grand Theft Auto V, released September 2013, cost $265 million for development and marketing and employed over 1000 artists.¹ High-end content is getting increasingly expensive every year; the need to work smarter, not harder, rises with the cost of AAA titles.

How, as an industry, do we work smarter? A potential solution is to rely on a more procedural, cyclical workflow as opposed to a linear pipeline.

   <http://www.ign.com/articles/2012/07/30/are-aaa-hardcore-games-doomed>
Figure 1.1 illustrates a common linear workflow, but where this model fails is in its feedback loop. Once you advance to a new a step, from design to development for example, manipulating a part of the design in the development stage can be destructive to the other previous steps. Connecting these steps with a more cyclical interactive workflow will allow aspects of a game in any phase to change without negatively affecting other stages of the pipeline. Creating the models and geometry of a game can be one of the most time consuming steps in a traditional pipeline. Bringing down the cost of creating content using a procedural workflow can be seen at Kitfox Games. Tanya Short, Kitfox’s Creative Director, states, “The benefits of procedural generation to a small team continue to outweigh the risks -- we're unlikely to give up on that anytime soon.” One of the many benefits of a procedural workflow is the ability to make changes and adapt to an ever changing environment, ultimately lowering the cost of production.

### 1.1 Procedural Modeling

Simply put, procedural modeling creates 3D models from a set of rules. Instead of making everything by hand, a program handles the more repetitious tasks, automatically creating models for an artist. A procedural façade, for example, has several different textures, windows, doors and shapes. Changing the shape of the building would require UV's, textures and topology to be discarded and an artist to repeat these steps. An intelligent procedural model can be manipulated while keeping UV's, general topology and other data attributes from

---


being degraded by revision. In an article on the future of blockbuster AAA titles, Danny O’Dwyer of the production company GameSpot says, “This is why there are hundreds of people working on the Assassin’s Creed franchise. Because we demand that each door is painted uniquely and each civilian is wearing something different.” The demand for more realism in Assassin’s Creed was answered by hiring more artists. At what point does this demand break a production’s budget?

1.2 Games

Incorporating procedural techniques in games has been in practice for several years now. One of the most successful examples of this is Blizzard’s Diablo franchise. Diablo (1998) pioneered a procedural dungeon; the beauty of this approach was a player rarely experienced the same dungeon twice. Playing the same level twice would yield the player an environment with new power-ups and monster locations in a seemingly new experience. The dungeons were self-similar, keeping the user familiar with the setting, yet not.

---

Modern games have been building on this technology; the upcoming title *No Man’s Sky* (expected release 2015) features an entire universe with worlds that are light years apart. This is all procedurally generated in a process developer Hello Games is calling ‘an infinitely procedural universe’. This type of game would be impossible to create by hand. Started by four friends, Hello Games is a small studio that now employs twelve. Every world of *No Man’s Sky* has its own unique foliage and animals that are generated by algorithms. When entering a world for the first time, players are the first to experience that world; no one else can tell them where to get the best equipment or anything else about that world. Founder and programmer for Hello Games, Sean Murray states, “It happens all the time. We create these systems, layers and layers of simple systems. Together they create something that is chaotic, it’s almost impossible to predict. I create the algorithms that generate our terrain for instance, and I’m always seeing rock formations, or rivers, waterfalls, caves that I just didn’t know could exist ... like those floating islands that you see in our trailer. For creatures and life, it always surprises me where life will find a way to grow. You find these bacteria readings in the deepest caves of planets that you thought were totally sterile. It’s a little thing, but it blows my mind.”⁵ The success of *No Man’s Sky* is yet to be seen, but the attention this game has received before an official release date has been announced could be a positive sign.

*Minecraft* (2011) is a game whose procedural structure is a testament to its success. The game is an enormous sandbox style world that is procedurally generated with a random starting point and world. Random and procedural generation should not be confused with each

---

other. Procedural generation uses a set of refined algorithms to realistically distribute attributes. Randomization is simply that, random, with little control and is relatively unsophisticated. For example, in *No Man’s Sky*, the color of the sky changes on different planets. This color is not chosen at random; algorithms ask where that planet is in its solar system, does it have water and what kind of atmosphere. These factors are calculated by the algorithm to determine a color or small section of colors to choose from.⁶

Similar to Hello Games, *Minecraft’s* development started with a small team, one developer, Markus Persson. Persson backed his vision by starting the game company Mojung, which employed 48 artists and developers at its peak. This small staff vastly increased the profit margin for *Minecraft*, which led to its recent enormous buy out price by Microsoft of $2.5 billion. The average profit margin for AAA titles is about 3%.⁷ *Minecraft*’s success would not be possible without a procedural system in place.

### 1.3 Pitfalls in Procedural Techniques

Procedural game design can be used to increase workflow and lower cost of games. As we have seen in *Diablo*, it can also be used to help tell the narrative and enhance the player’s experience. Why was a similar technique not used in a blockbuster such as *Grand Theft Auto V* you may ask?

---


Procedural techniques often use randomization for distribution, if left unrefined, that randomness can leave object placement feeling purposeless to the player. For example, in early builds of Genesis, a simple random function distributed modules over a building’s façade, this lead to oddities, such as exterior doors on the fourth floor. Genesis now intelligently detects what floor each module is being placed and what modules are being used around it to add a level of realism. Players will detect unrealistic placement of objects, reminding them they are playing a game and ultimately disrupting play. Another concern about procedural design is repetition; the human eye is excellent at picking out patterns, even patterns intended to present an impression of randomness. Flagship Studios *Hellgate: London* (2007) is a prime example of poorly executed design and too much ambition. *Hellgate: London’s* failure caused Flagship Studios to close its doors. Their initial sales were not the issue: “As it turns out, *Hellgate* sold a respectable amount of copies in the US and Europe. But the game's business model depended on subscription revenue in the long term, and not enough of those sales translated into subscriptions.”

Plenty of people purchased the game, yet few signed up for a subscription. Reviewers widely criticized the game for its repetition, Steve Butts of IGN says "Your own enjoyment of Hellgate will naturally depend on your tolerance for the repetitious levels and your overall desire to find the perfect combination of skills and items that will allow you to take on the toughest of the game's challenges with ease. Unfortunately for Hellgate, our tolerance and desire levels just aren't what they used to be."

---


1.4 Houdini Engine

Side Effects’ Houdini is an industry leader in procedural workflows. It can handle large amounts of data and calculations in a unique, dynamic way. It is the perfect medium for a procedural city generator, particularly if the Houdini Engine plugin is utilized. The Houdini Engine is used for data integration between Houdini and a game engine, in this case Unity. Side Effects president and CEO Kim Davidson says, “The Houdini Engine makes it possible for more CG artists to experience the power and flexibility of our procedural workflow.” He continues to say, “In essence, we have turned Houdini into the world’s most powerful plug-in for any number of DCC apps.”

The Houdini Engine allows for game designers to increase the efficiency of their pipeline by having easy access to data between two software packages. This affects Genesis by removing one of the more tedious steps in a current game design workflow, exporting and importing geometry. Instead, the tool itself is exported as a digital asset or .otl file from

---

10 July 11, 2013 news article “Houdin Engine”
11 July 11, 2013 news article “Houdin Engine”
Houdini. The asset is used directly in the game engine to generate geometry, textures and even UV coordinates. The more traditional approach of creating geometry inside of a 3D application and exporting that geometry into a format that is compatible with Unity, inhibits the procedural nature of an efficient workflow.

Genesis is not yet fully compatible with the Houdini Engine though several key features maintain functionality. It would be ideal if Genesis was fully integrated with the Houdini Engine and is the desired result in the next build. Currently, Genesis can generate building models and UV values only inside of the Houdini Engine. This generation is slow as Genesis wastes computation time attempting to create textures, roads and props.

2.0 Procedural Design

As we have discussed, procedural design can be powerful but can also take away from a game's core mechanics. We have seen an array of examples of procedural design, from full dungeons to entire worlds. Other games, such as Boarderlands, use a procedural approach for their weapons, keeping all weapon designs unique and drop points unpredictable. We can see that these practices fail when the player’s experience is interrupted by repetition. When players detect objects placed by an algorithm, it reminds them they are playing a game, breaking immersion and inhibiting play. We elaborate on play and immersion in section 3.

2.1 Procedural Techniques for City Generation

Discussions of procedural design, are in reference specifically to environment and placement of objects and how procedural design pertains to game play. It represents the structural design as a set of interacting functions that pass data back and forth seamlessly. The
design aspect should aid in the placement or creation of the art, ideally enhancing the player’s experience. This differs from procedural modeling as it focuses on game design rather than models.

Modeling a city is a unique challenge; cities are large and complex. Architecture and layout can vary greatly within one block and be nearly identical with a clear pattern two blocks over. Pascal Muller, CEO of Esri, the developer of City Engine states, “The potential applications for a procedural creation range from research and educational purposes, such as urban planning and creation of virtual environments to simulation.” Cities are well suited to be generated through procedural modeling. Several aspects of cites, such as brick patterns or window size, have distinct patterns and repeating themes in building. Therein lies the challenge of developing a procedural city generator; certain aspects are self-similar while others are unique. The solution to this challenge is a procedural design yielding a certain level of control to the artist. This solution must remove the more tedious aspects of placing a plethora of geometry while maintaining customization. In rare cases, these systems can add to the play of the game, such as the replay ability of Diablo II or the seemingly endless content of No Man’s Sky.

2.2 Handling large data sets:

Genesis handles these large data sets by dispersing each set off into its own category: Buildings, Building Modules,

---

12 Parish, H Yoav. & Muller, Pascal. “Procedural Modeling of Cities”
Sidewalks, Roads, Parks and Parking lots. Each run independently of each other and can be modified without negatively affecting other categories. Genesis can be run multiple times over different city blocks to achieve self-similarity of buildings or uniqueness of a city block. Using proxy models on certain city blocks gives the artist the ability to visualize one block while working on another in full detail. Individual control of each building is provided while maintaining visualization of other buildings at lower polygon counts. Artists can then customize a single building without losing the essence of the rest of the city. This maintains an efficient workflow, encouraging creativity.

Furthermore, nearly all objects are modeled using modules. A modular design is an approach to modeling that divides the object into chunks, in this case, a section of the facade.

Genesis utilizes multiple levels of modular design to enhance performance inside of Houdini and the desired game engine. The shapes in Figure 2.1 represent a building module; these shapes have a series of attributes applied to them to generate the placement of each window module as shown in Figure 2.2. Each point holds the starting position of each window module and all the attributes to assign texture, window size, location and scale. These modules are discussed in detail in section 3.
2.3 Why use a procedural approach?

Little can be considered safe from change throughout a production pipeline. Being able to adapt to these changes in a graceful and efficient manner is not only important to being a successful artist, but also must be a feature of a well-designed procedural tool. From an artist's perspective, if a decision is made to change the setting of a level, that artist will have to go back through and model every building, decoration, street light etc. saving little from his or her previous work. This can be incredibly frustrating, leading to an unhappy artist and possibly rushed or poorly-executed art. If the tedium of placement, texturing, UVing and modeling could be handled for the artist, the lessened frustration should ultimately lead to a product of higher quality. This quick exchange between revisions and final output encourages an efficient workflow and lowers the cost of production.

3. Game Design

Designing games to accommodate for all the different play styles and their interrelationships in game mechanics can be complex. Poorly planned attempts to keep producers, supervisors and an ever-demanding customer base satisfied can lead to failure through tactless implementation of overall game design. A common pitfall in design is ignoring what motivates people to play games, to have fun. Salen and Zimmerman, authors of Rules of Play, elaborate by stating, “Why play a game that isn't fun? The computer and video game industry is continually spurred on by an audience hungry for ever-more spectacular games and ever-more meaningful interaction. People play games because they want to; game designers must create experiences that both feed and satisfy this sense of desire.” How can game designers overcome this pitfall?
Fundamentally, game designers create a set of rules in an interactive system to guide players in an activity. But, what motivates this activity? More importantly, how can designers motivate the player and how does the player’s motivation relate to a positive play experience?

### 3.1 Immersion in Games

It is important to have a better definition of immersion and play to better understand how design affects a player’s experience. Immersion, the subjective state of being deeply engaged in an experience, can take many forms dependent on player activity. In an article on immersion in games, Ernest Adams identifies three types of immersion: Tactical, Strategic and Narrative.

- **Tactical Immersion** is the moment-by-moment action. A player is faced with simple problems that must be solved in seconds with little thought. Found in fast paced games, First Person Shooters for example, this type is commonly referred to as “In the groove” immersion.

- **Strategic Immersion** is the intellectual involvement of seeking a path to victory or optimizing a situation to win. While being Strategically Immersed, a player is calculating how to best overcome mental challenges. Chess is the ultimate form of strategic immersion.

- **Narrative Immersion**, as in books or movies, the player get immersed when he or she cares for the characters. Players are motivated by the narrative and how a story progresses. Role Playing Games often incorporate Strategic and Narrative Immersion.  

---

13 Adams, Ernest 2004. “Postmodernism and the Three Types of Immersion”.  
The tools used to create Narrative Immersion, such as good storytelling, are quite different than the other types. Immersion can be achieved by many different tools in the game designer’s box, such tools include: a narrative game structure, character development, rich game worlds, challenges to overcome, fast paced twitch action, freedom of choice and a smooth learning curve. In procedural design, maintaining continuity throughout the design process will assist in keeping a player immersed in a rich game world. The instant a player notices a texture seam or identical facades, immersion is broken and the user’s experience is degraded.

3.2 Play

The goal of the game developer is to create an enjoyable experience or simply put, play. Play happens by participating in a set of rules that are controlled by the designer and enhanced by the artist. These rules of play are key in a player’s motivation. Artists and designers should focus on play above all else. Getting lost in technology or straying from the core mechanic of any given game will subtract from the activity of play. It is essential to not allow a workflow or tool to inhibit the core mechanics of play. Salen and Zimmerman state that "focusing on the play of a game's core mechanics is a good starting point for designing powerful
Diablo and Borderlands are excellent examples of how designers use a procedural design, for their dungeons and weapons respectively, to enhance play by providing a rich game world. If properly utilized, Genesis could assist in maintaining several immersion types. Spatial immersion could be achieved by creating a convincing setting. That setting could support cognitive immersion by giving the artist the ability to tweak puzzles or problems quickly, for instance, making sure that special power-ups are just out of reach with the exact button sequence.

3.3 Asset management

Creating assets for games comes with special challenges. Game engines must be able to calculate everything from geometry to lights, in real time. It should be clear by now that procedural tools and game optimization go hand in hand. Poly count, for example, is a large factor in how quickly a game environment is rendered. Controlling polygons is essential to

---

Salen, Katie & Zunnerman, Eric "Rules of Play" pg. 326
optimizing the performance of a game and avoids interruption of play. Testing the performance of complex game assets at different polygonal levels can be time consuming and affect topology and UVs. A procedural tool with this functionality, like Genesis, can be used to efficiently optimize a model without a negatively effect.

Rich game worlds are made up of characters, story and environments. These environments can be enormous. As rewarding as it can be to create these worlds, few artists are thrilled of the prospect at placing every traffic light, stop sign or bus stop. These assets have countless attributes associated with them depending on their usefulness in the game. Houdini is uniquely suited to organize these attributes and maintain their integrity through a fluid pipeline.
4. Solution, city generator Genesis

The procedural modeling tool Genesis runs from 2D maps that focus on the creation of cities. Genesis has a unique set of rules to model and texture a city. These rules can be controlled by the user to set a certain look to a city or manipulated to use custom modules. The creation and implementation of the Genesis tool in a game engine is the focus and end product of this research. The end export from Houdini is in .fbx format and is easily adapted to function inside of Unreal and Unity engines. Unity was selected to showcase Genesis’s capabilities due to its implementation of the Houdini Engine.

4.1 Importing Structures – Buildings Creation

Genesis generates building structures by importing a 2D map image. This can be drawn by hand or imported from a favorite online map software, such as Yahoo maps. A small utility in Nuke can generate the 2D images Genesis

Figure 4.2 - Yahoo maps screen grab, starting point for Genesis

Figure 4.3 - Desired 2D map input for Genesis
requires. Figure 4.2 shows the desired input for Nuke, and Figure 4.3 shows the ideal input for Houdini. The ability to simply draw the outlines of your structures allows for advanced control when designing a building’s shape. These maps are small and can be changed quickly. Houdini’s native compositing software can also be used to keep all work inside of one package.

This information is used to direct the shape of each building and to assign façade modules, texture and size, all of which can be replaced with modules designed by the user. A procedural module is provided to quickly generate the more common themes in modern architecture, such as windows.

The height of these buildings cannot be read from a 2D image; a random height is given at the start with controls to adjust the height. Other controls are used for optimization, assigning custom
materials and controlling the overall shape of each building beyond the defaults. Details of each structure are distributed based on user set frequency.

Using a procedural module in conjunction with the procedural placement and design of each building gives the user control of the randomness of each city block as well as custom controls to refine the look. These systems work together to allow for the dynamic placement of common architectural details. Genesis has the ability to construct the following details, all with control for scale, translation and texture: awnings, balconies, cornices, columns, fire escapes, inlays, ledges, windows, multiple windows and window sills.

A similar workflow is used for sidewalks, parks and parking lots. These systems are all quite rough and require further attention but are provided as a proof of concept.

4.2 Road System

Many procedural road systems are available with a variety of solutions; many use Houdini. Matching these
systems to our previously mentioned technique of drawn building shapes was tricky. This tool is less procedural and requires additional input than a 2D shape. Roads are generated by the user drawing curves; each curve generates a road. It’s recommended that each road have its own curve as more curves increases the accuracy of intersections. Traffic lights, bus stops, parking meters and other objects typically seen around a city are generated based on the length and width of these curves.

Similar to the way by which the buildings are generated, these objects are designed to be modular, allowing for custom geometry or geometry sets for each city block. Road width can be adjusted per road. The sensitivity for the length and width can be adjusted for placement. This stands as a separate tool and runs close to real time and can optimize output for game engines or produce a higher level of detail if needed. In our example, our entire road system is less than 5,000 polys.

4.3 Procedural Customization

A unique technique inside of Genesis is the ability to paint directly onto the final geometry to change attributes like façade types or texture. Results are near real time and focus on giving artists intuitive dynamic control over the look of an individual structure in an efficient manner. Custom attributes are procedurally generated to give variation to modules based on

Figure 4.8 - Optimized low-poly render of road system
floor, building type and location. These can be re-seeded or painted on top of after the fact; this feature grants a “by hand” feel to a procedural workflow. Custom modules can be added through a set of user specified rules to control the frequency of custom modules.

The default frequency for all modules is separated by building, then again by floor to add self-similarity to city blocks.

Genesis comes with user driven rules that define how it generates city props. For instance, the width and length of the road defines what kind of intersection is produced. A smaller road will have stop signs placed at intersections with another small road where as longer or wider roads will get traffic lights. This comes complete with crosswalk and stop-line textures placed on the road. The threshold for these parameters can be adjusted or if you wish to change only one, override controls are given
to fix that one intersection that needs to be adjusted. This concept of Procedural Customization can be found in many key aspects of the city generation.

### 4.4 Experimentation and Practical Implementation

The main motivation for this topic was largely derived from a CMIVFX tutorial on procedural city generation based on Geographic Information Systems (GIS) data from opencitymaps.org. While this tool is fantastic for large mainstream cites, such as New York, its database is quite limited, even in large cities such as Detroit. I found myself drawing in most of the building outlines. The buildings also lacked a fundamental level of control and created a generic facade.

Even after moving from GIS data, it took much trial and error to achieve an environment that reached the level of customization necessary. The first build was a more traditional procedural building that was copied to the location of each building. Aligning the shape of these buildings to the shape of the desired location was near impossible as the shapes where being driven by L-systems. This had some successful results using cell noise to define the look of the city but this was too unrealistic and did not achieve the desired level of customization.

Early results of this build were used for a small freelance project with a team of three, one to handle the more detailed modeling, another to paint textures and myself as lead designer. We created a simple boating game that directs players to drive and dock a shipping frigate down the Detroit River. Genesis was used for all of the environment art and to model all the visible geometry of Detroit and Windsor from the river. It also handled the placement of all game art with little modification, including power-ups and check-points.
With tight deadlines and a low budget, I had serious reservations of accepting the job. The timeline was only six weeks from start to finish. I knew a small team would find it difficult to create all art, design the game, hit our deadlines and still turn a profit. The budget was not sufficient to hire enough artists to ensure hitting our goals. Also, we were working for Kinetic Studios, who in turn were contracted by the Detroit Historical Society (D.H.S.), whose own budget for the project was coming from the family of a ship captain lost at sea. Knowing all these participants - Kinetic, D.H.S. and the sponsoring family - would want creative approval in the final delivery, I knew we would be getting differences of opinion and artistic input from several sources. Knowing all this, a procedural system was necessary to keep up with time constraints while turning a profit.

What I did not know, starting out, was the large number of constraints needed to run the game inside of the custom arcade box Kinetic was building. Overcoming these constraints, on our short production schedule, made the last week quite hectic. If we had not been using a procedural system, these factors...
would have sunk the project or pushed back the deadline. Since the game was an important highlight for the Detroit Historical Society's new exhibit in the Dotson Great Lakes History Museum, pushing back the deadline was out of the question.

5.0 City Engine

A commercial procedural modeling tool, Esri's City Engine, is one of the more popular systems. City Engine is a standalone 3D application that uses highly accurate Geographic Information Systems (G.I.S.) to create cities. City Engine uses a type of L-System to generate roads, buildings and facades with a high level of detail. City Engine is efficient and can quickly generate large networks within seconds. Similar to Genesis, it starts with map data, though a bit more sophisticated than a simple screen shot. Its application in games is limited, yet the full version has potential, as it is compatible with Python and can export models in several industry standard formats. The advanced version comes with a steep price of $4,000.00 USD. City Engine has several applications beyond film and games, such as urban planning.

As impressive as City Engine is in terms of quality, it lacks in diversity for game designers. It could be used in a game pipeline with several additional steps, including the need for another 3D program, such as Maya, to achieve a seamless result between City Engine and game engine.

6.0 Conclusion

The procedural tool, Genesis, gives game designers procedural solutions for creating urban environments. Genesis automates the generation of entire cities and allows artists to quickly change the look and shape of a building or city structure with its procedural modules. In
addition, custom modules can be added to these models to allow artists to achieve any look they desire.

The cost and time saving benefits of using a procedural city generator for game environments can be seen in commercial AAA titles such as Boarderlands and in smaller productions such as the Dock a Freighter game. Integrating procedural techniques has the potential to create games with smaller budgets that focus more on. It is my belief that using procedural tools, such as Genesis, will allow smaller studios the opportunity to create games on smaller budgets and larger studios the ability to create games that would not have been possible.

6.1 Future Development

There are several key areas in Genesis that need future development to be considered production ready. As previously mentioned, parking-lots, sidewalks and parks all need to be refined to be anything more than placeholders to assist in fleshing out these specific visuals. The road system is also in need of an overhaul. Currently, there are two systems that detect intersections; one automatically detects intersections but is slow and the end results have messy topology. The results shown use a system that requires the user to paint what type of intersection, four or three way stops, is expected at each point. This system quickly generates cleaner geometry but suffers from workflow issues that force the user to re-paint some intersections if the base curve changes, defeating the purpose of using a procedural tool. A combination of these systems or fixing these caveats is necessary.

The city generator could be optimized to run more efficiently in a game engine. Several modules could be instanced inside of Unity, vastly increasing the speed in which the geometry
is rendered and reacts to dynamic game features, such as light and shadows. The ability to add specific architectural features, such as bay windows, could be more efficiently rendered as well. The only solution for a protruding façade is either to scale that floor or add custom modules. Cresting roofs, chimneys, dormers and eaves would all add to the level of detail that Genesis can achieve.

Another module level could be added to separate buildings into a classification of commercial, residential and skyscraper, which would add a level of control not available in the current build of Genesis. This would allow the rules of each building type to define the rules for each module associated with that type, giving even more realism to the system.

Interiors are another place for improvement. Genesis has laid down the base layer for interiors; from there, internal walls could be added by using the same concepts by which the building shapes are generated. Props for the interior could populate the building using modules for stairs, tables and other furniture.

Genesis has an efficient feedback loop between Houdini and Unity that could be even faster if Genesis was fully integrated with the Houdini Engine. This is the next major stepping stone for Genesis to be a viable tool for production in a gaming pipeline.
Appendix A:

Tutorial

To use Genesis, first create a 2D map, Fig. 4.3, of all the building’s shapes and locations. The height of the buildings can be controlled without a height map, but it is randomly generated and changes with the seed control. To add texture, ensure the appropriate number of shaders have been created in SHOPs and follow the recommended naming convention and folder structure shown in Appendix F.

Import this image into an image network, "COPS". Link this path to the "Building Map" operator path in the main interface. All of the buildings are generated off of this map, and you are now ready to start refining the look of each building.

To generate roads, use a 2D road map reference to draw curves to match reference or simply fill the negative space left by the building maps. Merge these curves together and wire them into the first "INPUT" of the Genesis Road Creation tool. Paint on the curves to create three and four way intersections. This information is used to populate the city with traffic signals, stop signs, parking meters, etc. Export each as .fbx for import into the game engine of choice.

Appendix B:

Genesis Building Generator Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Inputs 1,2,3,4</td>
<td>Wire custom modules into sub-network inputs.</td>
</tr>
<tr>
<td>Refresh</td>
<td>Re-calculates all expressions and seeds.</td>
</tr>
<tr>
<td>Building Map</td>
<td>Location of main 2D building map. All data is generated from this map.</td>
</tr>
<tr>
<td>Random Floor Scale</td>
<td>Controls default min / max values added to each floor.</td>
</tr>
<tr>
<td>Proxy mode</td>
<td>Replaces each building with several planes;</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>View Building Number</td>
<td>Displays only the number Genesis assigns to each building for working on a single building.</td>
</tr>
<tr>
<td>Single Building View</td>
<td>Generates a single building based on its number; this allows efficient customization of each building.</td>
</tr>
<tr>
<td>Building Number</td>
<td>Enter the number of the building associated with Single Building Mode. Disabled by default.</td>
</tr>
<tr>
<td>Paint Modules / Textures</td>
<td>Enables one to paint custom modules or textures. Disabled by default.</td>
</tr>
<tr>
<td>Seed</td>
<td>Sets random seed for all textures, scale, height and decorations.</td>
</tr>
<tr>
<td>Number of Buildings</td>
<td>Lets Genesis know how many buildings to expect and controls the multi-parameter tabs for customization of each building. The default is linked to the number of buildings it will generate, it can be overridden for optimization thought it is not recommended.</td>
</tr>
<tr>
<td>Multi-Parameter (numbered tabs)</td>
<td>Each number is linked to each buildings number. This allows the user to tab through each building and customize each controllable parameter. For example, tab 1 contains all the controls that are specific to building 1.</td>
</tr>
<tr>
<td>Building Height</td>
<td>Height scalar value.</td>
</tr>
<tr>
<td>Fire Escape</td>
<td>Toggles if that building has a fire-escape</td>
</tr>
<tr>
<td>Shader Override</td>
<td>Overrides base module shader</td>
</tr>
<tr>
<td>Awning Override</td>
<td>Overrides awning shaders for that module.</td>
</tr>
<tr>
<td>Decal Shader</td>
<td>Overrides window sills, parapets and support column shaders.</td>
</tr>
<tr>
<td>First / Middle / Top floor tabs (multi-parm)</td>
<td>Container for controls specific to a buildings floor. Used to customize a buildings look based on its floor.</td>
</tr>
<tr>
<td>Floor Scale</td>
<td>Scale of first, middle or top floor</td>
</tr>
<tr>
<td>Square Pillars</td>
<td>Toggle for square support pillars</td>
</tr>
<tr>
<td>Cylinder Pillars</td>
<td>Toggle for cylinder support pillars.</td>
</tr>
<tr>
<td>Ledge</td>
<td>Toggle for horizontal ledge for each floor.</td>
</tr>
<tr>
<td>Floor Ledge Scale</td>
<td>Controls scale for horizontal ledge.</td>
</tr>
<tr>
<td>Floor Pillar Scale</td>
<td>Controls scale for support pillars.</td>
</tr>
<tr>
<td>Random Modules</td>
<td>Toggles use of costume / procedural modules.</td>
</tr>
<tr>
<td>Floor Shader Override</td>
<td>Shader override specific to that building and floor.</td>
</tr>
</tbody>
</table>
## Appendix C:

### Genesis Road Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Inputs 1</td>
<td>Wire curves into sub-network input.</td>
</tr>
<tr>
<td>Road Scale</td>
<td>Universal scale of the road</td>
</tr>
<tr>
<td>Refine Curve</td>
<td>Optimizes road topology for game engines by limiting the roads subdivision around intersections. This can leave artifacts around complex intersections; leave on if none are found.</td>
</tr>
<tr>
<td>Road Samples</td>
<td>Controls the number of polygons for the road model. Creates dense road topology and is used to eliminate artifacts around intersections. Disabled by default and not recommended if being used for games.</td>
</tr>
<tr>
<td>Multi-lane Width</td>
<td>Control for the width a road needs to be before switching out textures with a 4,6 or 8 lane road.</td>
</tr>
</tbody>
</table>

## Appendix D:

### Genesis Road - Intersection Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Inputs 1</td>
<td>Wire custom traffic light modules into sub-network inputs.</td>
</tr>
<tr>
<td>Network Inputs 2</td>
<td>Wire custom stop sign modules into sub-network inputs.</td>
</tr>
<tr>
<td>Network Inputs 3</td>
<td>Wire custom yield sign modules into sub-network inputs.</td>
</tr>
<tr>
<td>Intersection Distance</td>
<td>Distance all traffic lights are placed from the center of each intersection.</td>
</tr>
<tr>
<td>Traffic Light Threshold</td>
<td>Controls the threshold that distributes traffic lights, stop and yield signs. The default placement references the length and width of each road; this controls the size threshold on that length and width. For example, the smaller length and width have yield signs placed, longer roads have traffic lights.</td>
</tr>
<tr>
<td>Stop sign Threshold</td>
<td>Same as Traffic Light Threshold but controls stop signs.</td>
</tr>
<tr>
<td>Yield Sign Threshold</td>
<td>Same as Traffic Light Threshold but controls yield signs.</td>
</tr>
</tbody>
</table>
Appendix E:

Sidewalk Parameters – Work in Progress

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Inputs 1</td>
<td>Wire custom parking meter modules into sub-network inputs.</td>
</tr>
<tr>
<td>Network Inputs 2</td>
<td>Wire custom tree modules into sub-network inputs.</td>
</tr>
<tr>
<td>Network Inputs 3</td>
<td>Wire custom miscellaneous modules into sub-network inputs.</td>
</tr>
<tr>
<td>Speed Limit Frequency</td>
<td>Controls how many speed limit signs are distributed.</td>
</tr>
<tr>
<td>Parking Meter Frequency</td>
<td>Controls how many parking meters are distributed.</td>
</tr>
<tr>
<td>Miscellaneous Frequency</td>
<td>Controls how many miscellaneous objects are distributed. This includes, trees, bus stops and park benches in the current build.</td>
</tr>
<tr>
<td>Map Threshold</td>
<td>Controls the threshold linked to the trace SOP used to generate sidewalks shape and decoration placement.</td>
</tr>
<tr>
<td>Sidewalk Map</td>
<td>2D map input to generate sidewalks.</td>
</tr>
</tbody>
</table>

Appendix F:

Naming and Folder Structure Conventions:

Shader names should be located inside of the geometry SOP Genesis has been loaded into. Or up one level from the Genesis, shaders should be named in the following manner with a padding of one: Wall_01, Concrete_01, Metal_Base_01, Sidewalk_01, Crosswalk_01, Road_01, MultiLane_Road_01, OneWay_Road_01, Awning_01.

Folder structure should follow this convention to maintain relative paths and ease of import into unity. A single base folder contains the following:

<table>
<thead>
<tr>
<th>Folder name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom Modules</td>
<td>Contains all costume modules.</td>
</tr>
<tr>
<td>Exports</td>
<td>.fbx exports for unity</td>
</tr>
<tr>
<td>Hip</td>
<td>Contain all Houdini work files</td>
</tr>
<tr>
<td>Maps</td>
<td>Contains all final 2D maps for Houdini import.</td>
</tr>
<tr>
<td>Modules</td>
<td>Contains all of Genesis’s default modules</td>
</tr>
<tr>
<td>Textures</td>
<td>Contains all default textures used for Genesis.</td>
</tr>
</tbody>
</table>
Appendix G

Survey of Procedural City Generator: Genesis

Administered by: Matt Fournier

Name: Derek Miller
Date: 08/27/2014
Qualifications: B.S. Computer Animation – Full Sail University

Rate the following 1-10, 10 being the best.

Usefulness - 8  Quality - 5  Flexibility - 9
Efficiency - 8  Ease of use - 5  Practicality - 8

Briefly describe the benefits or your favorite features.

Very quickly has the ability to create and customize buildings from hand drawn maps.

Identify any areas that require future development.

The user interface is a little clunky, this could be due to my limited knowledge of Houdini. The painting tools are slow, but it is very unique and effective.

Overall experience or other comments.

Overall, I feel this tool has a lot of potential but like anything needs some optimization, and some documentation. It would be really cool, if the materials could be procedurally generated and not texture maps. I’m really looking forward to use the final product.
Appendix H

Survey of Procedural City Generator: Genesis

Administered by Matt Fournier

Name: Patrick King

Date: 8/2/14

Qualifications: B.S. Computer Animation

Rate the following 1-10, 10 being the best.

Usefulness -9  Quality - 6  Flexibility - 9
Efficiency - 8  Ease of use - 5  Practicality - 8

Briefly describe the benefits or your favorite features.

Allow you to quickly layout a city scene but tweak positioning as needed and the project
develops.

Identify any areas that require future development.

I think the ability to use the tool in other 3d packages other then Houdini would benefit its Ease
of use as people can use the programs they are most comfortable.

Overall experience or other comments.

I enjoyed using the tool and I’m looking forward to being able to use it for a couple projects I
have coming up in the near future. Genesis will make it much easier to do city’s in the future
and I’m eager to get my hands on future versions.
Bibliography


