A Survey of Procedural Techniques for City Generation

George Kelly
kellygp@gmail.com
ITB, Dublin, Ireland

Hugh McCabe
hugh.mccabe@itb.ie
ITB, Dublin, Ireland

Abstract

The computer game industry requires a skilled workforce and this combined with the complexity of modern games, means that production costs are extremely high. One of the most time consuming aspects is the creation of game geometry, the virtual world which the players inhabit. Procedural techniques have been used within computer graphics to create natural textures, simulate special effects and generate complex natural models including trees and waterfalls. It is these procedural techniques that we intend to harness to generate geometry and textures suitable for a game situated in an urban environment. Procedural techniques can provide many benefits for computer graphics applications when the correct algorithm is used. An overview of several commonly used procedural techniques including fractals, L-systems, Perlin noise, tiling systems and cellular basis is provided. The function of each technique and the resulting output they create are discussed to better understand their characteristics, benefits and relevance to the city generation problem. City generation is the creation of an urban area which necessitates the creation of buildings, situated along streets and arranged in appropriate patterns. Some research has already taken place into recreating road network patterns and generating buildings that can vary in function and architectural style. We will study the main body of existing research into procedural city generation and provide an overview of their implementations and a critique of their functionality and results. Finally we present areas in which further research into the generation of cities is required and outline our research goals for city generation.

1 Introduction

As technology evolves and computing power increases, the consumer appetite for more detail, realism and scale is ever growing. The modern media industry, including games, films, advertising and television, is struggling to meet the expectations set by the largest projects and everyday production costs are spiralling out of control.

The traditional approach to meet consumer demand has been to simply increase the number of artists working on a project to produce larger, more detailed and realistic content. However, increasingly the artistic pipeline is not scaling, meaning that additional artist numbers do not generate a proportional yield of content. The additional costs incurred add to the already high development costs and are paid by the consumer. The result of this is that time and money that could have been allocated to improving game play or adding innovative features has been lost on content creation. As a consequence of high development costs, a barrier of entry into the market is created and new fledgling companies find it difficult to get a foothold thus stifling innovation.

A potential solution for the content creation problem is the application of procedural techniques. These techniques have been used for over 20 years in the field of computer graphics [17] for a wide range of applications: adding noise to existing textures [28], creating 3D textures of natural materials such as marble and wood [24], visualising life-like models of various tree and plant species [27] and generating detailed cellular textures such as skin or bark[17]. Entire procedural worlds are now possible and this is demonstrated in the MojoWorld [6] application, where assets including realistic natural features such as terrain, lakes, trees and shrubs are all generated using procedural techniques. Recently procedural applications have been expanded further to simulate special effects including particle systems, water, and even the natural physical movements of assets [5]. Complex scenes containing many different models would normally take months to manually construct, now vast section of these scenes can be created using specialist procedural generation packages [7] that can generate detailed and varied models in minutes. Procedural generation is a time saving method of rapidly and efficiently generating content that can help to alleviate and potentially solve the problems of escalating content creation costs.

Existing procedural solutions primarily apply procedural techniques to the generation of natural phenomena, but many of the same techniques have obvious applications in the generation of man-made artificial phenomena. Our work focuses on the creation of procedurally generated cities for use in games and other graphical applications that are situated in urban landscapes.

Cityscapes are difficult to model. They are rich in visual and functional complexity and are a result of development and evolution over hundreds of years under the influence of countless factors. Some of the major influential factors affecting cities include population, transport, environment, elevation, vegetation, geology and cultural influence. It is a formidable challenge for researchers and developers to create a realistic model of such a large and complex system. We aim to develop an accessible interactive software system that can automatically generate a realistic, detailed and varied model of a city suitable for use in real-time rendering.

In this paper we present a survey of procedural generation techniques and of attempts to apply these techniques to the city generation problem. In Section 2 we provide an overview of procedural generation in general and present a number of key techniques and algorithms. In Section 3 we describe how researchers have attempted to apply procedural techniques to city generation. Section 4 concludes with an outline of our proposed approach for creating a city generation system.

2 Procedural Techniques

The key property of procedural generation is that it describes the entity, be it geometry, texture or effect, in terms of a sequence of generation instructions rather than as a static block of data. The instructions can then be called on when required to create instances of the asset and the description can be parametrised to allow the generation of instances with varying characteristics. A typical example of this approach would be the population of a forest with procedurally generated unique trees [5].

Procedural techniques can thus be employed to produce varied assets. One of the most basic techniques that can be used is the generation of 3d primitives with random parameters, for example

a cuboid with random height. Simple algorithms utilizing pseudo random functions can be employed to generate noise for use in texturing and natural formations[28] More complex recursive algorithms such as fractals or L-systems can be used to recreate organic structures found in nature like snow flakes and trees [27]. Ebert et al. [17] identify the following as important features of procedural techniques:

- **Abstraction:** Geometric and texture data is not specified in the conventional sense, instead details are abstracted into an algorithm or set of procedures. These procedures are then handled by the computer and called on when needed. Minimal details are required and the operator can manipulate the model data easily without requiring intimate knowledge of the implementation.
- **Parametric Control:** Parameters are defined and adjusted that directly correspond to a specific behaviour in the procedural generation. The developer can define as many useful controls as required for the artists to operate effectively. Example of parameters include the height of the mountains in a terrain algorithm or the number of segments in a procedural sphere.
- **Flexibility:** It is possible to capture the essence of an entity without explicitly bounding it within real-world limits. Parameters can then be varied to produce a wide range of results which are not necessarily limited to the constraints of the original model.

Procedural techniques have been applied successfully in the generation of numerous complex phenomena in computer graphics and have proved beneficial for a number of reasons.

Textures, geometry or effects abstracted into procedural algorithms are not fixed at a set resolution or number of polygons. Procedural techniques are therefore inherently multi-resolution in nature and can vary the complexity of their output. This capability is of particular interest to computer graphics. For example *level of detail* (LOD) is important in any 3D rendering system and essential to real-time rendering applications [22]. The concept behind LOD is to use more simple versions of an entity if it contributes less to the final rendered image. So for an object that occupies only 4 pixels in the final image, 10,000 polygons are not required and a basic representation using 10 polygons would be sufficient. The multi-resolution nature of procedural techniques allows the possibility of automatically generating models at multiple levels of detail [17].

Concise descriptions for generated objects are possible and can often be expressed in the terms of a few simple parameters. These small descriptions can be used to create large amounts of detailed textures and geometry, this effect is known as data amplification [17] and provides developers with the means to create an entire game world that is easily distributable over low-bandwidth network connections. The conciseness of procedural techniques are exploited by *Demo Scene* creators who create and distribute scenes that are complex and rich in detail in the form of tiny executable files as small as 2KB [10].

The flexibility and control provided by procedural techniques give the designer a platform for artistic freedom and experimentation. New visual effects and original objects can be created by experimenting with parameter values that exceed normal boundaries. [7]

Typically procedural algorithms are implemented in software, however recent advances in graphics hardware have opened up the possibility of executing them directly on the GPU. For example, complex procedural techniques like volumetric textures that were previously impossible to run real-time can now be implemented in this manner [21][16]

We now describe a number of fundamental procedural techniques and algorithms that have been successfully employed within the domain of computer graphics.

2.1 Fractals

Natural shapes are not easily described by conventional geometric methods. Clouds are not spheres and mountains are not cones. Natural shapes tend to be irregular and fragmented and exhibit a complexity incomparable to regular geometry [29]. However these shapes can be described using a branch of mathematics called *fractal* mathematics. Benoît Mandelbrot, regarded as the 'father of fractals', coined the term fractal in 1975 [29] from the Latin *fractus* meaning broken.

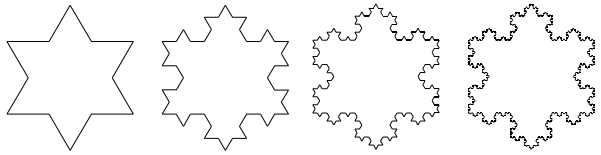


Figure 1: The first four iterations of the Koch snowflake

The basic concept of fractals is that they contain a large degree of *self similarity*. This means that they usually contain little copies of themselves buried deep within the original like the stars embedded in the Koch Snowflake[17] shown in Figure 1. Also, fractals possess infinite detail, so for any given fractal the closer we look at it the more detail it can reveal. [20]

Like any procedural technique, a fractal shape is defined by an algorithm for generating the shape. In the case of fractals these algorithms are recursive and successive recursions yield more detailed versions of the basic shape. The example of the Koch snowflake in Figure 2 shows four such recursions. Self-similarity is achieved by generating the same shapes or patterns at smaller and smaller scales as the recursion progresses, a property often referred to as scale invariance. There is no theoretical limit to the amount of recursion that can be done and hence infinite levels of detail exist within the shape.

Visualizing fractals manually is repetitive, tedious and limited and therefore computer-based implementations of fractal algorithms have been present from the start. Mandelbrot utilized computers while an employee at IBM to visualise complex fractals including the Mandelbrot Set [29] (see Figure 2).

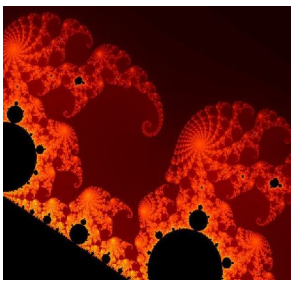


Figure 2: Mandelbrot Set. © Wiki public domain.

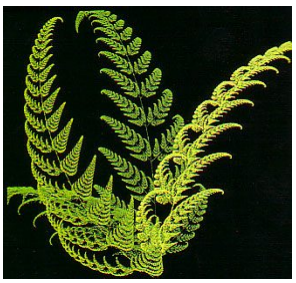


Figure 3: IFS Fractal Ferns [25]

Fractal-like shapes such as trees or ferns can be procedurally generated using relatively simple recursive algorithms. In fact a wide range of natural structures from simple plants to terrain can be generated in this manner [25]. Fractal algorithms provide effective abstraction from the structural complexity of the natural

objects they represent and can utilize recursion to provide varying levels of detail. These techniques also provide the key property of data amplification in that complex models can be generated from the recursive application of simple equations.

Fractals are limited however to self similar structures and the objects we are seeking to model may not necessarily contain this self-similarity. They are superseded in many contexts by other more flexible algorithms like formal grammars such as L-systems.

2.2 L-Systems

Lindenmayer systems, or L-systems for short, are a formal grammar devised by biologist A. Lindenmayer as a mathematical theory for biological development. L-systems were originally developed to study bacteria replication and the growth patterns of simple organisms such as Algae described by Lindenmayer in the *Journal of Theoretical Biology* in 1968 [32]. The system and its applications have evolved and are now applied in the field of computer graphics and in particular to the generation of fractals and the realistic modelling of plants.

The central concept of L-systems is that of rewriting [26]. In general, rewriting is a technique for defining complex objects by successively replacing parts of a simple initial object using a set of rewriting rules or productions. The components of an L-system are as follows:

- V (the alphabet) is a set of symbols containing elements that can be replaced (variables)
- S is a set of symbols containing elements that remain fixed (constants).
- ω (start, axiom or initiator) is a string of symbols and constants that define the initial state of the system.
- P is a set of rules or productions defining the way variables can be replaced with combinations of constants and other variables. A production consists of two strings - the predecessor and the successor.

An initial state or axiom, ω, is provided which is then rewritten using a series of rewriting rules or productions, P. The productions are applied iteratively, allowing large complex objects to be defined using a simple set of productions.

$\boldsymbol{V} = \{a, b\}$	
$\boldsymbol{\omega} = a$	$n=1:\ ab$
$\boldsymbol{P}_1 : a \rightarrow ab$	$n=2:\ abba$
$\boldsymbol{P}_2 : b \rightarrow ba$	$n=3:\ abbabaab$

Figure 4: *The Thue-Morse system*

L-systems can be used to visualise structures by embedding graphical symbols within the string that can be used later to render it. Turtle commands can be used to describe and visualize a wide range of L-systems including Koch's snowflake, plants and branching structures. The concept behind Turtle Graphics is that the 'turtle' is given instructions relative to its current position and as it moves it leaves a pen line mark behind it. Using turtle graphics: shapes, drawing and structures can be defined in the terms of a L-system. Using a bracket extension to Turtle Graphics, L-systems can support the branching structures such as trees that are predominant in nature. [32] Figure 5 illustrates the application of such an L-system used here to recreate a complex tree.

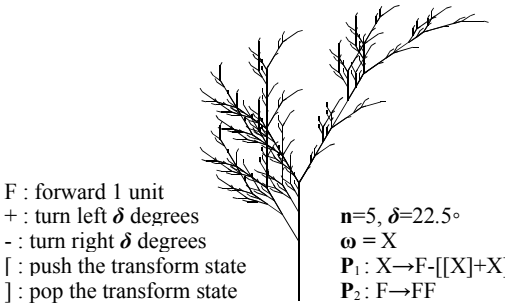


Figure 5: Tree formation generated with via the turtle graphics L-system interpreter. [27]

L-systems were designed to define and visualize sophisticated plants and other natural structures. As academic research has continued into their application in botany it has also continued in the realm of procedural generation. Significant advances have been made and packages are now available commercially that apply L-systems to generate rich landscapes of detailed flora covering a wide range of different species.

Figure 6: Speed Tree [5] screenshot demonstrating procedurally generated and real-time rendered trees.

L-systems are a good example of procedural techniques for a number of reasons. They allow complex models and organic structures to be defined, modelled and visualised using a concise set of productions. A varying level of complexity can be supported by parameters such as the recursion level of the L-system [15]. The algorithms can be defined in a compact and intuitive manner and can effectively abstract the recursive structure of many natural phenomena. L-system generation can be adjusted easily via external parameters and are extensible by nature similar to other formal grammars.

2.3 Perlin Noise

Perlin Noise was initially developed to help create more “natural looking” textures. The technique was developed by Ken Perlin for use in the film Tron in 1982. As a result from his work in Tron Dr. Perlin received an Academy Award for Technical Achievement in 1997 [24]. Noise is created by first using a pseudo random function to generate a series of values which are then interpolated into coherent noise. Several layers of this coherent noise are then composited together using different ratios to create a “natural looking” texture with fractal like detail.

Noise Function

A noise function generates random data. So each time the function is called a new number is returned. This is useful however does not allow control of the results obtained. In order obtain **parametric control** from the noise generator a seeded random function is used.

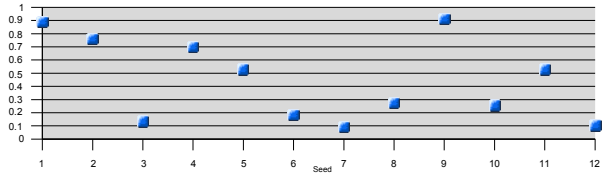


Figure 7: A seeded random generator will produce the same results when given the same input number or seed but still produces numbers in a random pattern.

Interpolation Function

Interpolation is a process of curve fitting in which a function is constructed that intersects exactly through the data points. This function can generate new data points given known data points, in this case the input points are those generated by the noise function. For a finite set of data points a function can be generated that allows us to obtain an infinite range of points. Several different algorithms are available to perform this interpolation. The algorithms vary in the number of data points they take as input, the accuracy they provide, their computational complexity and the smoothness of the curve that they generate. The graphs below demonstrate just two of the many different methods used for interpolation using data from Figure 7. Figure 8 shows the most basic linear interpolation and Figure 9 the more complex cubic interpolation.

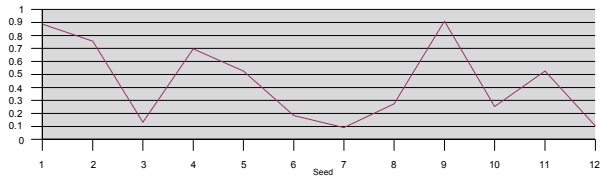


Figure 8: **Linear interpolation** is one of the simplest methods used and is often selected when speed is important and quality is of secondary importance. Linear interpolation only takes two points and the interpolant is calculated using the weighted mean.

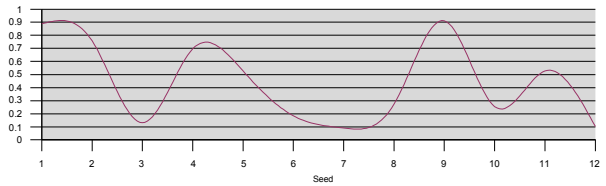


Figure 9: **Cubic interpolation** is quite computationally intensive. It is used when quality not speed is of particular importance and it outputs a continuous curve unlike linear interpolation. Cubic interpolation requires four points.

Turbulence

Results produced from interpolated noise have random properties but appear quite artificial rather than natural. In nature, there are many different scales of detail present. For example take a mountain range: large details are present with giant peaks and troughs, medium scale details are present through the smaller hills and crests, small details present through boulders and rocks, etc.

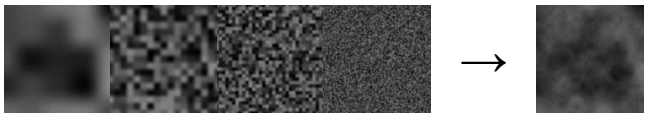


Figure 10: Combination of several layers of noise.

To provide a more useful texture source that resembles nature more *turbulence* is applied by combining several noise textures of differing scales. Each layer of noise is referred to as an *Octave* and layers are combined with different amplitudes and frequencies. The variation of amplitude and frequency can be expressed a *Persistence* value. *Persistence* can help describe the effect successive octaves have on the previous iterations by defining the amplitude between octaves as a fraction. Perlin Noise generated with a low ratio of persistence is typical smooth with very fine detail, Perlin Noise generated with a high persistence is more jagged with less fine detail.

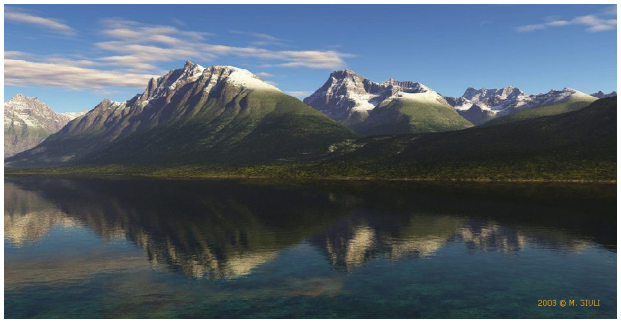


Figure 11: Photo realistic scenery and rendered using Terragen with procedural geometry generation and procedural texturing. © 2003 M. GIULI Terragen Artist.

Terragen[9] uses the Perlin Noise algorithm to generate photo realistic terrain, clouds and seas. Figure 11 Showcases the details and scale of output that can be achieved using the Terragen procedural generation software. Parametric control is vital to the procedural generation process as it allows the generation to be easily managed and enables vastly detailed scenes to be defined in the terms of a few parameters. The Perlin noise algorithm also provides a mechanism by which the height for any point or region can be calculated on the terrain without the need to store the massive terrain geometry data.

Figure 12: Marble vase textured with a Perlin procedural volumetric texture[24]

In addition to 2D textures, Perlin Noise can be used to generate 3D textures, often referred to as volumetric or solid textures. Volumetric texture differ from conventional 2D textures in that they do not require bindings to geometry coordinates but allow objects to be virtually carved from the texture as if they were a solid block [24]. The example shown in Figure 12 shows a vase carved out of a volumetric marble texture created using Perlin noise. The texture manages to replicate the veins running through the marble and achieves a higher level of realism than is possible using 2D texturing techniques.

Volumetric textures are computationally expensive to render, but the real barrier for their widespread use is their memory and

storage requirements. Compression such as S3TC can partly alleviate the memory problems of 3D textures but do not go far enough. Perlin Noise requires minute storage due to its procedural nature thus removing any storage burden, and can even by used to render volumetric textures in real-time using the pixel-shader hardware on recent GPU's [21].

As a procedural generation technique Perlin Noise provides a comprehensive set of benefits. Parametric control provides the developer with flexibility to control the output through high level parameters. Reproducible geometry and textures created using the algorithm have minimal storage requirements, can be generated efficiently and can be defined in the terms of a few simple parameters. Textures of any size and detail can be produced providing an innate level of detail. The output created is tile-able, allowing seamless joins suitable for techniques like repeating and layering which are common place in multi-texturing. It can also be used as method to enable real-time volumetric textures on modern graphics hardware.[21] Perlin Noise has proven to be one of the most useful procedural techniques and is beneficial in wide range of computer graphics applications.

2.4 Tiling

Tiling is one of the most basic procedural techniques and has traditionally been applied in game development. It is used in many classic games including titles such as Sonic, Mario and R-Type. Originally tiling was used by creating small sections of 2D graphics that could be repeated on screen and assembled together to create the virtual world. Games such as the *Shoot Em Up Construction Kit* [4] released in 1987 by *Sensible Software* allowed the user to construct and edit game maps using a library of tiles and a simple interface.

More recently tiling techniques have evolved and are used in the form of *multi-texturing* to create highly detailed and varied textures from layers of base textures. New materials are created by combining a set of detailed textures, colour maps and blending maps. Using this technique terrain can be procedurally textured by applying several layers of detailed tile-able textures.[9] Examples of texture layers could include rock, grass, sand and snow. These texture layers can be combined with varying degrees of influence on the final texture. Textures are applied to the terrain according a variety of specified parameters, they can be selected according to height, slope or specified explicitly using an image map. This solution allows vast areas to be textured in detail which is not possible using a single high resolution texture.

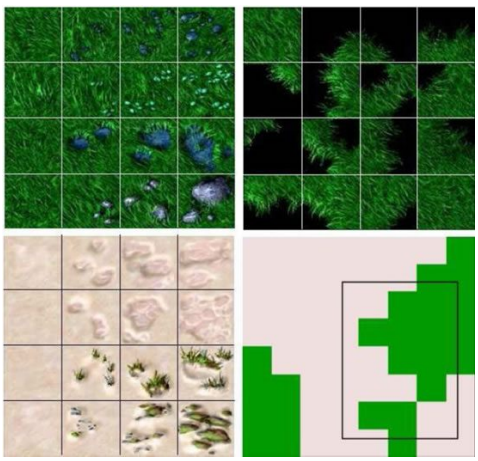


Figure 13: Warcraft® III uses stochastic information to procedurally generate Textures. © 2002 Blizzard Entertainment

Extended algorithms exist that use stochastic information such as probability distribution maps to procedurally texture landscape. An image map for the terrain area is supplied that stores the probability of using various tiles. Constraints can be specified to state which tiles can be joined under what conditions and whether they may be joined directly or require transitional tiles. Using a pseudo random function thousands of different permutations of worlds are possible from a single probability map. Each possible world can be stored and recalled by simply taking note of the seed used to create the world [14].

Tiling systems provide several advantages for graphics applications. Vast and detailed landscape or terrain for virtual worlds can be created from stochastic information and small set of texture tiles. These maps and game worlds can then be easily distributed for on-line gaming which is of particularly useful for massively multi-player on-line role-playing games (MMORG) and other on-line applications where game resources are shared. Storage and memory requirements are minimised so it is possible to optimally store and render worlds of vast dimensions in real-time on commodity hardware. Tiling is a good example of how a simple procedural technique can be applied and extended to provide benefits for graphics applications.

2.5 Voronoi Texture Basis

Voronoi diagrams were demonstrated as a method of procedural generation by S. Worley in his paper titled 'A Cellular Texture Basis Function', in which he detailed an algorithm that partitions space into a random array of cells creating cellular looking textures. The technique was devised to complement existing procedural techniques such as Perlin Noise and provide a method of procedural generation for cellular surfaces such as skin or bark. Voronoi diagrams have long preceded their application in procedural generation and have traditionally been used in a wide range of scientific applications including spacial analysis, planning, urban settlement analysis, geology, robotics and ecology.

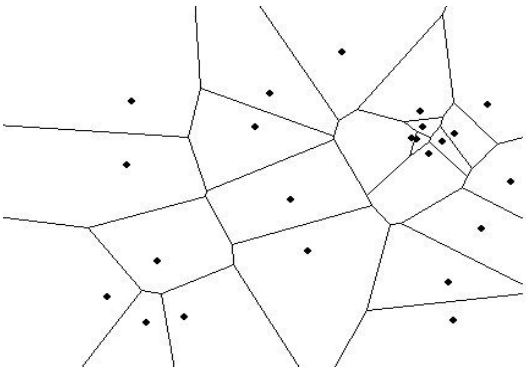


Figure 14: Voronoi Diagram with coloured cells

A Voronoi diagram is the decomposition of some metric space determined by distances to a specified discrete set of objects in the space. Figure 14 shows an area partitioned into cell by lines which are plotted using the points on the map. Each boundary line is positioned equidistant between each pair of neighbouring points. The resulting Voronoi diagram is a result of the position of the original points. A wide range of cellular patterns can be created by using different configurations to place the points used to create the diagram, also this data can be interpreted and rendered in many different ways for different effects.

Figure 15: Photo-realistic surfaces procedurally created using Worely's cellular basis algorithm. [3][17]

The Worley algorithm achieves effective abstraction for the generation of cellular surfaces by providing a small set of parameters termed the 'Worley constants' that can control of the algorithms operation yet allow great variance of the output.[3] Natural surfaces such as paper, skin, cobblestone, tree bark and sun baked mud are prime targets for this algorithm and can be recreated effectively with little input data required. [17] Figure 15 shows examples of Worleys algorithm applied to procedurally generate natural textures. The algorithm has been used successfully in procedural generation creating a variety of richly detailed cellular surfaces which can be concisely defined in the terms of a few simple parameters.

3 Procedural City Generation

The procedural methods outlined in the previous section have largely been applied to the generation of natural objects and textures. Only recently have researchers turned their attention to their application in the context of generation of man-made phenomena such as an urban area. In this section we shall review and evaluate research that has been carried out on procedural generation of cities. City generation is achieved through a series of stages that each use a number of techniques to create roads, lots, building structure and building faces.

Road networks are a key aspect of city character and identity. Road networks are difficult to generalize since they are an interwoven component of a complex system. When viewing road networks from a map or city plan a number of patterns can be observed. It is these patterns that are key for procedural generation as they encode the structure of the road network. There are numerous road network patterns deployed in cities ranging from the tightly structured grid plan network with perpendicular roads in a regular chequerboard structure to the hierarchical network with sprawling secondary and tertiary roads feeding into arterial roads in a branch like system. The patterns applied within a city are a result of numerous factors including location, geography, cultural influences, planning trends, etc. Cities can be categorised by the road patterns they contain: modern US cities like New York are arranged in a chequerboard or raster pattern, some European cities like Paris are structured with a radial or concentric pattern most evident. However most cities contain a number of patterns, with different patterns prevalent in different regions or neighbourhoods within the city. [33][31]

City buildings are difficult objects to procedurally generate because of their individuality. The buildings present in a modern city display a diverse range in both function and style. Buildings as functional units serve a specific purpose or role within each neighbourhood, borough, district and city. The number of roles for buildings are many and combined with the geographic composition within a city make for an extremely complex system. Such a complex system is difficult to model, but a simplified solution can be used, similar to that used in statistical analysis, that uses classes or groups to model building function. Usage groups such as commercial, residential and industrial can be used as select generalizations for the numerous building roles and a simple mechanism for modelling function within cities. The style of a building and in particular its geometry and materials are often the result of numerous architectural and cultural influences. Such a complex form is difficult to model and an approximation or substantially reduced model is needed to limit the complexity of the generation system.

To effectively evaluate the generation systems we have identified a number key criteria:

1. **Realism** – Does the generated city look like a real city?
2. **Scale** – Is the urban landscape at the scale of a city?
3. **Variation** – Can the city generation system recreate the variation of road networks and buildings found in real cities or is the output homogeneous?
4. **Input** – What is the minimal input data required to generate basic output and what input data is required for the best output?
5. **Efficiency** – How long does it take to create the examples shown and on what hardware are they generated? How computational efficient is the algorithm?
6. **Control** – Can the user influence city generation and receive immediate feedback on their actions? Is there a tactile intuitive method of control available or is the control restricted? To what degree can the user influence the generation results?
7. **Real-time** – Can the generated city be viewed in real-time? Are there any rendering optimisation techniques applied to enable real-time exploration?

An overview will be presented of each of the city generation systems and an insight provided into the functioning of the techniques and algorithms applied in the systems. Following each outline of the system we discuss and evaluate each system according to our criteria. Realism, Scale, Variation, Input, Efficiency, Control and Real-time optimisations.

3.1 Grid Layout & Geometric Primitives

Stefan Greuter et al.[19][13][8] outline a solution to procedurally generate a city in real-time. The techniques applied to generate the city are discussed in a number of papers and demonstrated in a virtual city application titled *Undiscovered City*. The application creates a road network using a simple grid layout upon which it can place buildings generated using a combination of simple geometric primitives. The research is specifically targeted for real-time applications and the *Undiscovered City* serves as a proof of concept running in real-time at interactive frame rates.

Road Network: Grid Layout

The roads of the city are created in the pattern of a uniform grid in a similar fashion to the centre of a modern planned American city like New York. The grid is regular and size of each block is constant but can be adjusted globally.

Figure 16: Screen shot at street level in the *Undiscovered City demo*

Figure 17: Screen shot from the *Neverland* demo

The grid based road network generation has been improved in the Neverland demo, a more recent work from Greuter et al., shown in Figure 17. This system extends some of the buildings over more than one grid block creating a more disjointed road network and giving the city a more realistic appearance. A paper detailing the Neverland demo has yet to be published.

Buildings: Geometric Primitives

The building generation system uses the location of buildings in the form of a grid coordinates as a seed for building generation. The appearance of each building is determined by this seed including properties such as height, width and number of floors. Generating buildings using a similar set of numbers such as neighbouring grid coordinates can result in similar looking buildings, so to overcome this a hashing function shown in Figure 18 is implemented in order to provide more random distribution.

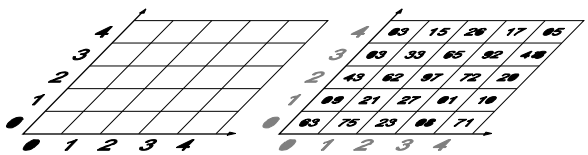


Figure 18: Grid Layout Coordinates & Hashing [13]

Building geometry is generated using the concept of combining geometric primitives to form building sections. Each building section is constructed using a different floor plan. The top most section of buildings are created by extruding a three dimensional volume from the most basic of floor plans, composed from only a few primitive shapes. In subsequent sections below, another primitive shape is added to the previous floor plan and a three dimensional volume is extruded in the same fashion. Figure 19 illustrates how the creation of consecutive sections are combined to form the complete geometric model of a building. Figure 20 shows the generated buildings with their textured faces which are not procedurally generated but are selected from a set of 10 building window textures.

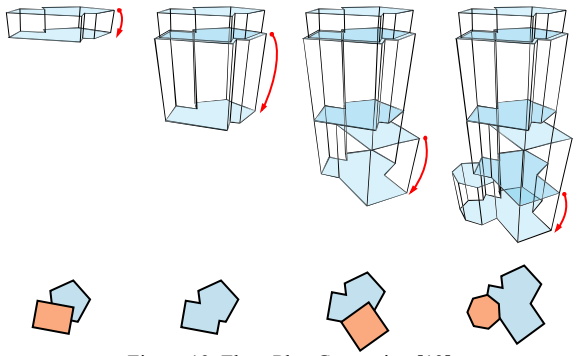


Figure 19: Floor Plan Generation [13]

Figure 20: Screen shot from the *Undiscovered City* demo

The *Undiscovered City* is designed with real-time applications in mind and implements optimisations such as a geometry caching and view frustum culling. The culling technique, referred to as *View Frustum Filling* [13], renders only the buildings visible within the view frustum as shown by Figure 21. By loading and rendering a reduced set of buildings the amount of memory required to store the scene and the graphical processing power required to render the scene are minimised enabling the real-time rendering of a large data set like a city. The regular grid road network allows easy detection of building visibility within the view frustum and hence provides a computationally efficient method to cull superfluous buildings from view.

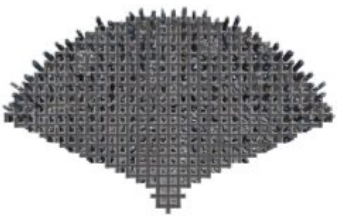


Figure 21: View frustum filling. [13]

In addition to culling building geometry, a building cache is also implemented. Buildings are generated in advance and defined as OpenGL display lists that can be stored in the building cache. The cache employs a LRU(least recently used) algorithm: recently

accessed buildings are kept in the cache while older less recently accessed items are replaced. As a result of using the building cache memory use is optimized and buildings can be recalled from cache for display an order of magnitude faster(up to 8x) than they can be generated from scratch.

Discussion

1. **Realism:** The single grid pattern used does not reflect real cities that are constructed from a number of patterns and the resulting road network appears artificial and homogeneous. Buildings appear angular and modern and are somewhat realistic but unconvincing. Simple windowed faces are used and the buildings are not geometrically detailed.
2. **Scale:** The grid layout system can create road networks on a very large scale and is limited only by the size of the integer based coordinates. At 2^{32} cells wide, the size is not a practical restriction for city generation.
3. **Variation:** The road network provides little variation, a single regular grid pattern is used and only the grid spacing can vary from city to city. The grid system is required for the real-time optimizations and so is largely inflexible. Only a single building type is constructed, an office skyscraper with 10 different window textures, no other type of building is supported. Although the geometry for each building is different the amount of variation insufficient to emulate a real city.
4. **Input:** No input maps or geo-statistical data is required. No external image maps are required and the application is standalone.
5. **Efficiency:** Road network and building generation take place in real-time and figures are provided for the generation and rendering of the *Undiscovered City*.
6. **Control:** Grid spacing can be adjusted using short-cut keys in the application and the changes can be viewed in real-time. The building generation process is not interactive and all buildings are generated using a random seed created using a set of building coordinates from the road grid network.
7. **Real-time:** The system is designed for real-time applications and can render views of large scale cities in real-time on commodity hardware from 2003 at interactive frame rates. [Performance for numbers of buildings being displayed on screen: 200 buildings @60fps, 500 buildings @20fps, 1000 buildings @5fps].

3.2 L-systems

Parish and Müller[23] presented the *CityEngine* in a paper titled *Procedural Modeling of Cities* at SIGGRAPH 2001. The CityEngine consists of a suite of components including road generation, building construction and building face creation that unite to form a pipeline for city generation. L-systems are selected as the key technique for procedural generation in the *CityEngine*. Lindenmayer-systems have traditionally been used to model natural phenomena but are also suitable for the generation of cities due to their concise nature, computational efficiency and data amplification properties.

Road Network: L-systems

L-systems have been used to model natural phenomena and the generation of plants and other branching structures provide some similarities to the generation of road networks. The CityEngine uses an extended form of L-systems titled Self-sensitive L-systems to construct road networks in a manner which takes existing growth into account.

Input is taken in the form of 2D image maps. Geographical information on elevation, vegetation and water boundaries is required and additional socio statistical image maps can also be included specifying information such as population density, land usage, street patterns and maximum building heights. A road network generation application, shown in Figure 23, is used to manage the generation of roads and allow the operating user to specify extra parameters such as the smoothing angle of road network edges, road width, etc. Although only a geographical input map is required the examples included in the paper, such as Virtual Manhattan in Figure 27, utilize a number of different input maps.

Road generation is accomplished through the use of two rule sets: the *Global Goals* and the *Local Constraints*. Road segments are initially plotted according to the Global Goals which are similar to the goals that a city designer may have. These tentative plans are then refined by the Local Constraints which reflect the practical constraints of the real world and the state of the existing road network.

Global Goals

- There are two different types of roads: highways or major roads connect population density centres which can be identified from the population density map supplied at input, small roads connect to the nearest highway.
- Streets follow some super imposed **geometric pattern**.
- Streets follow the path of least **elevation.**

Local Constraints

- Road segments are pruned to fit inside a legal area: line segments extending into water are pruned.
- Roads are rotated to fit inside a legal area: a road to the coast bends around the coastline like a coastal road.
- Highways are allowed to cross an illegal area of a certain distance: a highway approaching a limited span of water will cross over it like a bridge.
- Roads segments are checked to see if they intersect with existing roads or if they come within a certain distance of an existing road junction: Figure 22 shows how proposed road segments are modified to satisfy the self-sensitive rules.

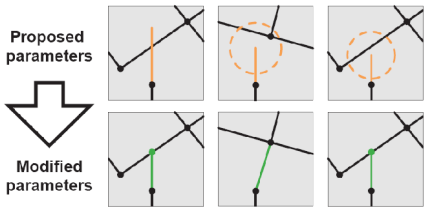


Figure 22: Self-sensitive road L-system [23]

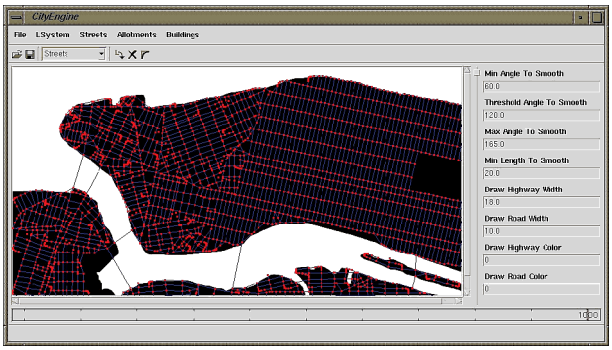


Figure 23: CityEngine GUI displaying Virtual Manhattan after 100 steps. [2][23]

Buildings: L-systems

The CityEngine constructs buildings on the road network in a series of distinct stages: define building allotments, create building geometry and generate textured faces. To define building allotments the CityEngine utilizes data from the previous road network generation stage. Figure 24 outlines the stages of allotment generation. Allotments or lots are calculated by first extracting blocks from the road network using the roads of the network as the dividing borders. Each basic extracted block is then divided into a series of potential lots via randomized subdivision. Lots that are too small or have no immediate street access are culled and removed from the system. The final lots generated by the CityEngine are shown in the right-most image of Figure 24 and appear both varied and practical.

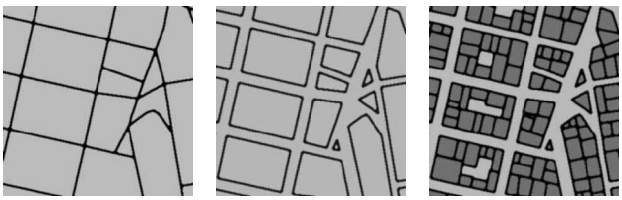


Figure 24: Lot Division Stages [23]

Building geometry is generated through the use of a parametric L-system. Several different building styles are implemented including: skyscrapers, commercial and residential with each type using a different set of L-system productions. The building type is determined from a zone map which can be passed in as an image map input.

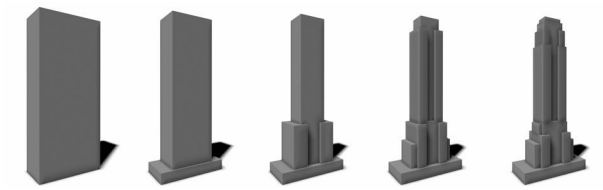


Figure 25: L-System building refinement from bounding box of the Empire State Building [2]

The initial state or axiom of the building L-system is a bounding box generated from the lot footprint and a building height image map if available. L-system operations consist of transformations (scale and move), extrusions, branching and termination, and the use of geometric templates for roofs, antennae, etc. L-systems allow for the addition of more productions and provide an extensible solution. A basic level of detail implementation is possible since each iteration of the building L-system is a refinement of a basic building bounding box as shown above in Figure 25.

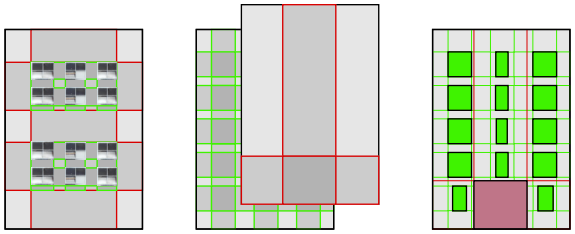


Figure 26: Building face construction [23]

Building faces are created procedurally by generating textures using an over-laid series of grid-like structures. Several layers of grid-like structures are used with functions that define how the layers are combined. The functions dictate which cells from what layer are selected to create the final face and can use conditional and statistical information to select cells. Cells typically contain doors or windows but can contain any building face feature. Shown in Figure 26 is the construction of a face: the red layer influences the selection of cells from the green layer. The resulting face is a conditional combination of multiple layers.

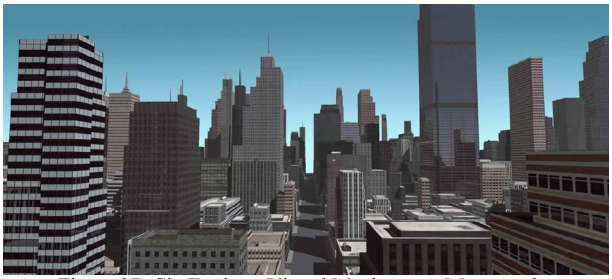


Figure 27: CityEngine - Virtual Manhattan – Maya render

The CityEngine produces data that can be imported into Maya, a commercial 3D package, for final rendering. The sample shown in Figure 27 illustrates such a rendering from Maya, in this case a showcase of Virtual Manhattan.

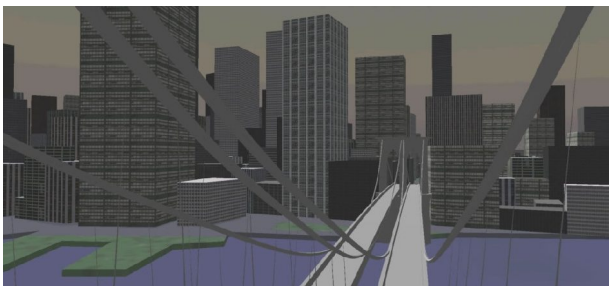


Figure 28: CityEngine - Virtual Manhattan – *DV/reality*

A real-time implementation is available utilizing *DV/reality* software from *Dimension*. DV/reality is a large scale visualisation tool designed to run on super computers and distributed rendering

applications. There are no real-time rendering features such as level of detail or geometry culling discussed and from the screen-shot of *DV/reality* in action in Figure 28 it is clearly evident that a reduced complexity model is being displayed. (Notice how the buildings appear more similar to the left most image of Figure 25 in contrast to the right).

Discussion

1. **Realism:** The CityEngine can create a complex and detailed road network using extended L-systems. Figures 23 demonstrates the generation of a realistic road network, but utilizes real statistical data making the generative capability of the system difficult to assess. The blocks from the road network are divided into realistic and practical lots upon which buildings can be constructed. L-system building generation provides an effective method of generating a realistic cityscape although the resulting buildings are basic. Several different types of buildings including skyscrapers, commercial and residential buildings can be created and green areas are also displayed. Overall a good visual balance is achieved with practical positioning.
2. **Scale:** Scale does not appear to be an limiting factor for the system and is possibly restricted only by the size of the input data maps.
3. **Variation:** a good range of road networks can be created and examples of different generated cities are shown including Paris – Circular, New York – Grid and San Francisco – Terrain wrapping. Buildings vary in shape and scale and a range of building types are catered for, but only limited range of style is demonstrated. In Virtual Manhattan a convincing clone of New York is shown but it may be more difficult to generate other cities where a different architectural style would be required.
4. **Input:** the minimum input required is a geography map however all of the samples shown utilize numerous input maps and include statistical data from a real world cities. A dependence on real-world data would require the acquisition of geo-statistical data to begin using the system which is not desirable. Also, from a practical point of view the system is more difficult to evaluate since it is difficult to determine which patterns are created by the L-systems and which are created as a result of the input data. Although only one input map is required, all of the samples shown in [23] utilize numerous maps to create realistic output like that illustrated in Figure 27
5. **Efficiency:** Road network generation is very efficient, the large road network of the Manhattan sample shown in Figure 27 is created in under 10 seconds. The next stage of generation the building stage takes longer to complete: Virtual Manhattan requires approximately 10 minutes to sub-divide the road network into lots, construct buildings and create textured faces. It is important to note that although the generation time is documented the time required for Maya to render Virtual Manhattan in is not disclosed and would likely take substantially longer than both combined.
6. **Control:** It is unclear how much user interaction is required and no interactive features are specifically documented. L-systems are by their very nature iterative and it appears that the number of iterations used by the system to generate output of an acceptable realism and detail is determined by the user trying different values on a trial and error basis. Other values are also specified manually such as the angle of deviation. Control of building generation appears to be limited to the numerous image maps that can be passed as input.
7. **Real-time:** A real-time demonstration is available using the *DV/Reality* software shown in Figure 28 that displays a simplified version of Virtual Manhattan. *DV/Reality*[DVR] is a visualisation tool designed to provide real-time rendering though the use of high powered graphics workstations and distributed rendering. No documentation on any real-time features in the CityEngine are provided and without features like geometry culling or LOD real-time rendering applications like gaming are not possible. It may be possible to easily add some optimizations such as a simple level of detail implementation based on the principle that each L-system iteration produces a more detailed building version refined from a simple cuboid primitive as illustrated in Figure 25.

3.3 Agent Based Simulation

Watson et al. [12] apply an agent based technique to generate cities in their solution titled *CityBuilder*. The system is built on the NetLogo™ platform which is a multi-agent programmable modelling environment based on the Logo programming language and is designed to provide users with a platform to explore emergent phenomena. The city generation is implemented by simulating cities using a set of agents that can model specific city entities such as developers, planning authorities and road builders. The *CityBuilder* system models not only the road network and buildings but also simulates the growth and development of the city over time.

Road Network: Agent Based Simulation

Roads are created from road segments that are assembled according to a grid pattern. Deviation from the pattern is allowed and can be specified via a parameter. A deviation value of zero will result in a strictly uniform grid like road network, a deviation value near one would result in an organic like network. The interconnectivity of the network can also be altered via constants that dictate the road density and the distance between road intersections.

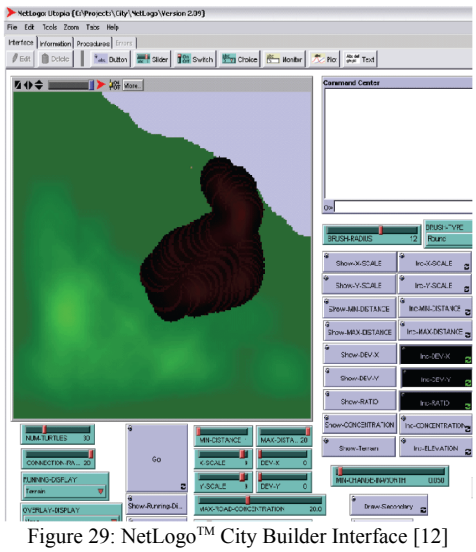


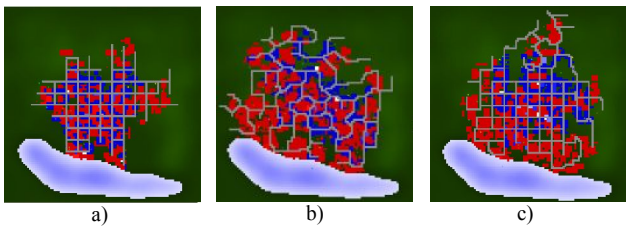
Figure 29: NetLogo™ City Builder Interface [12]

Input in the form of a terrain height map is required along with a specified water level to determine the legal area in which roads and buildings can be placed. Extra parameters such as road density, grid spacing and deviation from grid can be adjusted

using sliders in the interface shown in Figure 29 to alter the behaviour of the agents. Additionally users can specify certain parameter values for specific areas by painting on the map using a brush similar to that in a simple paint application.

The road segments are created by two types of agents – *extenders* and *connectors:*

- *Extenders* roam around terrain near to existing developments to search for land that is not serviced by the road network. Once that area of land has been discovered, it is assessed according to road density, proximity to existing junctions, and deviation from the start point. Roads follow parcel boundaries and try not to make large changes in elevation.
- *Connectors* roam over the existing road network sampling the distance taken to travel to a point within a given radius using a breadth first search of the road network. If this distance is too long the connector will propose a road segment between the two points, the proposed segment is subject to the same checks as extenders.



a) b) c)

Figure 30: Example output of differing city structures: a) Gridded, b) Organic & c) Mixed Gridded and Organic [12]

Road networks can be viewed evolving in real-time, and the examples shown were created in 15 to 30 minutes. Figure 15c) shows one of the main strengths of the agent based system by effectively blending between raster and suburban road styles.

Buildings: Agent Based Simulation

The generation of land usage for buildings is completed via the interaction of a number of agents but is primarily due to the work of *Developer agents*. *Developer agents* perform the role of urban developers and have similar goals: buy land, request planning permission, build and sell. A rectangular grid of patches represent the world and each patch may be occupied by a building or road. Patches are grouped into parcels under the ownership of the *building agent*. The building agent determines the zoning information of each parcel and tracks attributes of the buildings.

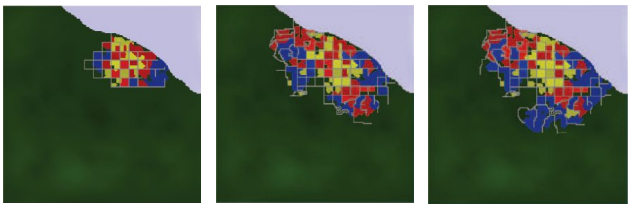


Figure 31: Development Sequence. Yellow is residential, red is commercial, blue is industrial. Roads are grey. [12]

Three distinct developer types are defined: residential, commercial and industrial. All developers seek to increase the value of their land and each developer type evaluates the value of land differently and uses a different set of rules to complete its goals. For example: residential developers seek land near the less busy areas of the road network in contrast to commercial developers who look for the busiest sections of the road network. Property is reviewed and a site is chosen. A proposal is then prepared that satisfies the clients needs and meets and the city's restrictions. The proposal must then be reviewed by the city. A developers' proposal is only successful if it passes the city regulations and makes a net positive impact on the community by providing a service or increasing the value of the land. After this process is complete the developer agent starts again looking for more property. Shown in Figure 31 are three snapshot images of the evolution of a small city from left to right.

The *CityBuilder* system creates a road network and defines land use that is then used to determine building types but does not generate actual building geometry and textures. The visualization of the city buildings is not a feature of the system but takes place externally in the proprietary SimCity game engine.

Discussion

1. **Realism:** The road network is appears realistic and has the ability to effectively transition between different road patterns, particularly the transition from central urban areas to less dense suburban areas. No buildings are generated but the land usage map appears realistic resembling real statistical data similar to that showcased in the chil.us [1] project.
2. **Scale:** The output created from the system and example shown in Figure 31 is limited in scale and is of a comparable scale to that of a village or small town rather than a city.
3. **Variation:** Different zones are supported with commercial zones using rigid block like road structures and residential areas using sprawling roads. Three different land usage and building types are defined commercial, residential and industrial. It is impossible to judge the variation achieved by those categories as the visualisation is performed by the SimCity engine which is outside of the system.
4. **Input:** A terrain height map and a water level input are required to determine the legal areas in which buildings can be placed. Other input can be specified by the user through the interactive application.
5. **Efficiency:** CityBuilder models not only the structure of a city but also its evolution and as a result of the added complexity the algorithm is computationally intensive and time consuming. A city of only limited scale similar to a village can be generated over a period of approximately 15 minutes not including the generation of any building geometry or textures.
6. **Control:** An innovative feature is available in the form of a paint tool that can be used to paint parameter values on the map. Numerical parameters such as road concentration, deviation and scale can be specified via an interactive application using the various sliders and widgets of the GUI.
7. **Real-time:** There are no real-time considerations or even a three dimensional model of the city. Visualization is provided via an external system, the SimCity engine, that uses a flat bitmap tile based game view.

The system could be easily expanded but with an algorithm of high computational complexity and it is not suited for procedural generation and could be more suitable for simulation applications.

3.4 Template Based Generation

Sun, Baciu et al propose an alternative approach to creating cities in their 2002 paper *Template-Based Generation of Road Networks for City Modeling* through the use of a collection of simple templates and a population adaptive template [18]. The basic concept of the system is that a road network template is applied to a geographic map as a plan and then the roads are deformed subject to local constraints.

Road Network: Template Based Generation

Several inputs are required in the form of 2D image maps. A colour image map which contains geographical information on land/water/vegetation is required. A grey-scale height map image to specify elevation is required. A population density map is required for the population-based template and is used to determine the varying road network density.

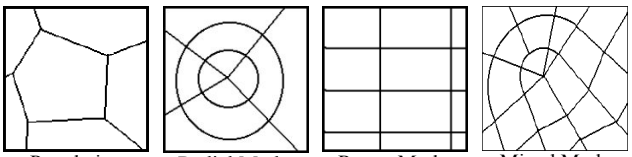


Figure 32: Road Patterns [18]

The population-based template is implemented using a Voronoi diagram. A road system is created that is representative of the population distribution. Road networks are suitably dense in highly populated areas and sparse in less populated areas. This is made possible by extracting density points from the population density input map and using the points as input sites for the Voronoi diagram. The edges or cell boundaries from the resulting diagram are used to create the interconnected road network. The other templates use procedural patterns to create the road network. The Raster Mode, Radial Mode and Mixed Mode templates serve as simplistic growing patterns, with roads starting from a defined centre point and growing in an iterative process toward the edges of a bounded area. The Mixed Mode is simply a compound of one or more of the other basic templates.

Templates define only the desired road pattern and just as road planners must respond to practical constraints, so must the pattern. Roads deviate from the supplied pattern changing direction rapidly to avoid obstacles such as water and curve gradually to avoid large changes in elevation. Roads are created in short steps, at each step the system emits several fixed length radials and selects the radial with the least variation in elevation that is in a legal zone. In the case of a tie between two radials the path of least deviation from the original path is chosen. The angles at which the radials are drawn is restricted by a freedom factor, F, which limits the maximum angle of deviation for each radial. The final shape of the road is a result of terrain deviation and the selected pattern is followed only as strictly as the freedom factor dictates it to be followed.

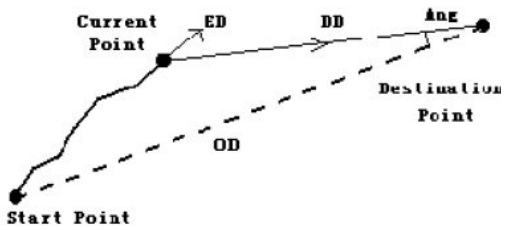


Figure 33: Lot Division Stages [18]

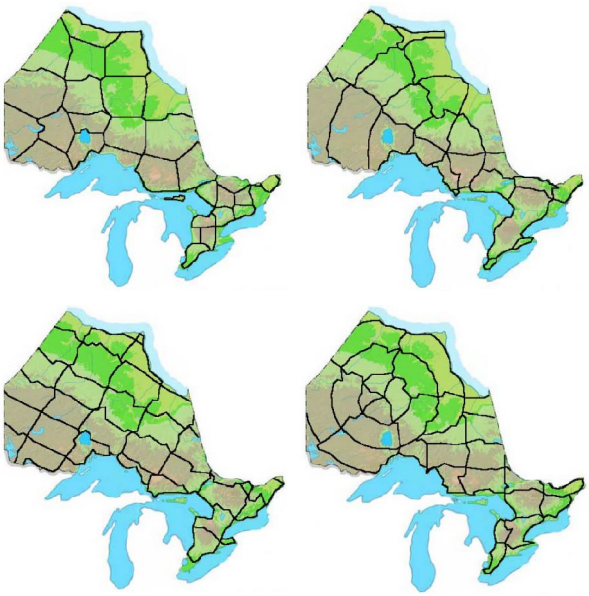


Figure 34: Results clockwise: Population-Based Template, Radial Mode, Raster Mode and Mixed Mode. [18]

Discussion

1. **Realism:** The applied template technique reflects the patterns found in cities but the results do not achieve the complexity and scale of real city road networks. The compound pattern aims to overcome the simplicity of the single patterns by combining a number of patterns, similar to a real city, but only combines two which is insufficient for the complexity of modern cities.
2. **Scale:** The examples shown in Figure 34 demonstrate limited complexity and are insufficient in scale to be classed as city scale road networks.
3. **Variation:** A choice of four templates is demonstrated and each can be deformed by the random terrain providing limited though varied output.
4. **Input:** Several inputs are required including geo-statistical data such as terrain height maps, a standard geographic map and a population density maps for the population based template.
5. **Efficiency:** No information is provided on the performance of the generation process.
6. **Control:** A reliance on statistical data and no indication of any user interaction to control the road network generation would imply that this solution is very rigid and inflexible.
7. **Real-time:** No real-time features or rendering optimisations are discussed.

3.5 Split Grammars

The *Instant Architecture* solution presented by Wonka et al. at Siggraph 2003 describes the generation of realistic buildings through the use of a new type of formal grammar called split grammars. These grammars are based on the concept of shape [11].

Buildings: Split Grammars

Split grammars are based upon the previous research and principles of shape grammars pioneered by Stiny[30]. A shape grammar is a formal grammar not unlike L-systems but it is based on the fundamental primitive of shapes rather than letters or symbols. Rules or productions map a shape or number of shapes to be be replaced by another shape or number of shapes. An initial set of shapes is supplied to start with and the rules are applied in an iterative manner.

The basic building blocks of the system and the objects that the grammar manipulates are simple attributed, parametrized, labelled shapes called *basic shapes.* A large number of rules or productions are required to transform the shapes. For the examples shown in the paper a database of around 200 rules and 40 attributes was assembled. Figure 30 shows an initial state and simple set of sample rules.

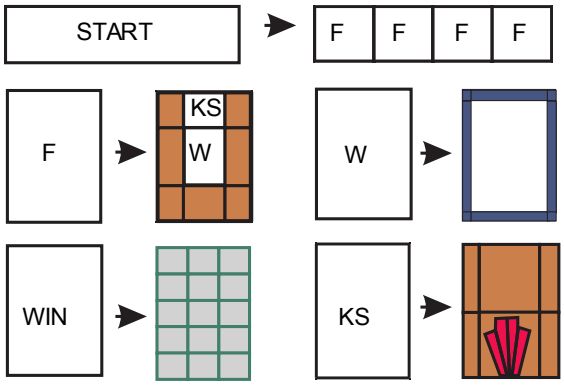


Figure 35: An example of split grammar. [11]

An initial starting state is provided and then transformed in an iterative process using rules from the database. The rules split buildings into faces, faces into structural sections, structural sections into components such as windows and so on, as shown in in Figure 30 with the end result shown of the completed derivation in Figure 16.

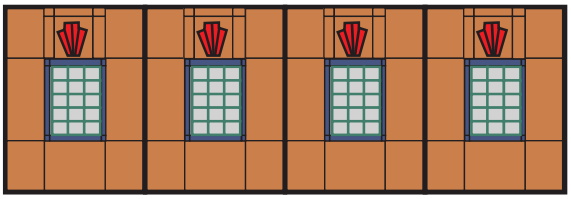


Figure 36: Completed derivation of the grammar in Figure 30 [11]

Attributes assigned to shapes are propagated from the initial state down through the system. The attributes store information about the building like its symmetry, age, use and visual properties. These are later used to render the building but are also used to help match transformation rules and find relevant replacements. In addition, a control grammar is applied that can change the attributes of basic shapes in order to apply spacial design concepts, such as setting the first floor of a building to be a shop or applying a vertical detail to a column of shapes. The resulting building models produced by the instant architecture system contain detailed local details such as window sills but also distinctive building features such as vertical details on the edges of buildings.

Figure 37: Screen shot of Instant Architecture [11]

Discussion

1. **Realism:** The split grammar technique produces very realistic buildings even going as far as to effectively recreate different styles of architecture.
2. **Scale:** The examples shown in Instant Architecture are limited in scale, they demonstrate the strengths of the system by creating a small group of buildings in a town square or centre. A high level of variation is shown in the examples but the number of buildings is limited and is not of city scale.
3. **Variation:** Building style varies greatly helping to produce very realistic output, however it is not clear how many different buildings types can be produced.
4. **Input:** The system requires substantial initial input with samples like those shown in Figure 37 requiring a database containing approx. 200 rules and 40 attributes, and took around two weeks to assemble. From this database a variety of buildings of different styles could be created and the data could be distributed with the system without requiring the user to assemble their own dataset.
5. **Efficiency:** The algorithm although complex is quite efficient creating buildings of up to 10,000 polygons in around 3 seconds on an Intel Pentium 4 at 2Ghz.
6. **Control:** No interactive editor or GUI is described but the split grammar rules can be edited in the database manually. This process is described as non trivial and requires a level of expertise and experience using the split grammars. It could well be a barrier to extending the system. There may also be constraints on the size of the system and the number of rules that it can manage with a reservation expressed that some derived designs may not even make sense if more rules are added.
7. **Real-time:** The detailed buildings that the system produces can be explored in real-time however the number of buildings on display at any one time is limited. It is clearly a limit of the system with such a high polygon count. Level of detail support would be essential to use the system for real-time applications.

The Instant Architecture solution produces realistic and detailed buildings but may require a level of expertise to operate that restrict it to an academic audience.

4 Future Research

In Section 3 we reviewed previous research into the procedural generation of cities. It is important for us to recognise the areas that can be improved in this research and to identify suitable candidates for further research. To evaluate the existing research we studied each city generation technique and assessed the systems performance according to a common set of criteria: Realism, Scale, Variation, Inputs, Efficiency, Control, Real-time provisions. After completing this analysis, we have found that previous research efforts have made good progress on a number of difficult goals by achieving a high level of variation, realism and scale. However, city generation research is by no means complete and we identified a number of areas which can be improved on by future research.

An accessible city generation system is difficult if the operation of the system requires a high level of expertise or if complex input such as geo-statistical data is a prerequisite to using the system. The City Engine system [23] is capable of producing visually sound results but the road network and buildings are generated using a complex set of rules, images maps and geo-statistical data. The Split Grammars technique proposed by Wonka et al.[11] requires the creation of a large set of complex architectural rules before building generation can begin. The agent based simulation technique also uses a large and complex set of pre-determined behaviours to specify how the agents act. Varied results can be obtained without strictly requiring changes to the city generation systems. But in order achieve results of a similar quality to some of the examples shown a level of expertise and in-depth knowledge of each system is required that is not possible for the general user.

The existing city generation solutions do not provide interactive, tactile or close control over the generation process. The grid layout system proposed by Greuter et al.[19] is very restrictive allowing only the grid spacing to be adjusted by the user. Growth based algorithms such as L-systems[23] or agent based simulation[12] are difficult to control due to their evolutionary nature. Both growth based generation solutions use a system of image maps to provide incentives for growth in certain areas of the city. An additional layer of control is specified in the form of numerical parameters that influence factors such as road branch distance, road branch angle, etc. The template based approach uses a selection of basic procedural templates to create a road network. The templates cannot be edited by the user and no user interaction is documented.

A city is a large and complex model and cannot be rendered easily in real-time on commodity hardware. In order to render such a complex model, optimization systems such as level of detail, culling or paging are necessary. Out of the city generation solutions studied only Greuter's system provides real-time optimisations via the use of view frustum filling and geometry caching.[13] The view frustum filling technique is possible by using a simple regular grid road network but is not applicable in its current form to more complex road networks.

We have reviewed the city generation systems and discussed some of the areas that warrant further research. From this discussion we have obtained a key list of goals for our city generation solution to build and improve upon existing work.

- Accessible – input data such as geo-statistical data or complex architectural rules should not be required to use the system.
- Interactive – the system should be capable of fully autonomous generation but also facilitate interactive control.
- Real-Time – for efficient rendering optimization techniques like culling, paging and level of detail should be implemented.

To summarize, the goal for our research is to create a city generation system suitable for real-time applications that is capable of creating realistic, varied and large scale cites in an efficient manner while remaining accessible to non-expert users.

Our current design comprises of three major components: *primary road generation*, *secondary road generation* and *building generation*. All three components are united within a standalone application providing interactive control over the city generation process.

The primary road generation component utilises procedural templates Sun et al. that encapsulate common city road network patterns such as raster, radial, hierarchical and cellular. In addition to the previous template based solution the templates are dynamically editable and form only the high level roads. Each road network template can be applied to a terrain as an interconnected graph with edges of the graph, the roads, automatically deformed by terrain characteristics such as gradients, water levels and other obstacles. The resultant road network graph is editable using an interactive 3D interface. Streets can be added, deleted and moved using junctions as control points for easy manipulation. Editing the control points affects not only the primary road network but also the secondary roads within providing a tactile method to control all road generation.

Cells resulting from the division of the city by the primary road network form the basic units upon which the secondary road generation component operates. A technique similar to that used in [23] based on L-systems can be applied inside the cell although several different algorithms including agent based approaches, and simple procedural templates can be applied within a single city. Global and local parameters are adjusted to specify the behaviour of generation algorithms in a citywide and/or local cell scope. To optimize rendering of the city we propose to page cells similar to terrain paging, the primary road network forms a skeleton and the secondary road network cells within can be pre-emptively generated and loaded on demand using a cell adjacency model.

Buildings can be placed on the lots created from the secondary road network and can be constructed using a generative grammar such as L-systems. Several different building usage types will be supported including commercial, industrial, and residential. Building structures are generated at run time and can thus provide a substantial reduction in memory usage by storing simple generation data and generating large complex geometry only when required. This geometry generation can be applied to construct several variants of buildings depending on the instantiation parameters used. L-systems refine a basic model into a complex model by applied a series of productions iteratively. A range dependant level of detail can be provided by including an instantiation parameter that specifies the number of L-system iterations proportional to the distance between the camera and building.

At present an interactive application has been built that implements the primary road generation component and partial secondary road generation enabling basic road network manipulation and interactive editing. Current work involves implementing complete secondary road generation and generative grammars for building generation. Real-Time rendering provisions including *cell paging*, *runtime geometry generation* and *L-system level of detail* are currently catered for and we would like to further explore and test these optimisation schemes.

References

[1] UrbanLab. www.chil.us = Chicago, Illinois, USA. 2006

[2] Pascal Mueller. City Engine Wiki. 2006

[3] . Cellular Texture Basis Functions implementation DarkTree procedural software.. 2006

[4] Jon Hare. Sensible Software. http://www.purplesensi.co.uk 2006

[5] Interactive Data Visualization Inc.. SpeedTree RT. http://www.speedtree.com 2006

[6] Ken Musgrave, Pandromeda.. Mojo World Applications. http://www.pandromeda.com/products/ 2006

[7] http://www.sidefx.com. Side Effects Software. Manufacturer of Houdini.. 2005

[8] Stefan Greuter, Nigel Stewart, Geoff Leach; 2004. Beyond the horizon. In Image Text and Sound Conferance 2004, , .

[9] . http://www.planetside.co.uk/terragen. 2004

[10] International Scene Organization. Scene Awards 2004. http://scene.org/awards.php?year=2004. 2004

[11] Peter Wonka, Michael Wimmer, Francois Sillion, William Ribarsky; 2003. Instant Architecture. In , Siggraph, .

[12] Thomas Lechner, Ben Watson, Uri Wilensky, Martin Felsen; 2003. Procedural City Modeling. In , , .

[13] Greuter S., Parker J., Stewart N., and Leach G.; 2003. Real-time procedural generation of `pseudo infinite' cities. In Proceedings of GRAPHITE 2003, ACM Press, 87-95.

[14] Sylvain Lefebvre, Fabrice Neyret 2003. Pattern Based Procedural Textures.

[15] Javier Lluch, Emilio Camahort, Roberto Vivó; 2003. Procedural Multiresolution for Plant and Tree Rendering. In , , .

[16] John Spitzer, Simon Green and NVIDIA Corporation; 2003. Noise and Procedural Techniques. In Proceedings of Game Developers Conference 2003, GDC, .

[17] David S. Ebert; F Kenton Musgrave; Darwyn Peachy; Ken Perlin; Steven Worley; Texturing & Modelling - A Procedural Approach. Morgan Kaufmann 2003.

[18] Jing Sun and Xiaobo Yu and George Baciu and Mark Green; 2002. Template-based generation of road networks for virtual city modeling. In , , .

[19] Stefan Greuter, Nigel Stewart, Jeremy Parker, Geoff Leach; 2002. Undiscovered Worlds - Towards a Framework for Real-Time Procedural Gener.... In MelbourneDAC 2003 Proceedings, ACM Press, .

[20] Wolfgang von der Linden, Ewald Schachinger. Computersimulations. http://itp.tugraz.at/LV/wvl/Comp_Simulationen/. 2002

[21] Evan Hart, ATI Research; 3D Textures and Pixel Shaders, ShaderX vertex and pixel tips and tricks.. 2002.

[22] TomasAkenine-Möller, Eric Haines; Real-Time Rendering. A K Peters, Ltd. 2002.

[23] Yoav I H Parish, Pascal Mueller; 2001. Procedural Modeling of Cities. In , , .

[24] Ken Perlin. Making Noise, http://www.noisemachine.com/talk1/index.html. 1999

[25] Michael F. Barnsley; Fractals Everywhere. Morgan Kaufman 1993.

[26] Lindenmayer and Prusinkiewicz; 1990. The Algorithmic Beauty of Plants.

[27] Prusinkiewicz, P., Lindenmayer, A., Hanan, J. S., et al.; The Algorithmic Beauty of Plants. Springer-Verlag 1990.

[28] Ken Perlin; 1985. An Image Synthesizer, in Proc ACM SIGGRAPH. In , , 287-296.

[29] Benoit B. Mandelbrot; The Fractal Geometry of Nature. W.H. Freeman & Co. 1982.

[30] Stiny, G.; Introduction to shape and shape grammars..Environment and Planning B 1980.

[31] Alexander C, Ishikawa S, Silverstein M; A Pattern Language: Towns, Buildings, Construction. Oxford University Press 1977.

[32] A. Lindenmayer; Mathematical models for cellular interaction in development, Parts I and II. 1968.

[33] Kevin Lynch; The Image of the City. Cambridge: MIT Press 1960.