A Time-line Approach for the Generation of Simulated Settlements

Benjamin Williams and Christopher J. Headleand
School of Computer Science
University of Lincoln
Lincoln, UK
Email: bwilliams@lincoln.ac.uk

Abstract—**In this paper we present a model for procedurally generating virtual settlements populated with roads, land parcels and buildings. Our model improves on existing research by considering historical influence on settlement growth. To do this, an interactive time-line is used, allowing for a designer to specify a number of architectural periods. These architectural periods are then used in the generation process, giving the designer a robust tool to interactively generate photo-realistic urban scenes. Our results show that a variety of settlement types and sizes can be generated. In addition, we demonstrate that road patterns within real-world settlements can be created using our system.**

Index Terms—**Procedural Content Generation; Virtual Worlds; Procedural Cities**

I. Motivation

There has been a large body of research into procedural modelling algorithms, in the field of visualization and computer graphics. Researchers in this field have discussed algorithms to create a variety of content, including terrains [1], [2], trees [3], [4], buildings [5] and entire in-game levels [6], [7]. These algorithms take away the task of manually creating assets to be rendered within a scene, and instead focus on how assets can be created on-demand. They also typically allow for a large set of core assets, through parametrization. This permits a designer to create a large quantity of visually disparate objects with little or no effort, other than the creation of the original algorithm.

The large demand for high fidelity scenes has been the driving force for advancements in graphical technology [8]. High quality scenes are utilized in a number of different areas, especially the games development industry and television/film production. As a result, procedural modelling approaches are used in industry as an alternative to traditional approaches. An example of this is the *MASSIVE* multi-agent system[1], which was used to procedurally generate large crowds in the production of the *Lord of the Rings* trilogy of films.

A frequent element seen in games and television are settlements, from sprawling cities to smaller villages and hamlets. Designing a settlement manually is a time-consuming process, largely due to the scale of the scene and the amount of elements used in its composition. As a result of this, interest in researching the procedural creation of cities and other settlements has risen in recent years.

[1]http://www.massivesoftware.com/

An aspect often overlooked by existing research in this area is providing historical context for the generated settlements. Providing historical context permits the environment designer to define time periods which influence the development of a city, e.g. architectural styles or urban planning decisions.

Currently, state-of-the-art algorithms for generating settlements only consider the creation of a city in a single period. Modelling techniques which develop settlements over time do exist [9]–[11], however very little of these approaches use historical context to influence the development of a settlement. Historical influence plays an important role in the development decisions took when developing a city. For example, economic depression has a large impact on development decisions took throughout a certain historical period, due a shortage of resources [12].

The aim of this paper is to elaborate on this, to provide a system which permits the automatic generation of virtual settlements using a historical time-line. This time-line will feature various interactively created historical periods, which will directly influence the growth of a settlement. This system will allow the designer to interactively create virtual settlements with a high degree of control over the resulting output. It will also give the designer a robust tool to model different road growth patterns and architectural periods, whilst also simulating the growth of many settlements over time.

II. Related Work

The procedural generation of virtual settlements has a variety of applications in many areas. For example, these applications include creating testing environments for robot navigation [13], [14], an interactive tool to aid games development [15] and even as a tool to aid in urban design [16]. This section will briefly review previous work into this field of research.

A. City Generation

An early system to generate entire virtual cities was presented by Parish and Müller, who introduced the *CityEngine* system [17]. This system takes input maps, such as population density and an elevation map, and uses these maps to generate a virtual city. To do this, the system uses a street graph and extended L-Systems to grow a road network guided by the various input maps.

Later approaches to this problem build on this work, by improving various aspects of city generation. For example, Vanegas et al. [18] introduce a system which allows for the interactive editing of these input maps, by using an agent-based approach to city generation. This system also takes into account geographical influences, such as land value and job density, which are largely based on the *UrbanSim* model [19]. A similar interactive system is introduced by Chen et al. [20], in which road networks are modelled using a tensor field interactively designed by the user. Yang et al. [21] propose a template matching method for the creation of road networks, by using hierarchical domain splitting. Similarly, Nishida et al. [22] propose a technique in which road networks are grown using the transformation of real-life road network examples. Following this, land parcels are computed and buildings are placed to produce a procedural generated city.

One area of research related to this is modelling urban land usage and zoning. In the work of Lechner et al. [23] the authors build on their previous work [24] to produce a system which procedurally models patterns of land usage in cities. Groenewegen et al. [25] expands on this to create cities using urban land usage modelling to guide city development. Other areas of research include the real-time generation of cities, such as the work of Greuter et al. [26], [27] in which the authors proposed a method for creating cities pseudo-infinitely.

Recent research into this domain has led to some interesting approaches. In the work of Garcia-Dorado et al. [28], the authors focus on the use of realistic weather simulation to enhance the fidelity of procedurally generated city scenes. Peng et al. [29] propose an algorithm to automatically generate street layouts within a city. The authors achieve this by using an integer-programming based approach to optimize road networks with levels of decreasing coverage. These road networks can then be used with an external program to place parcels and buildings.

Whilst there is a large body of research into the generation of cities, only a handful of approaches consider the growth and development of a city over time, with the majority of approaches simulating the development of cities at a fixed point in time. Interpolation techniques between maps exist [11], [30], which use image-based interpolation to model settlement development. Furthermore, Weber et al. [10] introduce a procedural city generation technique with emphasis on the development of the city over time.

These approaches however, do not consider the simulation of many cities together, and do not take into account neighbouring settlement around the city. Beneš et al. [9] tackle this issue by introducing a city generation system which both develops cities over time, whilst also taking into consideration the influence of neighbouring cities. To do this, the algorithm creates an initial road network, and grows it using traffic simulation. This approach however, does not simulate settlements separately, and only outputs a 2-dimensional representation of roads and parcels.

B. Village Generation

Whilst there is a large body of research into large-scale city development, the area of generating smaller, non-urban settlements (such as villages and hamlets) is an under-researched area. Through a survey of the literature, we discovered only two papers which specifically focus on generating non-urban settlements.

Glass et al. [31] pioneered the research into this domain, by demonstrating a technique to duplicate road patterns in African settlements. This was achieved by the application of Voronoi tessellation and L-Systems to approximate road patterns. This approach however, did not consider the generation of settlements, but instead focused on approximating the road patterns found in real-life aerial imagery.

This was improved upon by Emilien et al. [32], who demonstrate a technique to procedurally generate small villages. Firstly, the user provides a number of heatmaps for the terrain, such as proximity to water and terrain elevation, which are used to influence the placement of roads. Following this, an anisotropic conquest method is used to find land parcels adjacent to each road, and procedurally generated buildings are placed and correctly orientated within each parcel.

C. Historical Context

Another under-researched problem is the use of historical context in the generation of settlements.

To our knowledge, the only attempt to address this is in the work of Beneš et al [9]. In this paper, Beneš et al. present a model for the generation of road networks which takes into account some level of historical context when building a city. However, this approach only considers the influence on road networks and not architectural styles, nor the generation of any contextual information. Furthermore, the graphical output of this method is only limited to road networks.

III. Settlement Generation

In the following sub-sections we will introduce the various steps of our system, to procedurally generate settlements. Firstly, our method for initially placing settlements will be discussed, followed by the introduction of a historical timeline. Then, our technique to simulate road network growth and the generation of land parcels will be discussed. Finally, we will introduce our algorithm to create buildings within these land parcels.

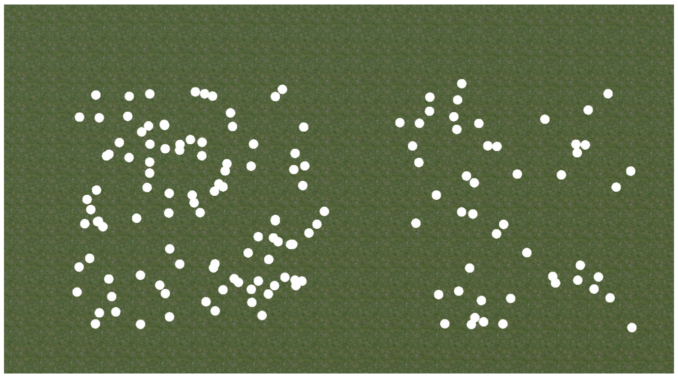
A. Settlement Seeding

Our system is flexible to allow for multiple settlements to be concurrently generated within a single environment.

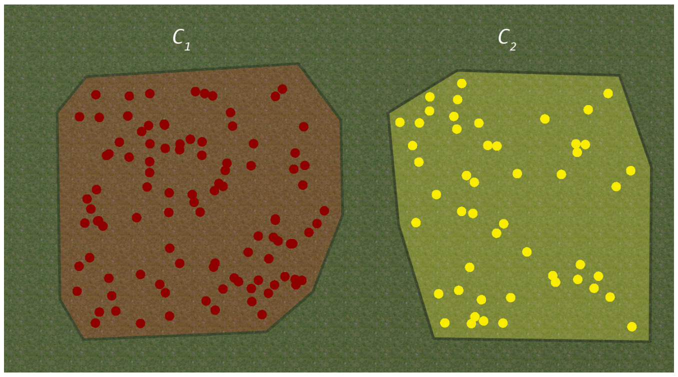
The starting position of the generation referred to as a *'seed position'*, as it is the point from which a settlement grows. To find these seed positions, we use a method based on the seeding algorithm introduced by Emilien et. al. [32]. The variation we use follows similar steps to the original algorithm, which will be described below.

In the first step, a number of randomly sampled seed points are chosen within the environment. Following this, each of these points is scored by using the function $S(p)$, where the steepness s of a point in the environment p is established. In addition, the score $S(p)$ is checked against a threshold value t: if the score $S(p)$ is greater than t, then it removed from the environment. The purpose of this step is to filter out points which exist on steep terrain, keeping only valid candidate points.

At this point, a number of candidate seed positions have been found which exist on flat terrain. The next step is to filter these further, to select the points which the settlements should grow from. Ideally, each of these positions should be placed at some distance away from others, to avoid settlements overlapping and allowing them to grow independently. One solution to this is to group clusters of points together which are close to one another. These groups of points can then be used to determine the initial starting size and position of the settlement. An additional benefit to using this approach is that the number of settlements used in the simulation is also determined by this clustering algorithm.



(a)



(b)

Fig. 1. A demonstration of the clustering algorithm, which groups point based on the nearest neighbouring point. In Fig. 1a seed points are randomly sampled within the environment, then in Fig. 1b, are grouped using the algorithm discussed in this section.

To group the points, a neighbourhood distance d_{n} is defined. This distance represents the local neighbourhood of each point. Each point p in P is then compared to each other point k in P where $k \neq p$.

If $\mathrm{d}(p,k) > d_{\mathrm{n}}$, then the point k is ignored as it is outside the local neighbourhood of p. Otherwise, p is assigned the group of k, or in the case that k has no assigned group, p is assigned a new group. If there are no points within the local neighbourhood of p, then p is assigned a new group by default, as it is an outlier.

To better illustrate this, a pseudo-code algorithm of this method can be seen in Fig. 3, and an example of its usage can be seen in Fig. 1. Following this process, each group of points represents a settlement. For example, in Fig. 1, there are two groups of points which represent two settlements. To obtain the point at which a settlement should be placed, the mean of all the points for that group is took and used.

At this point, the settlement seed positions to grow from have been found, and each settlement is ready to be simulated over time. The next step involves the actual simulation of each settlement.

B. Historical Timeline

A major component of our system is the addition of a historical time-line, with which the designer can specify different architectural periods to influence the development of each settlement.

To do this, the designer can use a graphical user interface to interactively create an arbitrary number of time periods. These time periods can be given a duration, and also sorted in a chronological order. The total duration of this time-line is simply the sum of the duration of each time period.

As the simulation runs, the elapsed time is incremented and recorded. This is then used to calculate the current time period for this time frame, by running through each period on the time-line and testing if the current elapsed time lies within the start/end time of that period.

The time periods which are created along the time-line are then used in the simulation process itself, to influence the development of each settlement. This is achieved by specifying a set of parameters to be used in the simulation, along with the time period – such as road length, building types, and so forth. By doing this, the system can easily calculate the current time period and use the parameters supplied with this period to influence the development of each settlement in some way. The types of parameters supplied with each time period will be discussed in greater detail later in this paper.

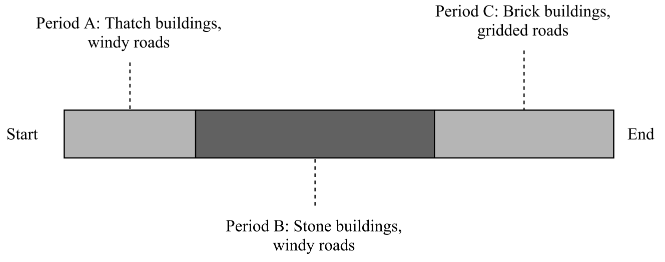

Period A: Thatch buildings, windy roads
Period C: Brick buildings, gridded roads
Start
End
Period B: Stone buildings, windy roads

Fig. 2. A visual example of a time-line with three architectural periods.

```
procedure CLUSTER($P, d_n$)
    $i \leftarrow 0$
    for $p \in P$ do
        for $k \in P$ do
            if $\mathrm{d}(p,k) > d_n$ or $p = k$ then
                continue
            end if
            $i \leftarrow i + 1$
            if group $(k) \neq$ NONE then
                group($p$) $\leftarrow$ group($k$)
            else
                group($p$) $\leftarrow$ nextGroupID
            end if
        end for
    end for
    if $i = 0$ then
        group($p$) $\leftarrow$ nextGroupID
    end if
end procedure
```

Fig. 3. Pseudo-code for the clustering algorithm used to group seed points which are close.

C. Road Network Growth

One of the most important aspects of generating virtual settlements is the creation of a road network. The road network of an urban area defines the connectivity between roads, as well as their location within the environment.

For the road network, we use the same approach took by Parish and Müller [17], of using a directed street graph $G = (V, E)$ to represent the connectivity of junctions within a settlement. By associating the vertices V with the junctions of the road network, roads can easily be defined as the edges E of the graph. This graph is then modified throughout the simulation to create new roads, and connect existing roads together.

Furthermore, each settlement's road network is initialized with a single vertex, which is placed at the seed position found for this settlement in Section 3.1. A random point is chosen around this vertex, forming an initial road. This enables the settlement to initially grow from this position outwards.

1) Road Creation: We simulate road network growth for each settlement in the following way, at each frame. Firstly, a junction within the road network is selected randomly as p. Then, the junctions connected *from* this junction p_{out} are found, along with the junctions connected *to* this junction p_{in}. The mean position of the point lists p_{out} and p_{in} are found with (1), as $\sigma_{\text{out}} = f(p_{\text{out}})$ and $\sigma_{\text{in}} = f(p_{\text{in}})$.

$$f(\lambda) = \begin{cases} \frac{\lambda_1 + \lambda_2 + \cdots + \lambda_n}{n} & n > 0 \\ p & \text{otherwise} \end{cases} \tag{1}$$

Following this, the angle θ is found between σ_{out} and σ_{in}, by using (2). A diagram illustrating this process can be seen in Fig. 4.

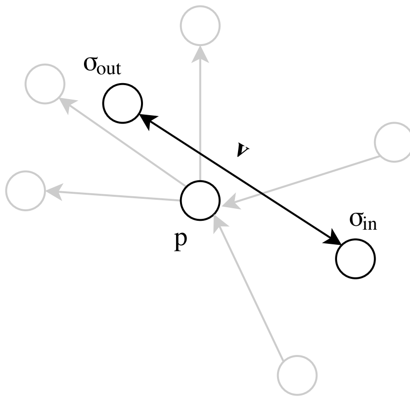


Fig. 4. A diagram showing σ_{out}, the mean position of the junctions connected out from p, and σ_{in}, the mean position of the junctions connected into p. Here θ is simply the angle of the vector $\mathbf{v}$.

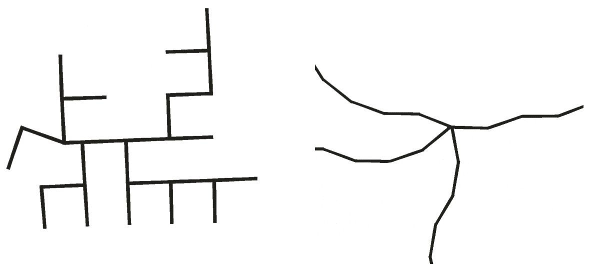
$$\begin{aligned} \mathbf{v} &= \sigma_{\text{out}} - \sigma_{\text{in}} \\ \theta &= \operatorname{arctan2}\left(\mathbf{v}_x, \mathbf{v}_y\right) \end{aligned} \tag{2}$$

At this point, the angle θ is used to find a new point offset from p. This point will become a new junction, and will be connected to p. To do this, two angular offsets are used. The first, θ_{offset}, is an offset which is *always* added to θ. The second, θ_{rand}, is an offset which is randomly chosen in the range $[-\theta_{\text{randmax}}, \theta_{\text{randmax}}]$. An illustration of the effects of altering θ_{offset} can be seen in Fig. 5.

To find the new point k, another value d is used, which defines the distance to offset k from p. The value of d is randomly chosen in the interval $[R_{\text{mindist}}, R_{\text{maxdist}}]$.

$$k = \begin{bmatrix} \sin\left(\theta + \theta_{\text{offset}} + \theta_{\text{rand}}\right) \\ \cos\left(\theta + \theta_{\text{offset}} + \theta_{\text{rand}}\right) \end{bmatrix} d \tag{3}$$

Following this step, the closest junction in the road network to k is found as c. If the distance between k and c is less than d, then k is discarded. Similarly, if the path between p and k intersects a road at any point, k is also discarded. Otherwise, k is added as a new vertex in the street graph G, and is connected from p, creating $p \rightarrow k$.



(a) $\theta_{\text{offset}} = 90°$ (b) $\theta_{\text{offset}} = 20°$

Fig. 5. An example of two road networks built with different values of θ_{offset}. Notice how simply altering this value results in very different road layouts.

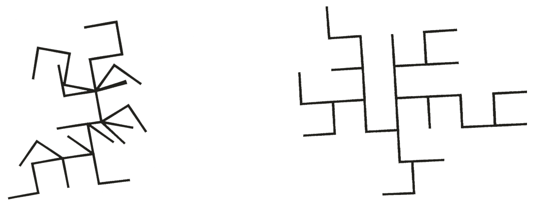

(a) Without minimum distance or intersection checks (b) With minimum distance and intersection checks

Fig. 6. An example of two road networks created with the same parameters, but with/without minimum distance and road intersection checks.

This step is took in order to maintain a suitable distance between junctions, and ensure that the offset angle of the road is correct. An example of this process can be seen in Fig. 6.

After this step, a road has successfully been added to a settlement. Additionally, this road is given an age to identify which time period it was built in, to be used later for building placement and land parcel generation.

Following the completion of a single frame, an additional step is performed to connect together existing roads. This is achieved by finding roads which are cul-de-sacs (junctions with either an indegree or outdegree of 0) and connecting them together if they are within a distance R_{cdist} of one another. If the path between these two junctions intersects any road, then this connection is not made.

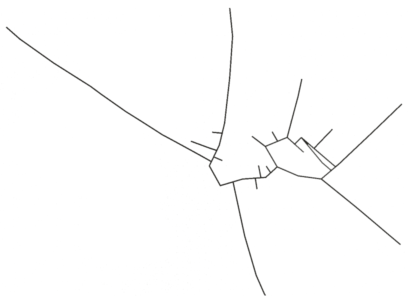
2) Designer-Controlled Parameters: As mentioned previously, our system allows for multiple time periods to be freely created along a timeline by a designer. Each of these time periods includes parameters to be used in the generation process, which influence the appearance of the resulting settlements.

In particular, the values of each parameter used in the creation of roads can be specified by the designer for each time period. A full table of these can be seen in Table I. This allows for multiple periods to be specified by the designer, with different architectural rules for the generation process to follow. For example, the designer may specify two time periods – one in which roads are gridded and short, and another in which roads are windy and long.

By allowing the designer to specify time periods with choosable parameters, the designer is given a powerful tool to generate a diverse range of road layouts, with a large degree of control over the output.

TABLE I
A FULL LIST OF VARIABLES WHICH CAN BE SPECIFIED IN EACH TIME PERIOD, FOR ROAD NETWORK GROWTH.

Parameter	Description
θ_{offset}	A constant angular offset to use when offsetting a point from the road, for road creation.
θ_{randmax}	The maximum random angular offset to use.
R_{mindist}	The minimum length of newly created roads.
R_{maxdist}	The maximum length of newly created roads.
R_{cdist}	The minimum distance between the created point and the vertex which was selected, to maintain a minimum distance between vertices.


(a)

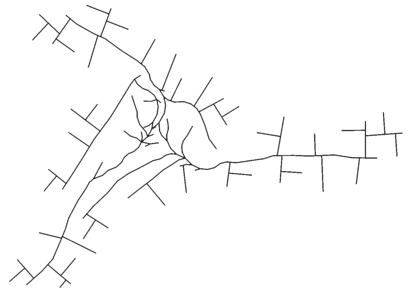

(b)

Fig. 7. Two examples of road networks built using our system. Each of these examples was built using multiple time periods and different sets of parameters.

Furthermore, our system allows the designer to limit road extensions to roads built in a specific time period. More specifically, the road growth algorithm can be configured to only select junctions built within a specific time period, rather than using any junction. This gives a greater degree of control to the designer, allowing them target and extend roads built in a specific time period. Two examples of road networks built using this functionality can be seen in Fig. 7.

D. Land Parcel Generation

Once the roads have been generated, the next step of the simulation is to allocate plots of land for buildings to be placed within. We refer to these plots of lands as *land parcels*. Ideally, these land parcels should be correctly orientated alongside roads, and minimize the amount of empty space between roads.

Additionally, another goal to consider is to eliminate overlap between parcels. This rule embodies the fact that plots of land are mutually exclusive, and any specific area of land belongs to a single owner *only*. Furthermore, the area of every parcel should not intersect the area of any road.

With these considerations in mind, we created an algorithm which creates land parcels which efficiently maximize empty space, for any arbitrary road network structure. Our algorithm also observes these considerations, and produces polygons which do not intersect one another, or the road network.

To do this, we use a method which casts rays orthogonal to the direction of a road, along its length. More specifically,

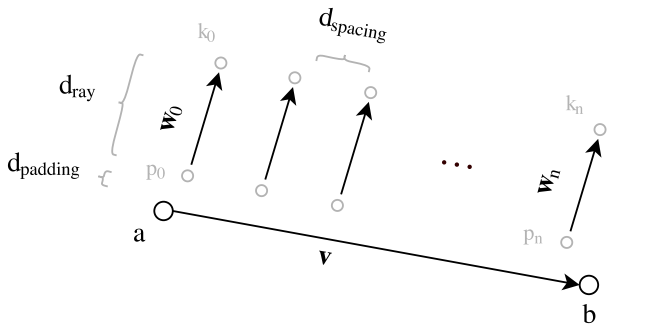


Fig. 8. An example of rays being cast along a road between a and b.

given the position a where a road starts, and b where it ends, rays are cast orthogonal to the vector $\mathbf{v} = (b - a)$.

The origins of these rays are spaced along $\mathbf{v}$ by a fixed distance d_{spacing}, in the direction of $\mathbf{v}$. Additionally, these rays are also position in the direction of the orthogonal vector $\mathbf{u}$ from $\mathbf{v}$ by a distance d_{padding}. Finally, each ray is cast from this origin for a distance d_{ray}. A diagram illustrating each of these variables is can be found in Fig. 8.

Each ray is cast iteratively, along $\mathbf{v}$. If the ray intersects the edge of another parcel, or any road, then the magnitude of the ray is reduced by the distance d_{padding}. This can be seen in Fig. 8.

Following this, if the magnitude of the ray is negative, or lower than a threshold d_{mindist} then it is not considered. Otherwise, the ray's origin position p and end position k are added as vertices in a polygon, which represents a land parcel. An illustration of this can be seen in Fig. 9a.

Following this process, the vertices of this polygon are connected together, forming an enclosed space. This can be seen in Fig. 9b. At this point, this polygon represents a land parcel to place a building within, and is added into the simulation.

This process is then ran for each generated road, on both sides, to produce land parcels for every road within the settlement. An example of output from this algorithm with an arbitrary road network can be seen in Fig. 10.

E. Building Placement

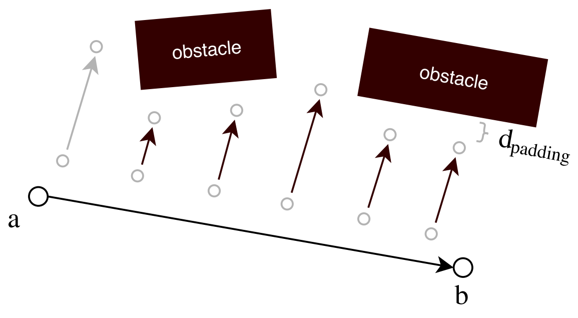
The final step in the simulation is to place 3D meshes of buildings into the environment, using the land parcels generated in the previous step.

These placed buildings should be situated entirely within a land parcel, with no part overstepping its boundaries. Furthermore, the space within each land parcel should be maximized, allowing multiple buildings to be placed alongside one another.

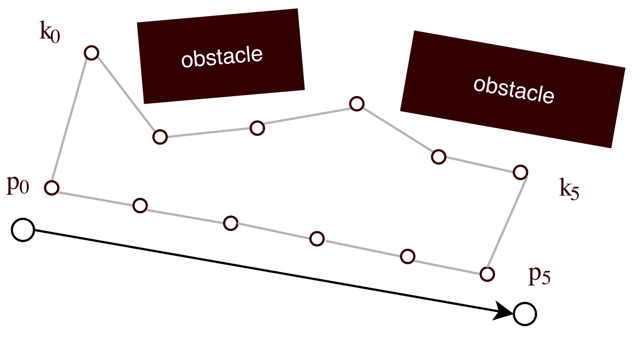
To do this, the relative dimensions of the land parcel are firstly took. Then, a 3D mesh of a building is selected at random for this time period. This selection process is then repeated if the dimensions of the mesh oversteps the land parcel, until either one is found, or none are found. If no meshes are found, then no buildings are placed within this parcel.

This mesh is then tested from the left side of the parcel until an available space is found. Once this building is placed, it is correctly rotated to face the road which runs alongside the parcel. To test for an available space, positions from the left edge are sampled at increasing distances, and the placement of the building is tested. If the building intersects another building, or exceeds the parcel boundaries, then the distance is increased and tested again. This is repeated until there is no available space within the parcel.

This process is ran for each land parcel generated from the previous step. Additionally, a number of trees are placed

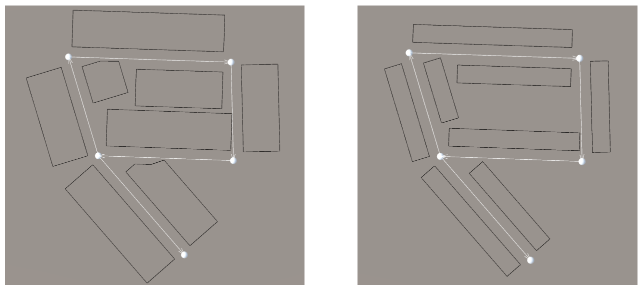


(a) Scaling ray length to eliminate intersection of obstacles.



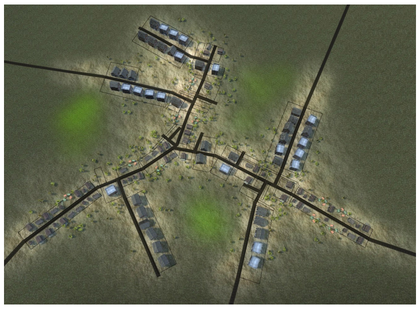
(b) The vertices are connected, to create a land parcel.

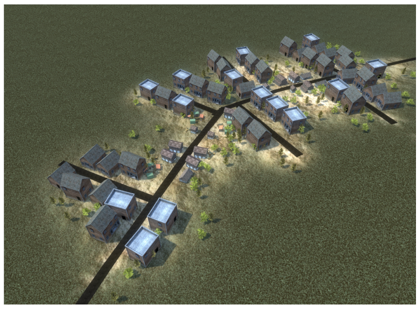
Fig. 9. A diagram showing rays being scaled due to intersection with obstacles, and being connected together to create a land parcel.



(a) $d_{\text{ray}} = 10$ (b) $d_{\text{ray}} = 5$

Fig. 10. An example of land parcels generated using our algorithm. Notice the different values of d_{ray} and their effects on the land parcel size.


(a) An aerial view of a generated village, with highlighted land parcels.


(b) A village built along an arterial road.

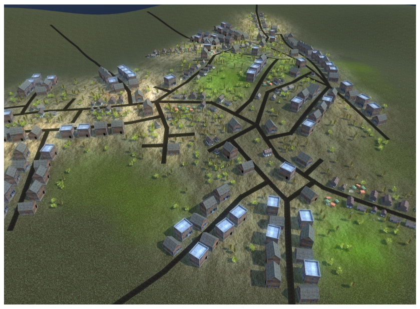

(c) A small town consisting of many streets and avenues.

Fig. 11. Examples of different types of settlements generated by using our system.

around each parcel, and removed if they intersect any road or land parcel.

IV. Results & Discussion

We implemented our procedural settlement system in C#, using the Unity3D[2] game engine to render the scene. Our system is capable of producing photo-realistic depictions of urban areas, with a variety of road network patterns and buildings.

For example, Fig. 11 shows a variety of different settlement types and sizes which can be created through the use of our system.

A. Evaluation

To evaluate our results, we compare output from our system against real-life aerial imagery and map data. This approach is often taken by similar papers to validate the accuracy of a generated settlement to a real-world example [18], [24], [32].

In comparing results from our system against geographical data, we found that a number of different road patterns found in real-world examples can be recreated using our system.

Residential culs-de-sac are typically built in groups orthogonal to tertiary roads, with a fixed spacing between each street. An example of this in the real world can be seen in Fig. 12a. Our system can be configured to recreate the qualities of these streets, producing realistic streets adjacent to a road. This can be seen in Fig. 12b.

Another type of common road pattern are intersections. These occur when two or more roads meet and intersect. A real-life example of this can be see in Fig. 13a, where two arterial roads meet. Our system is also capable of modelling these types of road, as seen in Fig. 13b.

Furthermore, our system is capable of producing settlements similar to small villages found in real geographical data. For example, we used a number of time periods to influence road growth and recreate *Cundall*, a small village in the UK.

This can be seen in Fig. 14.

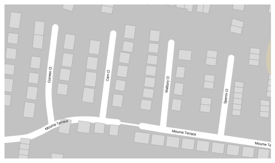
[2]https://unity3d.com/

V. Historical Context and Virtual Museums

To conclude, in this paper we presented a novel approach to the problem of procedural settlement generation. We expanded on existing work by introducing an interactive method to generate settlements through the use of a historical time-line. This time-line allows the designer to specify architectural periods, influencing building placement and road growth patterns.

Our inspiration for this approach is drawn from the lack of attention to historical context in settlement generation by related works. We believe historical context plays an important role in the development of settlements, and have created an approach which considers this.

The implementation of our system produces photo-realistic


(a)

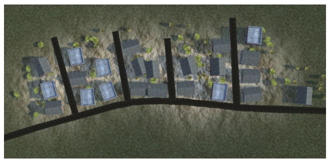
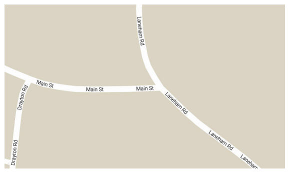

(b)

Fig. 12. A diagram showing the ability of our system to generate realistic culs-de-sac.



(a)

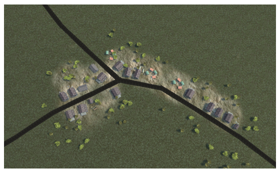
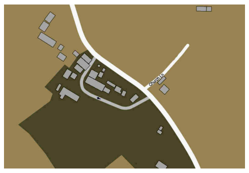

(b)

Fig. 13. A diagram showing the ability of our system to generate realistic cross-roads.


(a)

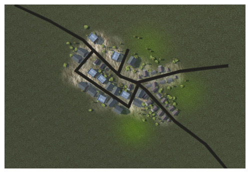

(b)

Fig. 14. A diagram showing the ability of our system to generate small villages, similar to a real-world example.

renderings of virtual settlements, including roads, land parcels, buildings and foliage. We validated the results from our system by comparing against real-life map data. Through this, we demonstrated that our system is capable of producing a variety of similar road growth patterns similar to those in real-life, giving the designer a robust tool to develop photo-realistic urban scenes.

We have presented early work to the problem of procedural settlement generation with historical context. The next phase of this research is to conduct a large-scale evaluation with users. In future work, we will consider other areas such as using historical context to create a virtual museum, which outlines the history of a generated settlement. We would also like to focus on inter-settlement influences, such as trading routes and traffic simulation.

References

[1] G. Cordonnier, J. Braun, M.-P. Cani, B. Benes, E. Galin, A. Peytavie, and E. Guérin, "Large scale terrain generation from tectonic uplift and fluvial erosion," in *Computer Graphics Forum*, vol. 35, no. 2. Wiley Online Library, 2016, pp. 165–175.

[2] J. Doran and I. Parberry, "Controlled procedural terrain generation using software agents," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 2, pp. 111–119, 2010.

[3] K. Xie, F. Yan, A. Sharf, O. Deussen, H. Huang, and B. Chen, "Tree modeling with real tree-parts examples," *IEEE transactions on visualization and computer graphics*, vol. 22, no. 12, pp. 2608–2618, 2016.

[4] J. Lluch, E. Camahort, and R. Vivó, "Procedural multiresolution for plant and tree rendering," in *Proceedings of the 2nd international conference on Computer graphics, virtual Reality, visualisation and interaction in Africa*. ACM, 2003, pp. 31–38.

[5] T. Adão, L. Magalhães, E. Peres, and F. Pereira, "Procedural generation of traversable buildings outlined by arbitrary convex shapes," *Procedia Technology*, vol. 16, pp. 310–321, 2014.

[6] D. Ashlock, C. Lee, and C. McGuinness, "Search-based procedural generation of maze-like levels," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 260–273, 2011.

[7] K. Compton and M. Mateas, "Procedural level design for platform games." in *AIIDE*, 2006, pp. 109–111.

[8] M. Hadwiger, J. M. Kniss, K. Engel, C. Rezk-Salama, and H. Landis, "High-quality volume graphics on consumer pc hardware," in *IEEE Visualization*. Citeseer, 2002.

[9] J. Beneš, A. Wilkie, and J. Křivánek, "Procedural modelling of urban road networks," in *Computer Graphics Forum*, vol. 33, no. 6. Wiley Online Library, 2014, pp. 132–142.

[10] B. Weber, P. Müller, P. Wonka, and M. Gross, "Interactive geometric simulation of 4d cities," in *Computer Graphics Forum*, vol. 28, no. 2. Wiley Online Library, 2009, pp. 481–492.

[11] L. Krecklau, C. Manthei, and L. Kobbelt, "Procedural interpolation of historical city maps," in *Computer Graphics Forum*, vol. 31, no. 2pt3. Wiley Online Library, 2012, pp. 691–700.

[12] J. E. Moser, "The great depression," *A COMPANION TO WORLD WAR II*, p. 47, 2013.

[13] D. González-Medina, L. Rodríguez-Ruiz, and I. García-Varea, "Procedural city generation for robotic simulation," in *Robot 2015: Second Iberian Robotics Conference*. Springer, 2016, pp. 707–719.

[14] C. J. Headleand, G. Henshall, L. Ap Cenydd, and W. Teahan, "Randomised multiconnected environment generator," 2014.

[15] R. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra, "Integrating procedural generation and manual editing of virtual worlds," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010, p. 2.

[16] Y. Sun and J. Taplin, "Adapting principles of developmental biology and agent-based modelling for automated urban residential layout design," *Environment and Planning B: Urban Analytics and City Science*, p. 2399808317690156, 2017.

[17] Y. I. Parish and P. Müller, "Procedural modeling of cities," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001, pp. 301–308.

[18] C. A. Vanegas, D. G. Aliaga, B. Benes, and P. A. Waddell, "Interactive design of urban spaces using geometrical and behavioral modeling," in *ACM Transactions on Graphics (TOG)*, vol. 28, no. 5. ACM, 2009, p. 111.

[19] P. Waddell, "Urbansim: Modeling urban development for land use, transportation, and environmental planning," *Journal of the American planning association*, vol. 68, no. 3, pp. 297–314, 2002.

[20] G. Chen, G. Esch, P. Wonka, P. Müller, and E. Zhang, "Interactive procedural street modeling," in *ACM transactions on graphics (TOG)*, vol. 27, no. 3. ACM, 2008, p. 103.

[21] Y.-L. Yang, J. Wang, E. Vouga, and P. Wonka, "Urban pattern: Layout design by hierarchical domain splitting," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, p. 181, 2013.

[22] G. Nishida, I. Garcia-Dorado, and D. Aliaga, "Example-driven procedural urban roads," in *Computer Graphics Forum*. Wiley Online Library, 2015.

[23] T. Lechner, P. Ren, B. Watson, C. Brozefski, and U. Wilenski, "Procedural modeling of urban land use," in *ACM SIGGRAPH 2006 Research posters*. ACM, 2006, p. 135.

[24] T. Lechner, B. Watson, P. Ren, U. Wilensky, S. Tisue, and M. Felsen, "Procedural modeling of land use in cities," 2004.

[25] S. Groenewegen, R. M. Smelik, K. J. de Kraker, and R. Bidarra, "Procedural city layout generation based on urban land use models." in *Eurographics (Short Papers)*, 2009, pp. 45–48.

[26] S. Greuter, J. Parker, N. Stewart, and G. Leach, "Real-time procedural generation ofpseudo infinite'cities," in *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*. ACM, 2003, pp. 87–ff.

[27] ——, "Undiscovered worlds–towards a framework for real-time procedural world generation," in *Fifth International Digital Arts and Culture Conference, Melbourne, Australia*, vol. 5, 2003, p. 5.

[28] I. Garcia-Dorado, D. G. Aliaga, S. Bhalachandran, P. Schmid, and D. Niyogi, "Fast weather simulation for inverse procedural design of 3d urban models," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 2, p. 21, 2017.

[29] C.-H. Peng, Y.-L. Yang, F. Bao, D. Fink, D.-M. Yan, P. Wonka, and N. J. Mitra, "Computational network design from functional specifications," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 131, 2016.

[30] C. A. Vanegas, D. G. Aliaga, B. Benes, and P. Waddell, "Visualization of simulated urban spaces: Inferring parameterized generation of streets, parcels, and aerial imagery," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 3, pp. 424–435, 2009.

[31] K. R. Glass, C. Morkel, and S. D. Bangay, "Duplicating road patterns in south african informal settlements using procedural techniques," in *Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*. ACM, 2006, pp. 161–169.

[32] A. Emilien, A. Bernhardt, A. Peytavie, M.-P. Cani, and E. Galin, "Procedural generation of villages on arbitrary terrains," *The Visual Computer*, vol. 28, no. 6-8, pp. 809–818, 2012.