A semantic approach to patch-based procedural generation of urban road networks

Edward Teng
Delft University of Technology
edward.teng010@gmail.com

Rafael Bidarra
Delft University of Technology
r.bidarra@tudelft.nl

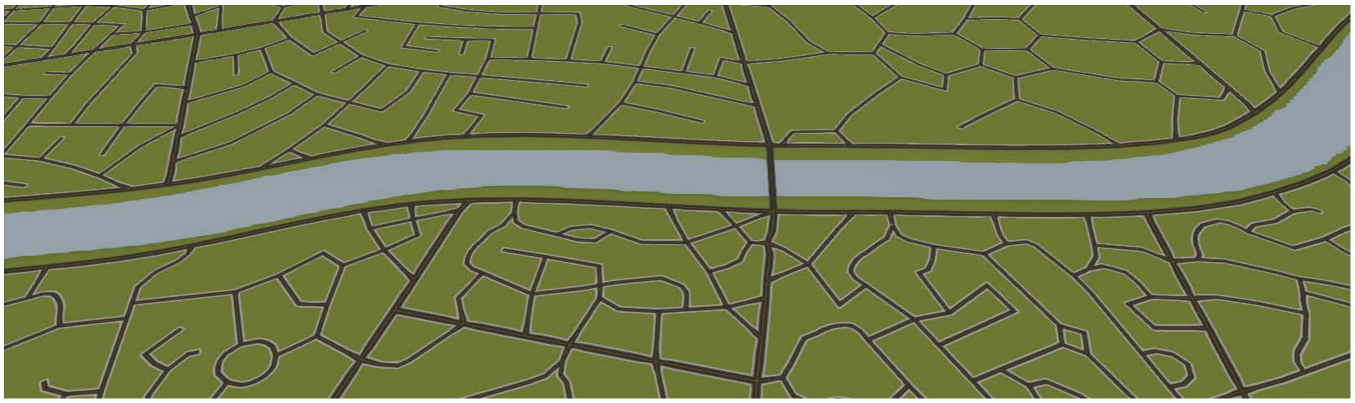

Figure 1: A road network generated using our parametric (upper half) and patch-based (lower half) methods.

ABSTRACT

A road network is one of the core elements of urban environments, strongly defining their layout. Procedural modeling has been increasingly used to create such road networks. However, many procedural methods are complex and difficult to master by non-experts, often have a limited and hard-to-control expressive range, and require a variety of specialized input data to generate a complex road network. To mitigate this, some methods proposed to use stochastic data on road patches extracted from example maps to design a road network following a given urban style. We propose a novel patch-based method that uses the semantics of individual patches to help guiding the procedural generation. Our approach combines the advantages of patch-based generation with those of conventional parametric methods. Due to the intuitive character of semantic parameters and tags, our approach provides for an easy customization of fictive road network creation, allowing a user to easily define various types of road network styles, containing only the desired features and structures of real-world road networks.

CCS CONCEPTS

• **Computing methodologies → Computer graphics**; *Shape modeling*;

KEYWORDS

Patch-based network generation, Urban modeling, Procedural content generation

ACM Reference format:
Edward Teng and Rafael Bidarra. 2017. A semantic approach to patch-based procedural generation of urban road networks. In *Proceedings of FDG'17, Hyannis, MA, USA, August 14-17, 2017,* 10 pages.
https://doi.org/10.1145/3102071.3110569

1 INTRODUCTION

Building small scale urban virtual environments by hand is a viable option, but for environments with a large amount of objects or structures, it is a tedious work. To assist artists in tackling this upscaling problem, procedural modeling techniques have been often utilized.

Several procedural modeling methods have been proposed to generate road networks, ranging from simple grid patterns to complex and realistic urban styles. However, procedural content generation can be difficult for users with no knowledge of modeling city layouts. In addition, the controllability of the generator is often poor and non-intuitive.

1.1 Related work

Researchers have proposed various procedural methods for modeling road networks, e.g. template-based (Sun et al. [16]), tensor-based (Chen et al. [2]) and agent-based (Lechner et al. [7], [6]). Kelly and McCabe [4] and Smelik et al. [14] discuss several procedural modeling methods, including their pros and cons.

Parametric and *example-based* methods are the most related to our approach. Parish and Müller [10] adapted the idea of L-systems (Prusinkiewicz and Lindenmayer [11]) to generate large cities, including their road networks. To achieve more realistic cities, typically a number of input maps and statistical data from the real world has to be given. Although parameters can be specified to modify the road network, it is not always clear nor intuitive how these parameters affect the network. Kelly and McCabe [5] proposed an approach where the user first creates the primary road network and a growth-based algorithm generates the secondary road network within the region(s) defined by the primary roads. For this, they provide a set of predefined generation parameters, e.g. grid, industrial and suburban, that facilitates creating road networks.

Aliaga et al. [1] introduced a parametric, stochastic example-based approach. Instead of having the user specify the generation parameters, this approach obtains the parameters, such as intersection type, street length and tortuosity, from input examples. Another example-based approach was introduced by Nishida et al. [8]. Instead of only using parameters, this approach mainly uses patches (interesting arterial and local road structures) extracted from example maps taken from OpenStreetMap [9]. This allows the user to create realistic detailed road networks containing complex structures. However, this approach uses the patch as-is, it does not allow individual patch rotation, deformation or scaling, thus lacking flexibility. Furthermore, even though the method uses some information on the patch, such as the type of streets (e.g. arterial or local), many more properties of a patch can be identified that are useful for the road generation process.

1.2 Contributions

We present a novel semantic approach that combines a patch-based method with the advantages of a parametric method for the customized generation of fictive road networks. The patch-based method enables the creation of road networks with complex structures, while the parametric method provides a convenient fall-back mechanism, and allows for the generation of more 'standard' road networks, e.g. grid and suburban styles. For each patch used, features are identified which embody some patch function or meaning, and are very useful for the controllability of the road generation and for its expressive power. We call this information the *patch semantics*. These features can be related to individual patch elements (e.g. a vertex or street segment) or to the patch as a whole.

With our approach, we solve two major challenges: seamlessly integrating patch-based and parametric methods in a controllable and intuitive way, and more importantly, representing the semantics of a patch and defining how this semantics can be utilized by a procedural generator. Hence, our contributions are the following:

- The use of patch semantics to help guiding a patch-based road network generator
- A controllable parametric road generation method with a high expressive power
- A high level settings scheme that allows non-experts to easily create and modify road networks

We implemented our approach in a fully functional prototype, that has been described elsewhere [17]; some of its features are also described in a short video, illustrating the flexibility of the interactive creation of various road networks, and a walk-through on its results. Our system allows for a very customized generation of large and complex road networks, that can be exported and loaded into any modeling system for further urban development, e.g. allotment and building generation. In order to quickly build a variety of 3D environments for our direct application domain, we used the procedural engine Sceelix [12], due to its very convenient node-based approach [13].

2 OVERVIEW

Road networks without crossovers or viaducts can be represented as a *planar geometric graph* in which each *vertex* can be seen as a point in a road network (such as an inflexion or an intersection), and each *edge* as a street (segment) connecting those vertices. Real-world road networks consist of (at least) main and local streets: *main streets* provide rapid displacement and access by connecting different areas, while *local streets* grow into (e.g. residential) local areas, providing access to the nearest main streets. In our geometric graph, main streets define a space partition into *main cells*; within each main cell, local streets (may) define *local cells*.

Our approach proposes the definition, representation and use of patch semantics and the integration of patch-based and parametric methods to generate road networks. The pipeline of this approach can be divided into two main phases: main- and local street generation, as depicted in Figure 2.

In the first phase, main streets are generated using a parametric graph growing algorithm, inspired on Kelly and McCabe [5]. The selected area is populated according to the available parameters: street length and its deviation range, minimum street length, vertex degree range, minimum street angle, angle deviation range and snap radius, see Figure 3. Presets can be defined to easily generate a specific type of a road network. Unlike Nishida et al. [8], we chose to use parametric instead of patch-based generation for main street generation, as it yields quite suitable cells and avoids a strong dependency on the particular set of main road patches used.

In the second phase, we propagate the local streets within each main cell. For this, a database of patches is first created and processed, identifying and classifying their patch semantics. Our patch-based method iteratively takes each (main) cell, initializes it and grows the local road network within it, as described in the following two paragraphs.

Cell initialization This process aims at creating a first set of *starting vertices* for the local street propagation within a cell. Many possible ways can be devised to achieve this; we have chosen to attach (suitable) patches to the main streets, and grow the network from there inwards, thus ensuring that the local streets being grown are connected to the main streets. Patch semantics, primarily their structure, determines which patches are suitable for this initialization. Requiring this specific patch structure typically ensures a natural transition from a main to a local street.

Network propagation The propagation process focuses on iteratively choosing a patch and appending it to (at least) one current starting vertex of the local street network being grown. Naturally, each new patch appended may bring in one or more connectable vertices to the pool. The patch selection takes into consideration both the network surroundings (e.g. neighboring vertices or edges,

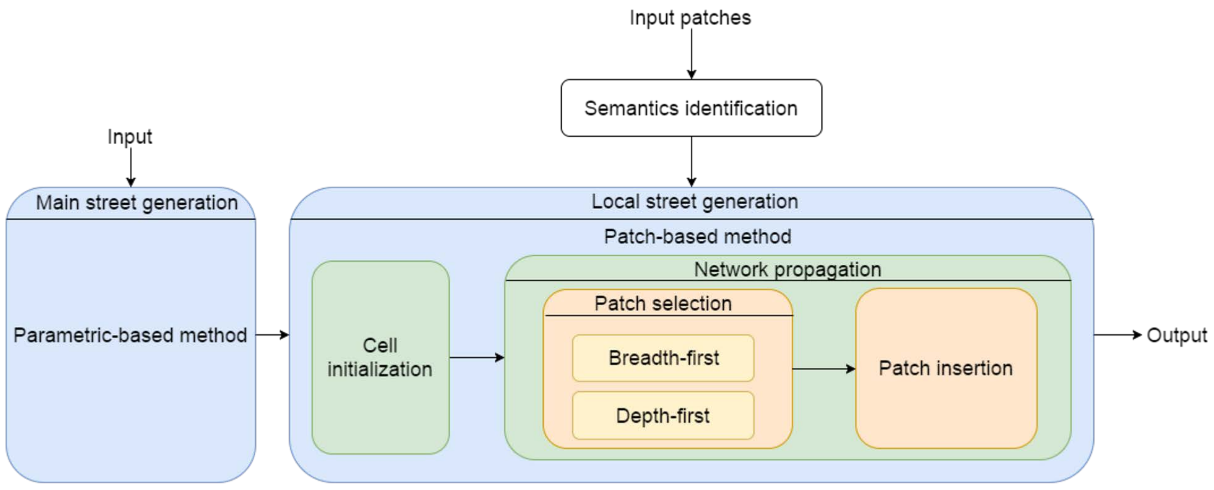

Figure 2: The general pipeline of our approach

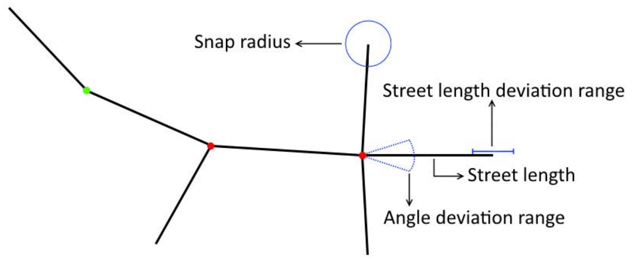

Figure 3: Diagram of generation parameters. The street length and its deviation range define the length of the road segment being created. The angle of the segment depends on the vertex degree, the angle of the adjacent edges and the angle deviation range. The degree of a vertex is constrained by the vertex degree range specified by the user. However, when there is no growth possible, a vertex may end up having a lower degree than the minimum (leaving e.g. a dead-end). In this illustration, a [2-4] degree range is used. The vertex in green has degree 2, and the vertices in red, degree 3 and 4 (also called intersection vertices). The snap radius is used for a snap operation, which 'shifts' an edge to end on a nearby vertex or edge.

available space) and each patch's semantics (e.g. size, connectivity). We consider two patch selection strategies when looking up a suitable patch: the *breadth-first* method, which typically propagates the network 'sideways' within the cell, using the new patch to bridge two current starting vertices; and the *depth-first* method, which typically propagates the network inwards into the cell, simply attaching a fitting patch to one starting vertex. Whenever no suitable patch can be found (e.g. no patch fits in the available space), the parametric growth is used instead.

3 PATCH-BASED METHOD

In this section we describe in more detail the main steps of our patch-based approach. The method requires a database of patches as input, in any suitable format. In our prototype system, patches for this database can be either manually created or extracted from example maps. For the former, we developed an interactive patch editor, that also supports the definition of *pattern elements*, i.e. an assembly of one or more patches that can be repeated in a predictable manner to form a road *pattern* (as e.g. those designed in many modern urban neighborhoods). For the latter, we implemented a patch extraction algorithm similar to that described by Nishida et al. [2015], and applied it to a variety of urban maps. For the purposes of this research, a patch database can consist of any combination of the above, no matter their origin.

3.1 Semantic identification

The semantics of a patch is classified into three categories: vertex, edge and patch. The first two categories describe the semantic information of individual vertices and edges of a patch, while the patch category describes the semantics of the patch as a whole. Patch semantics in our approach can be represented by tags on entities of each of these categories. Any patch property that is found useful for the road generation, or somehow meaningful for configuring the desired output, can be defined. In our current patch database, all the tags defined were automatically identified, based on geometric or topological analysis, as shown in Table 1. However, there is no objection to manually defining any additional tags on database patches. Figure 4 depicts some patch examples for various tags.

Each patch tag has its own purpose. The propagation process uses snap and clip operations wherever convenient and possible. A clip operation creates an intersection vertex at the intersection of two edges and clips the remaining edge segment. For patches, such operations are only applicable to snappable edges, i.e. edges that are connected to an intersection vertex. The main or local edge

Table 1: Patch semantic tags

Category	Tag	Definition
Vertex	Connectable	A vertex that allows connections
	Intersection	A vertex with at least 3 degree
Edge	Dead-end	An edge with no connectable vertices
	Snappable	An edge that allows snapping
	Main	An edge representing a main street
	Local	An edge representing a local street
Patch	Roundabout	A patch with a circular intersection
	Loop	A patch with one or more closed areas
	Cul-de-sac	A patch with one or more dead-end edges
	Curved	A patch with segments forming a curve
	Straight	A patch with only straight segments
	Terminal	A patch with only one connectable vertex
	Initial	A patch suitable for initializing the propagation process

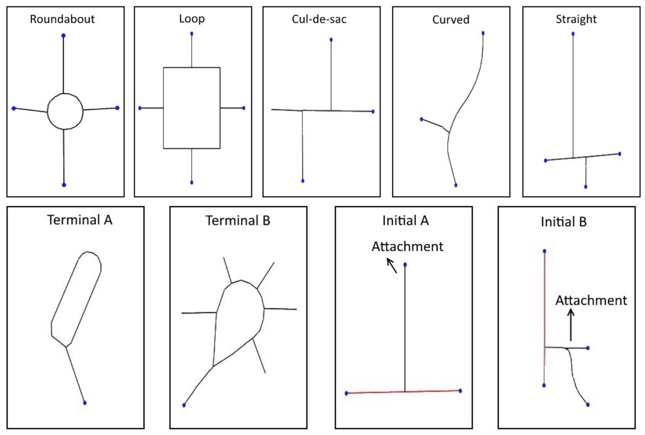

Figure 4: Examples of patches with different semantic tags. A blue dot represents a connectable vertex and a red line represents a main street.

property is used to indicate whether and how a patch can be used for initialization.

Some patch tags indicate geometric or topological properties, as e.g. Roundabout, Loop, Cul-de-sac, Curved and Straight. In order to control the patch-based generation, patches with such tags can be filtered out. For example, a Terminal patch, by definition, prevents any further propagation as it adds no connectable vertices to the pool. Thus, in general, it is only applied when there is a limited area left. As another example, an Initial patch consists of a straight main street edge and an attachment, i.e. a connected set of local street type edges with at least one connectable vertex. In this way, it can be deployed during the initialization phase of a main cell (and, of course, also during network propagation).

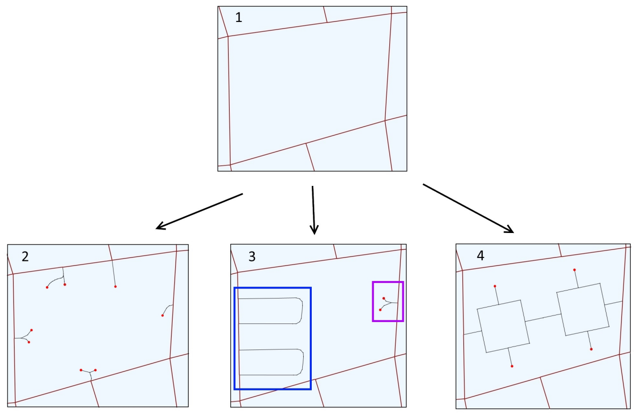

Figure 5: Different methods to initialize a cell. (1) The selected main cell. (2) Initialization using patches on main streets. The red dots represent connectable vertices. (3) Initialization using a pattern on a main street. Notice that the pattern (in blue rectangle) does not have a connectable vertex, so an additional patch (in purple rectangle) is added to supply connectable vertices. (4) Initialization using a pattern between main streets.

3.2 Cell initialization

In this process, initial patches are inserted onto a main cell, in order to produce the first set of starting vertices to be used in the propagation process. Among the various possible methods to achieve this, we implemented the following three, available for the designer to choose from (see also Figure 5).

Patches on main streets This method places patches with Initial tag on main streets of the cell, see Figure 5.2. *Seed points*, i.e. main street positions to be used for patch placement, are calculated so that patches are properly distributed over the main cell perimeter. For every seed point, an Initial patch is randomly chosen, after which it is translated to the seed point, rotated such that the main streets (of patch and seed point) are aligned.

Pattern on a main street This method creates and adds a pattern on a main street, see Figure 5.3. To form a pattern, a pattern element has a scale and a multiplier. The scale indicates the scaling of the element and the multiplier determines how many times the scaled element is replicated. To add the pattern to the graph, the element is scaled and repeated according to mentioned values, and then rotated and translated such that the pattern lies on the desired main street. If the pattern does not contain any connectable vertex, a main street that lies far from the pattern is initialized with an initial patch, in order to increase the space and chances of propagation in that cell.

Pattern between main streets Another method is to place a pattern across two main streets, see Figure 5.4. The seed points for the start and end vertices of the pattern are placed in the middle of each main street pair to minimize unnecessary obstacles for the pattern placement. The procedure to find a base pattern is fairly similar to the previous method, except that the line connecting the seed points is used instead of a main street.

3.3 Network propagation

The propagation process populates a main cell with local streets, by iteratively attaching a patch to the vertex currently being used for the propagation. For this, we maintain a pool of candidate starting vertices in a queue, which was first populated during cell initialization, and is subsequently grown with the connectable vertices of each new patch appended to the network; see Algorithm 1. First, the next *current vertex* is popped from the queue, and a check is done on the feasibility of attaching a patch to it. This check takes into account possible constraints on the current vertex degree as well as on the actual space available in the computed propagation direction. Next, after filtering out patches that would not fit the available space, two patch selection methods are successively attempted: (i) breadth-first, that checks whether there is a patch that can connect the current vertex to a nearby vertex; and (ii) depth-first, that searches for a patch that can be simply appended to the current vertex of the network. For both methods holds that only valid patches, i.e. complying with the user-defined parameters and constraints, can be used and appended to the network. If none of the methods finds a suitable patch for the current vertex, parametric growth is used instead, as a fall-back method.

Algorithm 1 Network propagation

```
1:  Input: graph, patches, cell
2:  Output: graph
3:  while graph.candidateVertices.Count >0 do
4:      currentVertex ← graph.candidateVertices.pop()
5:      propagDirection ← calculate the propagation
            direction for currentVertex
6:      if propagDirection is valid then
7:          filteredPatches ← filter available patches
8:          if filteredPatches.isEmpty() then
9:              apply parametric growth on currentVertex
10:         else
11:             suitablePatch ← apply Breadth-First to find a
                    suitable patch                    ▷ Algorithm 2
12:             if suitablePatch == null then
13:                 suitablePatch ← apply Depth-First to find a
                        suitable patch                ▷ Algorithm 3
14:             if suitablePatch != null then
15:                 attach suitablePatch on currentVertex
16:             else
17:                 apply parameteric-based growth on currentVertex
```

Propagation direction The propagation direction is an indication of which approximate direction the new patch could take, if attached to the current vertex (the actual direction may deviate within a given range, similar to the parametric generation). This direction is given by the angle bisector of the largest angle formed by the adjacent edges of the current vertex. As can be seen in Figure 6, the propagation direction depends on the degree of the current vertex.

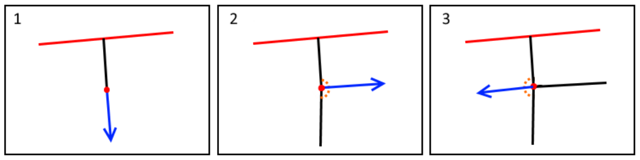

Figure 6: Determination of the propagation direction (blue arrow) for a current vertex (in red) of different degrees: (1) degree 1, (2) degree 2, and (3) degree 3.

Patch filtering This process filters available patches so that only patches that fit the work area within the cell are eligible during the patch selection process. The work area available for propagation, which is typically substantial at the early stages, becomes smaller as the propagation progresses cell inwards. As a result, more and more large patches will not fit in that space, and can better be excluded at the selection stage, for the sake of efficiency. We use an approximation of the work area instead of the actual work area, because it is computationally less expensive and provides a satisfactory estimation; see Teng [17] for details.

Breadth-first method The main purpose of this method is to find a suitable patch that can connect the current vertex with one of the remaining connectable vertices in the graph pool; see Algorithm 2. Basically, the method has to search for two pairs of vertices that are at (approximately) the same distance: (i) a *graph vertex pair*, consisting of the current vertex (start) and another connectable vertex (target) of the graph; and (ii) a *patch vertex pair*, consisting of two connectable vertices (start and target) of the candidate patch. For efficiency, patch vertex pair distances can be precomputed and stored with the patch. All suitable solutions found (patch, patch vertex pair, and graph vertex pair) are collected and rated. The rating system indicates the suitability of the solution for the situation at hand. To preserve the geometry of the patch as much as possible, the rating depends on the difference between the patch vertex pair distance and the graph vertex pair distance, and is given by

$$1 - \frac{|distancePatchPair - distanceGraphPair|}{distanceGraphPair} \tag{1}$$

with *distancePatchPair* as the patch vertex pair distance and *distanceGraphPair* graph vertex pair distance. Ultimately, the highest rated solution, if any, is chosen (patch with corresponding vertex pairs). The patch is then appended to the graph, possibly after the necessary rotation and snapping adjustments, because in most cases, the patch vertex pair distance will not fit exactly the graph vertex pair distance. Several methods can be used for this, e.g. (i) a patch snappable edge (if any) is extended or shortened so that the patch fits perfectly between the two vertices, and (ii) compute the patch rotation by which the distance between the two target vertices is the shortest and let the snap operation merge them.

Depth-first method The main purpose of this method is to find a suitable patch that can be appended at the current vertex, to

Algorithm 2 Breadth-First

```
1: Input: graph, filteredPatches, currentVertex
2: Output: patch or null
3: suitablePatches ← ∅
4: validConnectableVertices ← select connectable graph
        vertices of degree 1 that are in 'line-of-sight'
        with currentVertex
5: for each vertex in validConnectableVertices do
6:     distanceGraphPair ← distance(currentVertex, vertex)
7:     for each patch in filteredPatches do
8:         pairs ← find connectable vertex pairs in patch
               at approximately distanceGraphPair of
               each other (i.e. within the snap radius)
9:         for each pair in pairs do
10:            rotation ← compute patch rotationsuch that
                   the connection vertices in pair approx.
                   coincide with currentVertex and vertex
11:            orient patch according to rotation
12:            if patch is valid then
13:                rating ← calculate the rating for patch
                       and its pair                    ▷ Equation 1
14:                add rating, pair and patch to suitablePatches
15: if suitablePatches.Count() >0 then
16:     return the highest rated patch
17: else
18:     return null
```

grow the network in the propagation direction (again, possibly with some deviation); see Algorithm 3. Because here, unlike the breadth-first method, only one connectable vertex of the patch is used to append it to the current vertex, we calculate one rating for each of its connectable vertices. Per patch, the base rating for each connectable vertex is initialized at 1 and, subsequently, the rating mechanism adjusts it according to a number of heuristics, as follows:

- For every other connectable vertex of the patch nearby a dead-end edge of the graph decreases the rating by 0.2;
- For every edge of the patch nearby another connectable vertex of the graph decreases the rating by 0.2;
- For every dead-end edge of the patch that is not nearby an edge of the graph decreases the rating by 0.2 otherwise increases the rating by 0.1.

The first two conditions reduce the rating of that particular patch attachment, as it could obstruct subsequent propagation. The last condition reduces the rating to avoid the use of Cul-de-sac patches in an open area where they could block further propagation. However, it increases the rating to promote the use of Cul-de-sac patches where there is a limited area. The start rating, increment/decrement values are relative due to every patch being handled similarly. However, it is important to note that the rating is penalized more heavily, e.g. higher decrease value than the increase value, to avoid situations where propagation can be hindered.

Eventually, the patch (if any) with the connectable vertex having the highest rating is selected as the most suitable.

Algorithm 3 Depth-First

```
1: Input: graph, filteredPatches, currentVertex,
        propagDirection
2: Output: patch or null
3: suitablePatches ← ∅
4: for each patch in filteredPatches do
5:     for each connectableVertex in patch.connectableVertices do
6:         attachDirection ← small variation on propagDirection
7:         orient patch so that the adjacent edge to
               connectableVertex is aligned with attachDirection
8:         if patch is valid then
9:             rating ← calculate the rating for patch and its
                   connectableVertex
10:            add rating, connectableVertex and patch to
                   suitablePatches
11: if suitablePatches.Count() >0 then
12:     return the highest rated patch
13: else
14:     return null
```

Constraints For the generation of plausible road networks, it is important that every parametric road segment, as well as every patch selected by the patch-based method, smoothly fits within the network being generated. For the sake of plausibility, a variety of customizable parameters and thresholds have been defined, e.g. snap radius, minimum street angle, minimum street length and vertex degree range. We defined the following four constraints on a segment (or patch) that help enforce segment validity and increase its overall plausibility:

(1) a proposed road segment must connect to a connectable vertex or edge that lies within the snap radius of the end of the road segment;
(2) the angle between two road segments sharing a vertex must be higher than the minimum street angle;
(3) the length of a road segment must be longer than the minimum street length; and
(4) the vertex degree may never exceed the maximum degree of the specified range.

To abide by these constraints, the following *connect* and *clip* operations checks are defined, and applied whenever needed:

- **Connect** the road segment to the nearest vertex or create an intersection to the nearest edge within the snap radius of the target vertex of the proposed road segment. However, discard the road segment whenever: (i) the angle between the proposed and existing edges is lower than the minimum street angle; (ii) the length of any of the split edges is shorter than the minimum street length; or (iii) the target vertex has the maximum degree allowed.
- **Clip** the road segment when intersecting another edge and insert a vertex on the intersecting point. However, like for the connect operation, discard the proposed road segment if (i) or (ii) hold.

4 RESULTS AND ANALYSIS

In this section we present and discuss a variety of results achieved by our patch-based method, and we analyze the controllability of the parametric generator in order to discuss the expressive range of our approach.

4.1 Results of the patch-based generator

To validate the results of the patch-based generator, we have built a database of 98 patches, most of them extracted from real-world data and some from manual creation. For the parametric method, the following parameters are used: street length (50m), street length deviation range (10m), minimum street length (20m), vertex degree range [4-4], minimum street angle (70°), angle deviation range (10%) and snap radius (20m). Figure 7 shows a road network using all available patches. By allowing designers to exclude any undesired patch tags during the generation process, various other road network with different properties and appearance can be generated, as shown in Figure 8.

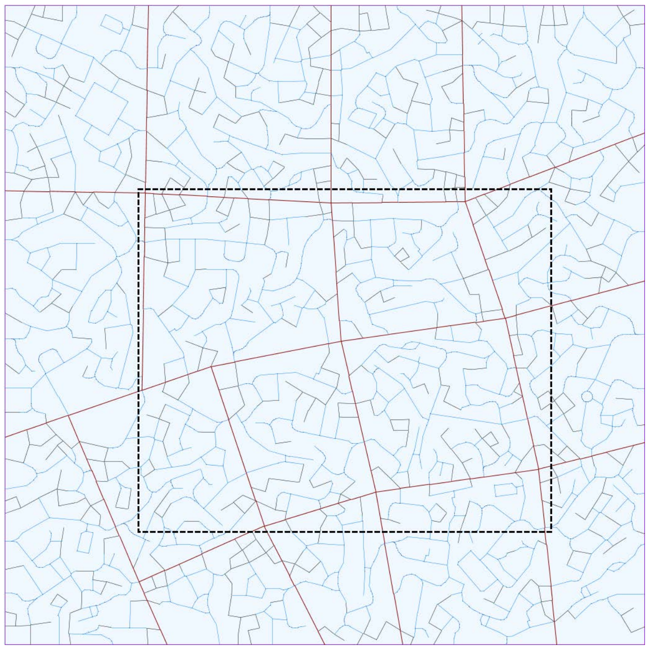
Figure 7: A road network with an area of 4 km^2 generated using the complete patch database. Main streets (red lines) were generated using the parametric method; local streets were generated using both patches (blue lines) and, when needed, the parametric method (black lines).

Table 2 presents generation data about the road networks in Figures 7 and 8. There are three noteworthy aspects arising from this data:

(1) *The differences in the generation time.*
The generation of a road network using a small patch collection is faster than using a large one: the larger the collection of patches available, the more patches need to be evaluated during the selection process.

(2) *The amount of unique patches used is significantly lower than the total amount of available patches.*
This can be partly due to 'special case' patches, such as terminal patches, which are avoided in certain situations. In addition, some patches, e.g. T and × (simple) structures, are more likely to be chosen than patches with somewhat more complex structures. Therefore, we provided a convenient way of controlling the output of a patch-based road network by setting a limit on the frequency of occurrence of a patch. In this way, limiting e.g. each patch to occur only once, increases the amount of unique patches, although typically decreasing the total amount of patches used.

(3) *The total amount of patches used is comparable for all networks.*
The second road network (excluding curved patches) uses a considerably small amount of unique patches, yet it uses a total amount of patches comparable to the others. We have seen that

Table 2: Generation data on the road networks in Figures 7 and 8

Tags	Number of patches				
Excluded	Available	Unique Used	Used	Time (s)	Figure
None	98	58	381	25	7
Curved	39	29	354	13	8.a
Straight	59	38	335	16	8.b
Loop	75	44	362	21	8.c
Cul-de-sac	84	52	345	22	8.d

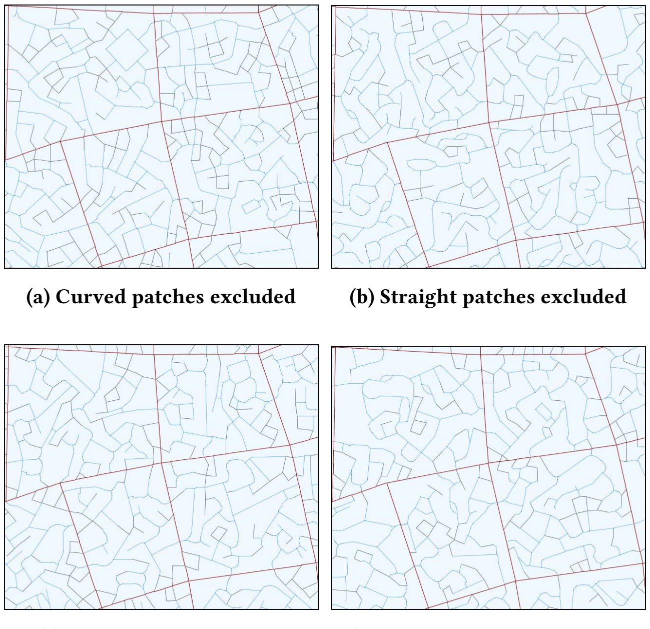
(a) Curved patches excluded (b) Straight patches excluded
(c) Loop patches excluded (d) Cul-de-sac patches excluded

Figure 8: Different road networks for the highlighted area in Figure 7, generated excluding patches with a specific tag.

in such cases, even though using less unique patches, there are often a few among them that can fit easily in many situations.

Performance The generation times for the patch-based method are also dependent on the process of filtering out 'non-fitting' patches, for which we compute a work area, as described in Section 3.3. This means that, for the same area covered, it takes longer to generate a road network with a few large cells than a road network with many smaller cells. As an example, for an area of 4 km^2, generating a patch-based road network with our prototype system takes, on average, 40 seconds if it is divided into 4 main cells, and 23 seconds, if it is divided into 16 main cells.

Limitations Presently, there are two major limitations in our patch-based generation approach. First, it is not possible to explicitly place a given patch at a certain location, nor to assure that a patch is used at least once. Second, despite the selection methods and rating mechanisms to find a suitable patch, it still sometimes happens that quite large and complex local cells are generated.

4.2 Analysis of the parametric generator

One way to measure the power and controllability of procedural generation methods is to analyze their *expressive range*. The expressive range of a method indicates the variety of its output content, obtained by acting on some control mechanism available, e.g. generation parameters. Similarly to Smith and Whitehead [15], we analyze the expressive range of our approach by following four steps: determine the comparison metrics, generate content, visualize the expressive range and analyze the impact of parameters. For our approach, we soon found out that any attempt at measuring and analyzing the expressive range of the patch-based method would be first and foremost dependent on the actual database of patches, more than on any actual controllable settings. Therefore, we decided to focus the analysis on our parametric generator.

Comparison metrics It is important that the chosen metrics measure relevant features of road networks. We have chosen *street connectivity* and *street density*, as these two metrics are independent, have an intuitive connotation and can be precisely defined. For example, a road network with high connectivity means that it is typically 'straightforward' to go from A to B, while a dense road network means that it contains a high amount of streets within an area. The street density of a road network, which can be visually perceived on a map, is defined as the total street length (in km) per area unit (in km^2). As for the connectivity, we looked into the various street connectivity metrics described by Dill [3]. For our study, we chose the Connected Node Ratio (CNR) as our metric, because it is based on the notion of dead-ends, which have an intuitive connotation, and it has a low computational cost:

$$CNR = \frac{\#intersections}{\#intersections + \#deadends} \tag{2}$$

A high CNR value indicates a relatively low amount of dead-ends compared to the amount of intersections and, thus, a higher level of connectivity.

Content generation Our parametric generator provides numerous parameters to control the output road network; see Figure 3 for the most important. For this analysis, we have chosen to generate road networks by varying the *minimum street angle* and the *vertex degree range* because these two parameters have a strong impact on the output. We vary the minimum street angle from 50° to 90° with increments of 10° (an angle lower than 50° would be too acute, and higher than 90°, too restrictive). For the vertex degree range, we start at a [2-3] range, a go up to a [4-5] range. For the whole analysis, all other generator parameters were kept fixed with the following values: street length (50m), street length deviation range (10m), minimum street length (20m), street angle deviation range (10%) and snap radius (20m).

Output visualization Figure 9 depicts the expressive range data collected, using hexagonal bin plots. For each of the plots, we generated 1000 road networks covering an area of 1 km^2. Each hexagonal bin indicates the amount of networks with fairly similar connectivity and density scores. Due to space limitations, we left out the plots for some intermediate degree ranges; their results are consistent with these, and can be found elsewhere [17].

Impact of generation parameters From the analysis of the expressive range data in Figure 9, we can make a number of observations about the impact of the parameters used on the generated road networks:

Focusing on each column, thus with a fixed minimum street angle, we notice that:

- The higher the vertex degree range, usually the denser the graph, while the connectivity stays relatively steady.
- There is a drop of density from vertex degree range [2-5] to [3-3]. The reason for this is that the range [2-5] allows up to 5 edge connections for each vertex, and the range [3-3], only up to 3; as a result, in general, more edges can be generated using the former.
- The last column (for a minimum street angle of 90°) shows a rather steady density, in contrast to the other four columns. The reason is that this minimum street angle in practice restricts the maximum degree of a vertex to 4, occurring only when the angle between two edges is exactly 90°(which does not happen very often). This is not the case for the other columns: with lower minimum street angles, there is more flexibility, and thus a wider margin for street density variations.

Focusing on each row, thus with a fixed vertex degree range, we notice that:

- The higher the minimum street angle, the lower the connectivity of the graph and usually also the lower the density.
- The first row has relatively steady (low) density scores, as the connectivity scores decrease. The reason for this is that, due to the already low vertex degree range [2-3], these road networks are already quite sparse, so that a higher minimum street angle only causes less snapping, and hence more dead-ends.

From the observations above, we can conclude that the minimum street angle and vertex degree range have indeed a considerable impact on both street connectivity and street density. Even more, we can actually determine which approximate value ranges these two

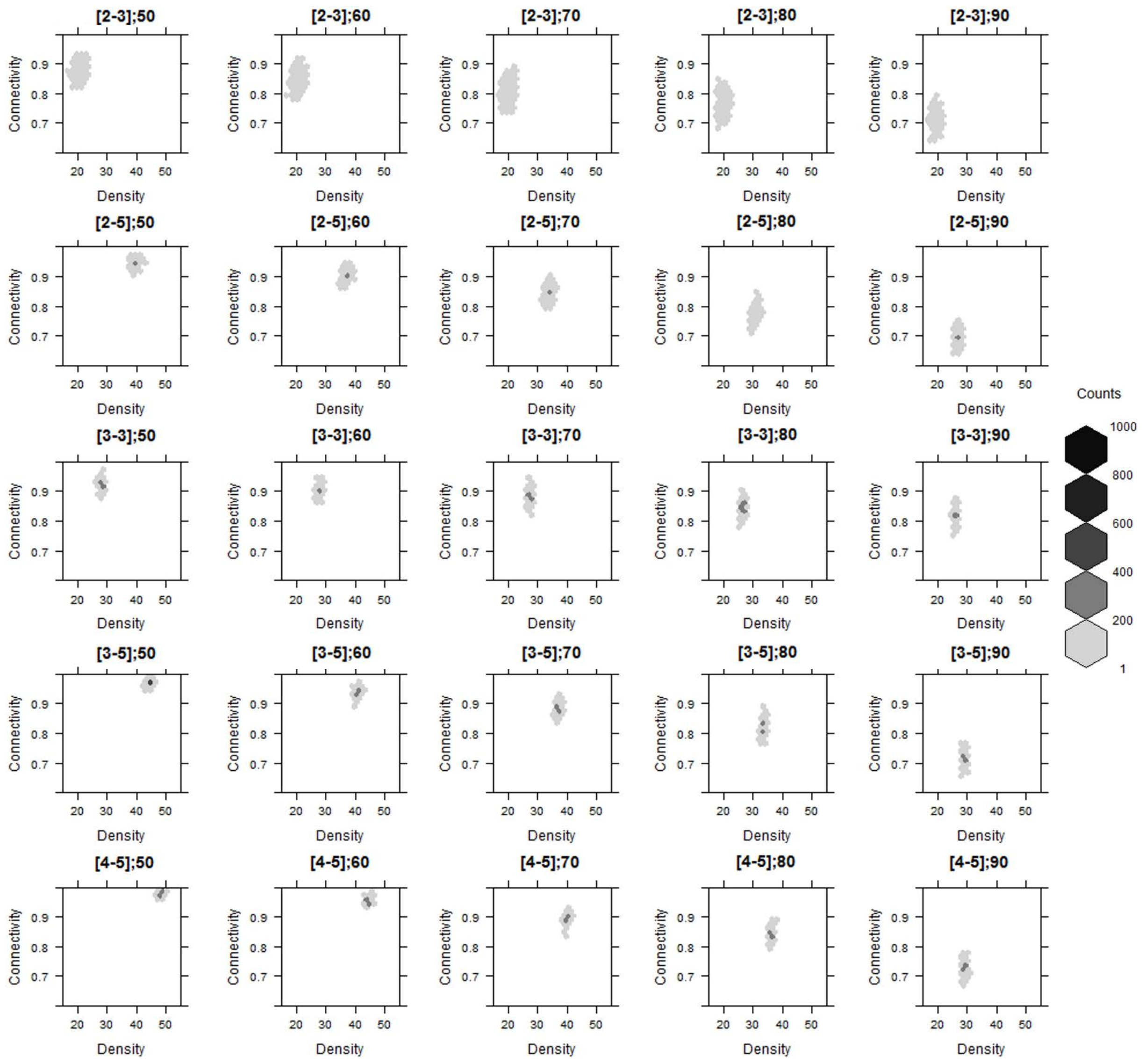

Figure 9: Expressive range of the parametric generator, varying the vertex degree range (vertically) and the minimum street angle (horizontally). For each plot, the connectivity (y-axis) is measured by the Connected Node Ratio, see Equation (2), and the density (x-axis) is the total street length per km^2. Plots on the same row share the vertex degree range, but use increasing minimum street angles, while plots on the same column share the minimum street angle, but use different vertex degree ranges. All generated road networks have a connectivity ranging between 0.6 and 1, and a density ranging between 15 and 55.

parameters require, in order to generate road networks with specific connectivity and density. A road network with high connectivity and density can be achieved by specifying a high vertex degree range and a low minimum street angle; a road network with high connectivity and low density can be generated using a low vertex degree range and a low minimum street angle; finally, defining a low vertex degree range and a high minimum street angle will create a road network with low connectivity and low density.

As can be seen from the above analysis, as well as from the combined plot of Figure 10, altogether the output of the generator, obtained by varying only the minimum street angle and vertex degree range, covers a very large area of the space defined by the two metrics. However, one can notice that the bottom right area is missing. Network graphs falling in that area have a low connectivity and a high density. In terms of road networks, having a relatively high number of dead-ends and a high street density seems quite non plausible, as it would mean having e.g. many unconnected streets twisted very close to each other. We managed to generate such networks by forcing unlikely low snap radius and high minimum street length, but the output is always a rather improbable and non-realistic road network. In other words, the 'reluctance' of the generator to create such networks is an intended consequence of the

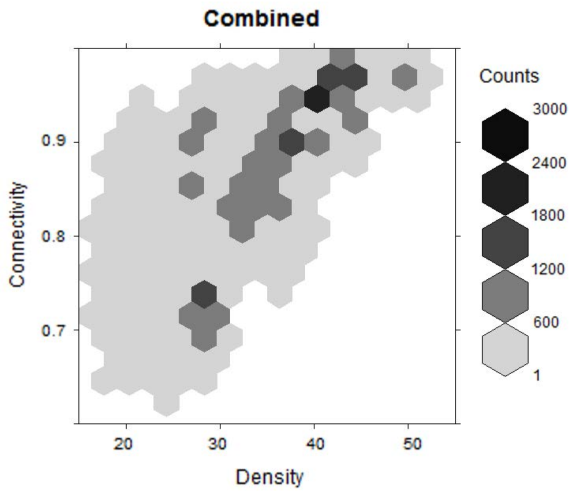

Figure 10: The combined bin plot for all data used in the expressive range study of Figure 9.

design goal of promoting the generation of plausible road networks only.

In conclusion, based on these two metrics, we can say that the expressive range of our parametric road network generator is quite large, and that the features of its output can be easily controlled by parameters such as minimum street angle and vertex degree range.

5 CONCLUSION

We presented a novel semantic approach to generate fictive road networks with plausible complex structures, which combines the advantages of a patch-based method with those of a parametric method. For our patch-based method, the semantics of each individual patch is represented and stored in a patch database. The process of first trying to propagate the road network by appending patches using the patch-based method, and employing a parametric method as a fall-back mechanism when no suitable patch can be found, combines both methods into an integrated solution.

Our patch-based generation is controllable by selecting the desired patch semantics, e.g. by excluding one or multiple patch tags, and also by limiting the number of occurrences a patch can be used. Our parametric generator is configurable using various parameters, which allow for the generation of a large variety of road network styles. The generator has been shown to have a wide expressive range. In order to facilitate its use by non-experts, one can define any number of presets that can be used to easily generate different types of road networks. From our tests, we have confirmed that users without any technical knowledge of procedural techniques were able to create very specific road networks on demand [17].

The first direct application for our approach was to assist neuropsychologists with the creation of virtual urban environments with customized walking paths. With these, they can better diagnose patients with a variety of orientation disorders by exposing them to fully configurable and safe virtual environments created in-house to fit their unique situation. We envision numerous applications, especially aimed at enabling non-technical users to generate customized urban road networks that fulfill very specific requirements and preferences, possibly induced by their unique target user(s). Examples of these are driving simulators, courier delivery training systems, urban adventure games, etc.

There are a few promising directions for future work. First, in addition to using patch patterns for initializing a main cell, it would be interesting to explore how they could also be used during the propagation process. Second, the breadth-first method currently only seeks one connectable vertex when searching for a suitable patch. However, because some patches have multiple connectable vertices, it might be worth exploring how one could seek and select a patch that connects multiple connectable vertices. Third, our choice for a planar geometric graph representation is somewhat restrictive of the output road networks. It would be interesting to explore how the use of non-planar patches could solve this limitation.

REFERENCES

[1] Daniel G Aliaga, Carlos A Vanegas, and Bedřich Beneš. 2008. Interactive example-based urban layout synthesis. *ACM Transactions on Graphics* 27, 5 (2008), 160.
[2] Guoning Chen, Gregory Esch, Peter Wonka, Pascal Müller, and Eugene Zhang. 2008. Interactive procedural street modeling. *ACM Transactions on Graphics (TOG)* 27, 3 (2008), 103.
[3] Jennifer Dill. 2004. Measuring network connectivity for bicycling and walking. In *83rd Annual Meeting of the Transportation Research Board, Washington, DC.* 11–15.
[4] George Kelly and Hugh McCabe. 2006. A survey of procedural techniques for city generation. *Institute of Technology Blanchardstown Journal* 14 (2006), 87–130.
[5] George Kelly and Hugh McCabe. 2007. Citygen: An interactive system for procedural city generation. In *Fifth International Conference on Game Design and Technology.* 8–16.
[6] Thomas Lechner, Pin Ren, Ben Watson, Craig Brozefski, and Uri Wilenski. 2006. Procedural modeling of urban land use. In *ACM SIGGRAPH 2006 Research posters.* ACM, 135.
[7] Thomas Lechner, Ben Watson, and Uri Wilensky. 2003. Procedural city modeling. In *1st Midwestern Graphics Conference.*
[8] G. Nishida, I. Garcia-Dorado, and D. G. Aliaga. 2016. Example-Driven Procedural Urban Roads. *Comput. Graph. Forum* 35, 6 (Sept. 2016), 5–17. https://doi.org/10.1111/cgf.12728
[9] OpenStreetMap. 2017. OpenStreetMap. (2017). http://www.openstreetmap.org/ visited on 2017-04-20.
[10] Yoav IH Parish and Pascal Müller. 2001. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques.* ACM, 301–308.
[11] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. 1991. The algorithmic beauty of plants (the virtual laboratory). (1991).
[12] Sceelix. 2017. Sceelix. (2017). https://www.sceelix.com/ visited on 2016-12-05.
[13] Pedro Silva, Elmar Eisemann, Rafael Bidarra, and Antonio Coelho. 2015. Procedural content graphs for urban modeling. *International Journal of Computer Games Technology* 2015, 808904 (jun 2015). http://www.hindawi.com/journals/ijcgt/2015/808904/
[14] Ruben M Smelik, Tim Tutenel, Rafael Bidarra, and Bedřich Beneš. 2014. A survey on procedural modelling for virtual worlds. *Computer Graphics Forum* 33, 6 (2014), 31–50.
[15] Gillian Smith and Jim Whitehead. 2010. Analyzing the expressive range of a level generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games.* ACM.
[16] Jing Sun, Xiaobo Yu, George Baciu, and Mark Green. 2002. Template-based generation of road networks for virtual city modeling. In *Proceedings of the ACM symposium on Virtual reality software and technology.* ACM, 33–40.
[17] E. Teng. 2017. *A semantic approach to patch-based procedural generation of urban road networks.* Master's thesis. Delft University of Technology. https://graphics.tudelft.nl/edward-teng