EUROGRAPHICS 2012 / P. Cignoni, T. Ertl
(Guest Editors)

Volume 31 (2012), Number 2

Procedural Interpolation of Historical City Maps

Lars Krecklau, Christopher Manthei, Leif Kobbelt

RWTH Aachen University, Germany

Abstract

We propose a novel approach for the temporal interpolation of city maps. The input to our algorithm is a sparse set of historical city maps plus optional additional knowledge about construction or destruction events. The output is a fast forward animation of the city map development where roads and buildings are constructed and destroyed over time in order to match the sparse historical facts and to look plausible where no precise facts are available. A smooth transition between any real-world data could be interesting for educational purposes, because our system conveys an intuition of the city development. The insertion of data, like when and where a certain building or road existed, is efficiently performed by an intuitive graphical user interface. Our system collects all this information into a global dependency graph of events. By propagating time intervals through the dependency graph we can automatically derive the earliest and latest possible date for each event which are guaranteeing temporal as well as geographical consistency (e.g. buildings can only appear along roads that have been constructed before). During the simulation of the city development, events are scheduled according to a score function that rates the plausibility of the development (e.g. cities grow along major roads). Finally, the events are properly distributed over time to control the dynamics of the city development. Based on the city map animation we create a procedural city model in order to render a 3D animation of the city development over decades.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation I.3.4 [Computer Graphics]: Graphics Utilities—Paint Systems

1. Introduction

Creating and visualizing 3D models of real cities has become an important topic in a lot of different areas such as tourist information applications, city planning, or historical research. Unfortunately, most reconstruction methods only focus on a static model at a fixed point in time like the Rome Reborn project [Fri08]. Compared to the reconstruction of a very small set of fixed time points, an animation is a much more convenient way for the illustration of a historical city development, because it directly provides the viewer an intuition of how the urban development might have taken place. In this paper we propose an approach that produces smooth and realistic animations. This is an interesting aspect, especially for educational purposes, because real world data is interpolated in order to show historical facts in an integrated and consistent style communicating the big picture of a city development. Furthermore, our technique could be utilized in the games or films industry as an innovative transition effect showing a time-lapse animation of the city development as the story progresses.

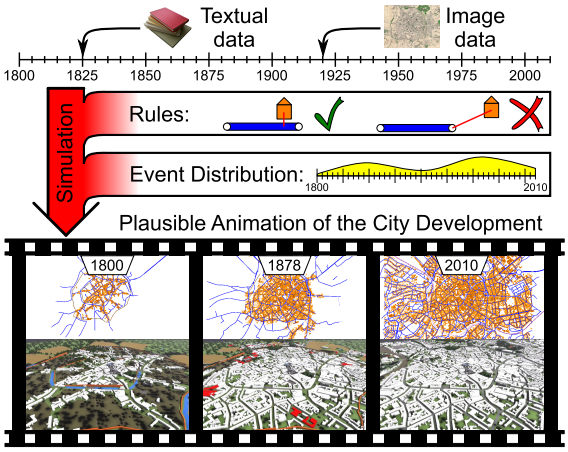

Figure 1: *After defining a sparse set of events (e.g. taken from historical maps), we create a dependency graph from a rule set (e.g. houses are build close to streets). Based on a proper event distribution we then apply a procedural interpolation resulting in a plausible animation of the city development. The bottom image sequence shows the development of Aachen, Germany over 210 years.*

If the 4th dimension, i.e., time, is taken into account, several challenging problems have to be solved in order to generate a consistent animation. Most importantly, we can not simply extrapolate a complex mathematical model as it is done in previous work [WMWG09], since we are not looking for a *possible* outcome of the unknown city development in the future. Instead, we have to find a plausible simulation of the city development that has to stay *consistent* with the given historical data from the past. Therefore, our paper focusses on a technique to properly interpolate the given data.

Typically, the available data of historical events is relatively sparse and sometimes it is even very imprecise. Especially the city maps of previous centuries are rather inaccurate, because the people during that time did not have any satellite images or precise measuring devices. If those maps are just blended over time, the result is disappointing, since the street graphs of different time points will not perfectly match due to the distortion that is caused by the imprecise measuring. We tackle this problem by rectifying images of historic city maps with respect to the current state of the city map, which is assumed to be precise. Taking the current city map as a ground truth, a user just has to select objects that did not exist in the past or insert new objects that might have been destroyed meanwhile.

Usually, a large part of the data is available in a textual form such as "this building was built in 1875". Sometimes the information is only provided in an even more fuzzy way like "this building was built sometimes in between 1975 and 1990". By providing an intuitive painting metaphor, this information can be easily inserted into our system and is thereby converted to a visual interpretation.

For the actual simulation of a possible city map development, we have to take into account that any *events*, like building a street or destroying a house, are based on a set of logical rules. Most importantly, we observe interdependencies between several events, e.g., a street will only be built if it is connected to the rest of the street network. Furthermore, the importance of events might differ which can be expressed by a scoring function, e.g., constructing objects that are closer to the city center will be preferred in order to produce a more realistic development of the city boundary. Our system provides a flexible set of procedures that heuristically derive such *dependency relations* or *scores*. Although the logical rules would lead to a locally well defined behavior of the city development, we still need to control the global appearance of the animation. Our approach uses a *global distribution function* as a reference to decide how many events are started at a certain point in time while ensuring that the animation still satisfies any inserted information in order to generate an interpolation of the real world data. We further enhance the simulation by a simple interpolation technique for any underlying land use maps such as a visualization layer for the vegetation or an economical distribution map of industrial versus residential areas.

For an efficient and intuitive creation of a realistic city map development animation, our paper mainly focuses on the following challenges:

Procedural Interpolation Considering a graph of interdependencies between events, a scoring function for the importance of an event and a histogram to control the global distribution of events over time, our algorithm produces a plausible simulation of the city map development that properly interpolates any inserted real world data such as historical maps or fuzzy textual information. Furthermore, maps containing meta information are used to influence the rules on a high level of abstraction.

Interaction We provide an intuitive user interface that enables the insertion of potentially thousands of event constraints within seconds. We support the user by allowing the use of images of historical maps in the background. Since old maps are usually heavily distorted due to the inaccuracy, we apply a simple distortion technique to those maps in order to align them to the current state of the city map.

2. Related Work

The idea of defining rules for the automatic creation of complex geometry has become a well established concept in plant modeling by utilizing L-Systems [PL96]. Since the process of city development behaves in a similar way to the growth process of a plant, the L-System formalism was extended by self-sensitivity and applied for the creation of street networks [PM01]. Additionally, building lots will be generated which are subdivided to create polygons that serve as floor plans. A common method to automatically create complex buildings from these floor plans was inspired by the concept of shape grammars [Sti75]. By proceeding in a strict coarse-to-fine fashion, facade details are generated by successively subdividing boxes and utilizing arbitrary geometry as atomic elements [WWSR03, MWH*06]. For some good overviews of common procedural techniques in the domain of city modeling, we would like to refer to the work of Vanegas et al. [VAW*09] and Watson et al. [WMV*08]. Although our paper strictly focuses on the street network generation for city development animations, we also present several images (and several video sequences in the accompanying video) of a simplistic 3D model that was derived from the street networks by extruding building footprints and randomly placing some vegetation.

While the coarse-to-fine method for simple buildings is a very intuitive and constructive approach, which can even be controlled interactively [LWW08], the modeling of plants relies on very professional knowledge for the definition of the rule set which mostly involves lots of abstract (and interdependent) parameters to adjust the outcome. Even a small change of one parameter value often results in unpredictable new results due to the highly recursive rule definitions within the L-System. Therefore, recent work has presented several

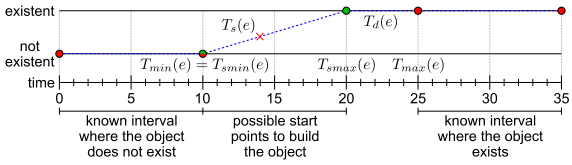

Figure 2: *The user can define intervals where a certain attribute of an object (here its existence) is known. Based on this information and on the duration $T_d(e)$ that is needed to complete the event, we derive the interval $[T_{smin}(e), T_{smax}(e)]$ in which the event has to be started in order to fulfill the known constraints. Our system will automatically compute a possible start point $T_s(e)$ which is influenced by external factors, interdependencies and a global distribution function.*

methods to guide the growth process [PHL*09, BvMM11] which was also applied for a high-level control of street networks [CEW*08, LSWW11, VABW09]. While these methods allow for an interactive manipulation of the resulting *structure*, our approach defines former states of this structure by a simple paint metaphor in order to animate its *development* over time. Even more important is the fact, that we want to interpolate real world data and not only artificially generated street networks. Although there is some research about the inverse procedural modeling problem, like deriving context-free L-Systems from a given structure [vBM*10], defining a self-sensitive L-System for the automatic creation of a specific (real world) street network is still an open problem.

In the research area of urban planning a carefully designed mathematical model can be used as a helping tool to make future decisions. It is however generally accepted that it is not possible to reliably predict the overall future development of a city with these models [HMWF03, Wu02]. At best, the dynamic development of a small subset of aspects defining the city can be extrapolated into the near future under the assumption that trends in the development do not change too quickly. This, however, will certainly not end in a previously specified state of the city. Seminal work has been presented for this kind of simulation based on agents [LWWF03, Wad02] or L-Systems [WMWG09]. While these methods are *model driven* and thus rely on a large amount of professional knowledge to define a complex model that produces a realistic *extrapolation* starting from an initial data set, we use a *data driven* approach for the *interpolation* of real world data (e.g. historic city maps) based on a rather simple model. This also enables us to easily simulate drastic changes in the city development, like the period of industrialization, by inserting a sample point (a historic city map) which provides the needed information. Of course, our intermediate stages of the city development do not necessarily agree with the real setting, but we can demonstrate in an experiment that, e.g., when we interpolate between a city map from 1865 and 1924, the interpolated result for the year 1906 is very similar to the ground truth.

3. Core Framework

Basically, a city map in our system consists of vertices, edges and polygons. Each primitive has a certain *object type* defining its semantics. A polygon, e.g., might be a floor plan of a house while an edge of a graph might be a street segment. Each of the object types can have several *attributes* that describe a particular instance of that object in the city map. Typical attributes are, e.g., the style and height of a house or the type of a street. The key idea of our approach is the simulation of these attributes over time by placing *events* in a timeline. An event in our system describes the change of an attribute value at a certain point in time (cf. Figure 2). Most importantly, the existence of an object is also expressed as an attribute which is provided by all object types. The existence is a simple flag that determines whether or not a specific object instance exists at a certain point in time.

As additional input to our system, each time step in the timeline can contain an image representing the land use at the specific year. Our application will automatically derive a possible image of the land use in between two explicitly given states (cf. Figure 3). In addition to visualization purposes, these maps are furthermore used by our algorithm for high level control of the shape of the city map development. By inserting images of economic maps into our system, the simulation will be also influenced by this meta information enabling the application of realistic effects like balancing the ratio of residential and industrial areas.

3.1. Events

An *event e* can be understood as starting a task that needs some time $T_d(e)$ to fulfill, e.g., changing the existence attribute of a house from *false* to *true* refers to starting the build process of that house which might take several years to be finished. The main problem here is, that the exact start date of the build processes is usually not known. In seldom cases, a precise information on this data can be found in historical books, but the majority of build events can only be soft constrained by specifying a time interval which can be derived from historical city maps. For example, if two city maps from 1800 and from 1850 are given, we assume that all objects that exist in the map from 1850, but not in the map from 1800, have been built meanwhile. Therefore, any event could theoretically be placed somewhere in between 1800 and 1850. More generally, the placement of events is limited by user defined soft constraints $T_{min}(e)$ and $T_{max}(e)$ reflecting the chronological successive values of a certain attribute. For an intuitive handling, $T_{min}(e)$ and $T_{max}(e)$ are placed in a timeline. While we can easily denote $T_{smin}(e) = T_{min}(e)$ as the earliest point in time for the event e to be started, $T_{max}(e)$ is not the latest point in time, because e still needs some time $T_d(e)$ to be fulfilled. Therefore, the latest possible start point for e is $T_{smax}(e) = T_{max}(e) - T_d(e)$. The actual start point of an event that has been placed by our system will be denoted as $T_s(e)$.

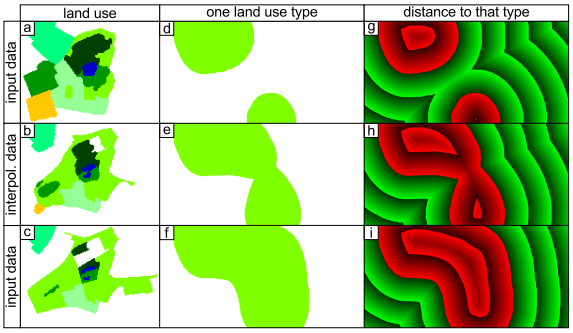

Figure 3: *By defining two land use images at two different points in time (a,c) we can automatically derive an intermediate land use image (b). We animate each land use type separately (d-f) by calculating the distances to the contour for each explicitly defined map (g,i) and linearly interpolating the pixel values (h). We then decide for each pixel separately to take the land use type that yields the highest value.*

3.2. Land Use

A land use map is an image providing meta information for the simulation or for visualization purposes. Our system provides layers of land use maps which are separately interpolated. Each layer contains several areas that are visualized by different colors. E.g. for a vegetation layer we can interpret the colors among others as forests, grass, or fields. The user can explicitly define land use maps at any time step by loading it from a file or directly painting it in our application. The interpolation between two explicitly defined time steps of one layer is done in three passes. First, we calculate a distance map for the contour of each color of all explicitly defined land use maps. Afterwards we linearly interpolate the values for all remaining time steps, yielding a distance map for each color in each time step. Figures 3.d - 3.i illustrate these passes with one color. Finally, for each pixel in the interpolated map, we take the maximal value of all distance maps that were generated for all the colors resulting in a smooth animation of the areas (cf. Figure 3.a - 3.c).

4. Simulation

For a plausible city development animation we have to take external factors into account that influence the automatic placement of events. Our system relies on *interdependencies* of the events to schedule their chronological order, a *global distribution function* restricting the number of events which are placed in each time step and a *scoring function* that determines the priority of a certain event. Initially, a dependency graph is created that reflects any relation between certain events such as the construction of a street segment that can only be started, if it is connected to the rest of the street network. Since the timeline has a limited resolution, e.g., one year steps, we can iteratively determine the set of events that are placed at each point in time T_c based on the scoring and the distribution function.

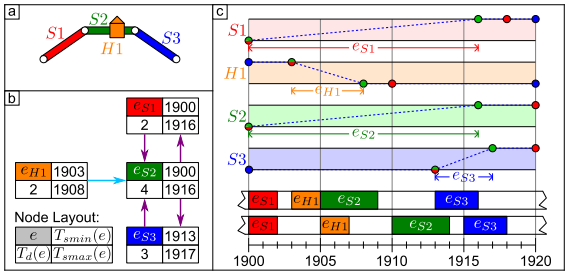

Figure 4: *This is an example situation in our system. (a) The scene consists of 3 streets and a house. (b) The dependency graph contains different colored edges with respect to alternative options, e.g., e_{S2} can start as soon as e_{H1} and (e_{S1} or e_{S3}) are finished. (c) The defined intervals for the scene. At the bottom of this image, we show 2 possible distributions for placing the events in the timeline without violating any constraints. The first one places all events as early as possible whereas the second one promotes a uniform distribution.*

In each step of the simulation, i.e., for each T_c, we have to apply the following calculations to all events that have not yet been started. We first compute the earliest point in time $T_{dmin}(e) \geq T_{smin}(e)$ (*lower bound*) when any of the unplaced events can be started with respect to its backward dependencies. The latest point in time $T_{dmax}(e) \leq T_{smax}(e)$ (*upper bound*) is then determined to guarantee that all forward dependent events can still be fulfilled. Based on the calculated interval for a possible placement and other external factors, e.g., the distance of an object to the city center, a score is identified that reflects the priority of a certain event. With respect to a global distribution function (e.g. a uniform distribution) the best events are chosen to be started at the current step T_c in the timeline.

For a consistent terminology, we will distinguish the following sets of events during the iterative simulation:

- $E_{remaining}$: events that have not been started so far
- $E_{finished}$: events that have already been finished
- $E_{progress}$: events that are still in progress
- $E_{placed} = E_{finished} \dot{\cup} E_{progress}$
- E_{all}: all events in the system ($E_{placed} \dot{\cup} E_{remaining}$)

Before the new events are started, which are determined by the simulation in time step T_c, we also have to consider that for all events $e \in E_{progress}$ one time step has passed since the last iteration. Therefore, we interpret the value $T_d(e)$ as the remaining execution time and decrement it in each time step by 1. Once an event $e \in E_{progress}$ has reached a value of 0 for $T_d(e)$, we remove it from the set $E_{progress}$ and put it into the set $E_{finished}$. Any new events that are chosen at the end of one step of the simulation are removed from the set $E_{remaining}$ and put into the set $E_{progress}$. Consequently, the set $E_{remaining}$ will be empty after the last step of the simulation.

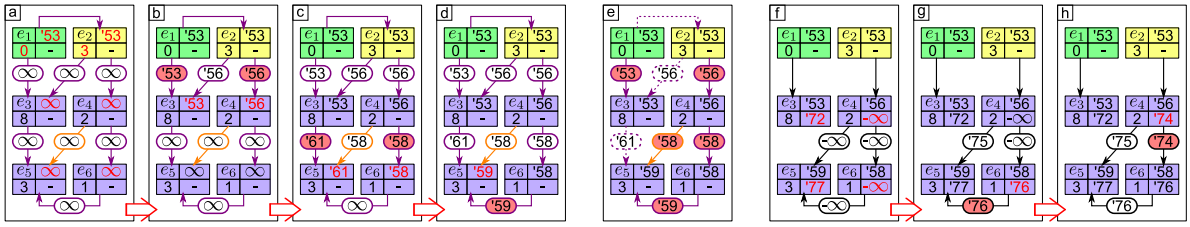

Figure 5: *Example for the computation of the lower (a-d) and upper (f-h) bound. The colors of the nodes visualize the sets $E_{finished}$ (green), $E_{progress}$ (yellow) and $E_{remaining}$ (blue). In this scenario we assume that each event is defined in the range $[1930, 1980]$ and the current time step T_c equals 1953. Each node contains the name (top left), its remaining build time $T_d(e)$ (bottom left), as well as the values $T_{dmin}(e)$ (top right) and $T_{dmax}(e)$ (bottom right). (a) Initialization: $T_{dmin}(e) = T_c \; \forall e \in E_{placed}$, $T_{dmin}(e) = \infty \; \forall e \in E_{remaining}$. (b-d) Iteratively update the lower bound. (e) All critical paths of the lower bound calculation result in a DAG which is the input to the upper bound computation. (f) Initialize $T_{dmax}(e)$ for all events e without outgoing edges with $1980 - T_d(e)$. All other nodes in $E_{remaining}$ are set to $-\infty$. (g,h) Iteratively update the upper bound.*

4.1. Dependencies

For a consistent city development animation, we have to take into account, that events may be dependent on each other. Figure 4, e.g., depicts the dependency relation for a build event e_{S2} of a street $S2$ which can only be placed if at least one of the adjacent streets $S1$ *or* $S3$ already exists *and* if the house $H1$ has been destroyed, i.e., one of the events e_{S1} *or* e_{S3} *and* the event e_{H1} are finished.

Formally, we represent the problem as a dependency graph $G(E_{all}, D)$ containing all events E_{all} as its nodes and all dependencies D as colored directed edges. Each element in D is a 3-tuple (e_1, e_2, c) with $e_1, e_2 \in E_{all}$ defining an edge $e_1 \leftarrow e_2$, i.e., e_1 depends on e_2, and a color $c \in \mathbb{N}$. Edges with the same color represent alternative options (which corresponds to an *or* relation, cf. Figure 4.b). The set of colors in the dependency relation of an event e is addressed as $C(e)$.

Currently, our system calculates the logical dependencies automatically from the given maps based on some simple rules or heuristics. Obviously, all objects are dependent on other objects that occupy the same space, i.e., at a specific point in time only one of these objects can exist in parallel. More specific rules include streets and railroads being dependent on the existence of any adjacent street or railroad, respectively, and houses being dependent on a street or another house in a certain distance. In our experiments, these rules already serve for a very realistic city map animation, however, our system was designed to be easily extendable by further rules if more specific use cases are needed.

4.2. Lower and Upper Bounds

For a chronologically correct city map interpolation, all events have to be executed and finished in their corresponding interval of the timeline with respect to their dependencies. Therefore, we need to prune the initial starting point interval $[T_{smin}(e), T_{smax}(e)]$ of an event e and calculate a new lower bound $T_{dmin}(e) \geq T_{smin}(e)$ which is the earliest point in time for the placement of e with respect to its dependencies. Afterwards, a new upper bound $T_{dmax}(e) \leq T_{smax}(e)$ is determined to guarantee that all other events e', which are dependent on e, can still be finished within their specified interval $[T_{min}(e'), T_{max}(e')]$.

Notice, our dependency graph makes use of conjunctive as well as disjunctive conditions. Thus, a simple *Dijkstra algorithm*, which is typically used in many search problems, is not sufficient for our dependency graph. The Dijkstra algorithm would only be applicable to our problem if there were just disjunctions involved. A *critical path method*, which is typically applied in task scheduling within the domain of *operations research* [HL02], is also not sufficient for our case, since it would only be applicable to our problem if there were just conjunctions involved. In this section, we describe our technique, which can handle both types of conditions, in form of a fixed point iteration to provide a clear mathematical exposition. In Section 7 we explain how to efficiently implement the algorithm.

At each time step T_c of the simulation, we calculate the lower bounds $T_{dmin}(e)$ by applying a kind of fixed point iteration method to all events $e \in E_{remaining}$. Initially, we set $T_{dmin}(e) = T_c$ for all $e \in E_{placed}$, because those events have already been placed by a previous step of the algorithm (cf. Figure 5.a). Note, that $T_d(e) = 0$ for all $e \in E_{finished}$ and that $T_d(e)$ is only the remaining time to finish an event $e \in E_{progress}$. Furthermore, we set $T_{dmin}(e) = \infty$ for all $e \in E_{remaining}$. For those events, $T_d(e)$ still refers to the execution time (i.e. the duration to process the event) that was initially set for each event. Since we know, that all events $e \in E_{all}$ can be started at $T_{dmin}(e)$ at the earliest, those events will be finished at $T_{dmin}(e) + T_d(e)$ at the earliest.

Each step in the fixed point iteration will now update the values $T_{dmin}(e)$ for all events $e \in E_{remaining}$ with respect to its dependencies (cf. Figure 5.b-d). Let $I_c(e)$ be the set of preceding events of e along the incoming edges with color c, i.e., $e' \in I_c(e) \Leftrightarrow (e, e', c) \in D$. For each color $c \in C(e)$, we then choose its preceding event $e' \in I_c(e)$ that yields the earliest point in time for starting the event e:

$$T_{dmin}(e, c) = \min_{e' \in I_c(e)} \{T_{dmin}(e') + T_d(e')\}$$

An event e can only be started, iff at least one event $e' \in I_c(e)$ for each color $c \in C(e)$ has been finished, so that the absolute earliest starting point $T_{dmin}(e)$ for the event e has to be determined by taking the maximum over all the previously calculated minima as long as this value does not lie below the defined earliest starting point $T_{smin}(e)$:

$$T_{dmin}(e) = \max\{T_{smin}(e), \max_{c \in C(e)} \{T_{dmin}(e,c)\}\}$$

We iterate the process until $T_{dmin}(e)$ has converged for all $e \in E_{remaining}$, i.e., more iterations do not improve the result.

For the calculation of the upper bound, we create a subgraph $G_u(E_{all}, D_u) \subseteq G(E_{all}, D)$ which is a directed acyclic graph (*DAG*) containing all *critical paths* that are needed to fulfill the dependencies of any event as early as possible. The edges D_u are 2-tuples that are formally defined as follows:

$$(e, e^*) \in D_u \Leftrightarrow \exists c \in C(e) : e^* = \arg\min_{e' \in I_c(e)} \{T_{dmin}(e') + T_d(e')\}$$

For these paths we calculate the upper bounds $T_{dmax}(e)$ of the events $e \in E_{remaining}$ as follows: Let $E_{end} \subseteq E_{remaining}$ be the set of events which have no incoming edges in G_u, i.e., there is no critical path containing an event e' that is a dependency for $e \in E_{end}$. Consequently, we can initially set $T_{dmax}(e) = T_{smax}(e)$ for all events $e \in E_{end}$, because those events could theoretically be placed at the end of the simulation. For all other events $e \in E_{dmax} = E_{remaining} \setminus E_{end}$, we have to set $T_{dmax}(e) = -\infty$.

Similar to the lower bound computation, we now apply a kind of fixed point iteration method to determine the values $T_{dmax}(e)$ for all events $e \in E_{dmax}$. Let $O(e)$ be the set of succeeding events of e along the outgoing edges in the DAG, i.e., $e' \in O(e) \Leftrightarrow \exists (e', e) \in D_u$. We then regard all dependent events $e' \in O(e)$ of an event e in all critical paths and choose the one that yields the lowest upper bound:

$$T_{dmax}(e) = \min\{T_{smax}(e), \min_{e' \in O(e)} \{T_{dmax}(e') - T_d(e)\}\}$$

Note, that this value is not allowed to exceed the latest possible starting point $T_{smax}(e)$. As soon as the process has converged, $T_{dmax}(e)$ is the latest starting point for the events $e \in E_{remaining}$ such that the critical paths in G_u could still be fulfilled, i.e., although the simulation might choose the time step $T_{dmax}(e)$ to start an event $e \in E_{remaining}$, it is still guaranteed that all events e' which are dependent on e can be finished within their defined interval $[T_{min}(e'), T_{max}(e')]$.

In rare cases, if the input data is corrupted, e.g., the street network is not fully connected, or intervals are wrongly defined, i.e., there is no chronological ordering of events that can fulfill the dependencies, the simulation might yield invalid bounds. In that case, we get $T_{dmin}(e) > T_{dmax}(e)$ for some events e. This conflict results from a contradicting relationship in the dependency graph. We overcome this problem by marking the corresponding objects in our application in order to give the user direct feedback of any wrongly defined intervals or any corrupted data.

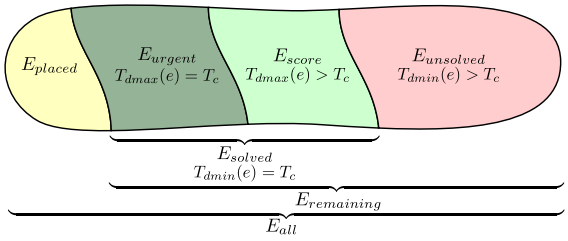

Figure 6: *This figure shows the different sets that are involved for the calculation of the scoring. First, we split the set $E_{remaining}$ into the sets E_{solved} and $E_{unsolved}$, i.e., whether or not they can be scheduled in the current step. Within the set E_{solved}, we furthermore have a subset E_{urgent} with events that have to be placed in the current time step in order to guarantee that all dependent events can still be finished in the user defined intervals.*

4.3. Score Rules and Land Use Maps

Based on the previously calculated lower and upper bound we divide the set $E_{remaining}$ into the two disjoint sets E_{solved} and $E_{unsolved}$ (cf. Figure 6). An event $e \in E_{remaining}$ belongs to the set E_{solved}, iff its dependencies can be fulfilled in the current time step T_c, i.e., $T_{dmin}(e) = T_c$. All the remaining events $e \in E_{remaining} \setminus E_{solved}$ are then put into set $E_{unsolved}$, i.e., $T_{dmin}(e) > T_c$. Furthermore, we classify events as *urgent* by putting them into the set $E_{urgent} \subseteq E_{solved}$ whenever $T_{dmax}(e) = T_c$. Those events have to be placed in the current step of the algorithm to guarantee that other events e' which are dependent on e can be finished within their defined interval $[T_{min}(e'), T_{max}(e')]$. This is one of the most essential points within our simulation, since the interdependencies ensure an interpolation of the input data.

For all other events $e \in E_{score} = E_{solved} \setminus E_{urgent}$ we calculate a normalized score $S(e) \in [0.0, 1.0]$, where a value closer to 0.0 represents a rather unimportant event whereas a value closer to 1.0 reflects an important event for the current time step. The score can be influenced by several factors. In our specific implementation, $S(e)$ mainly relies on the following observations: Since an event becomes more urgent as it approaches its upper bound, we basically take the previous calculation of the upper and lower bounds as a first approximation on the importance of an event by calculating $S(e) = 1.0/(T_{dmax}(e) - T_c)$. We also have to take into consideration, how many events $N_{dep}(e)$ are still dependent on an event e in the overall system. The score is then weighted by $N_{dep}(e)/|E_{remaining}|$ to assign a higher priority to events on which more other events are dependent. This ensures a more balanced animation, since we avoid large sets E_{urgent} of urgent events in later simulation steps.

The overall shape of the city development will be controlled in several ways. The distance to a defined city center, e.g., prevents the city from growing too fast into an arbitrary direction. Our system can also apply weightings of the score by considering several land use maps. Instead of just using

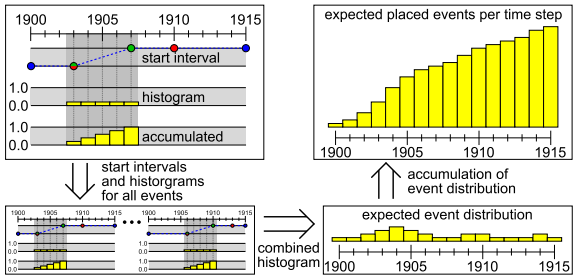

Figure 7: *For a globally balanced animation, we derive an event distribution by taking into account all the user defined intervals. Each interval is converted into a uniformly distributed histogram. The accumulated histogram reflects the probability, that the event is placed in a certain time step. Summing up all the histograms of all events yields the global event distribution and by accumulation we get the expected number of events that have to be started at each time step.*

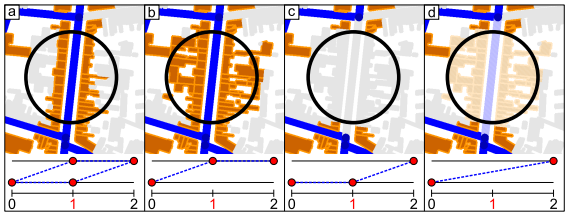

Figure 8: *This figure illustrates the behavior of the paint metaphor, when the existence attribute of several object classes is modified at time step* 1*. Here we assume that all objects do not exist (grayed) at time step* 0 *whereas at time step* 2 *they have to exist (colored). (a) Initially, the attribute value differs for several objects that lie under the brush circle. (b) The user paints with the attribute value* "exists"*. (c) The user paints with the attribute value* "does not exist"*. (d) The user deletes the explicit definition of attribute values at the current time step. Whenever the currently selected year T_c lies in the defined interval of a build or destruction event of an object, the objects are drawn in grayed colors.*

a city center, we can thereby animate the overall shape of the city development, i.e., streets and buildings are only allowed to be built in designated areas. Moreover the ratio of residential and industrial areas can be controlled by setting a higher score for all events that belong to the currently disadvantaged type.

The provided rules and land use maps we presented already serve for a plausible animation of city map development, however, the system is flexible enough to easily insert any alternative scoring rules for specific fine tuning.

4.4. Histogram

The logical rules, i.e., the interdependencies and the scoring, serve for a well defined local behavior of the city development. However, this is not sufficient to get a global control of the animation. Our system uses an accumulated histogram that determines the number of events for each time step in order to create a realistic event distribution (cf. Figure 7). Based on the start intervals that are defined for all events, we derive the absolute number $N_{abs}(T_c)$ of events that should have been placed in the overall system at each simulated time step T_c. More specifically, we increment the accumulated histogram at each time step T_c within the interval $[T_{smin}(e), T_{smax}(e)]$ for each event $e \in E_{all}$ by an amount of $1.0/(T_{smax}(e) - T_{smin}(e))$. This perfectly reflects the impact of each event to the overall animation and produces a balanced distribution based on the real data input. Let $N_{placed}(T_c)$ be the number of events $|E_{placed}|$ that are currently placed in the simulation at time step T_c. We can calculate the actual number $N_{rel}(T_c)$ of events that is placed at a time step T_c as follows: $N_{rel}(T_c) = max(0, N_{abs}(T_c) - N_{placed}(T_c - 1))$.

After each step T_c of the simulation, we take the complete set E_{urgent} as well as the $N_{rel}(T_c) - |E_{urgent}|$ best events $e \in E_{score}$ with respect to their score $S(e)$ and put them into the set $E_{progress}$. In the ideal case, $N_{abs}(T_c)$ is equal to $|E_{placed}|$ for each time step. However, in rare situations we observe $|E_{urgent}| > N_{rel}(T_c)$ or $|E_{solved}| < N_{rel}(T_c)$. In the former case, more events are currently placed in the system than the accumulated histogram expected, i.e., $|E_{placed}| > N_{abs}(T_c)$, whereas in the latter case, we are not able to place the expected number of events, i.e., $|E_{placed}| < N_{abs}(T_c)$, because too few of the events fulfill their dependencies. Since the calculation of $N_{rel}(T_c)$ is based on the already placed events, this effect is amortized after several simulation steps and will not heavily influence the overall appearance of the animation. In practice, we observe such cases very rarely and they are usually amortized after one further simulation step.

5. Information Authoring

Shape Modeling Our system is capable of creating, changing, or deleting any of the primitives from the input maps, i.e., vertices, edges, or polygons. In practice, a lot of work is saved by just loading the current street network and floor plans from any available resources such as OpenStreetMap [Ope10].

Timeline Editing For any precise data, the user can define certain attribute values in the timeline which implicitly defines the intervals for the placement of the events. Since it would be too time consuming to apply this approach for animating the attribute values of a whole city with thousands of elements, we introduce a *paint metaphor* to edit the attribute values of a wide range of objects in parallel. The user first selects the class of objects that should be edited, e.g., streets and houses. Afterwards, a certain attribute value is defined which will be used for the painting process. All elements

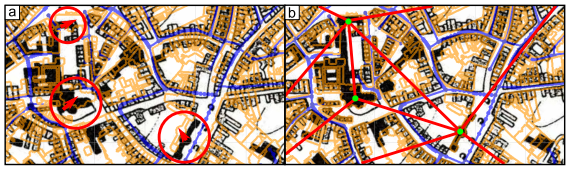

Figure 9: *(a) Historical city maps are usually distorted. (b) The user can define correspondences between the historical map and the current city map. We then apply a deformation based on a Delaunay triangulation. By using the deformed image in the background, the user can efficiently paint the attribute value* "does not exist" *for any objects that were not yet build in past decades.*

that share the specified attribute and that touch the current brushstroke will insert the specified value into the timeline (cf. Figure 8). This interaction metaphor becomes a powerful tool, if images of historical city maps are loaded to the background to support the editing process.

Historical City Maps Unfortunately, historical city maps are usually heavily distorted, because the people did not have any precise measuring devices or satellite images during that time. Assuming, that today's city maps are correct and that large parts of old city maps contain a subset of today's objects, we can apply a simple distortion technique to the historical maps (cf. Figure 9). First, an image of a historical map is loaded to the background which is then manually scaled, rotated and translated to align with the current map as well as possible. Afterwards, a large subset of the street crossings are already well-aligned with the current street network. For most of the remaining crossings, however, it is obvious that they should match the current street graph and that this error comes from an imprecise measure. The user can now easily establish the correspondences by placing a *source position* in the image of the historical map and setting its *target position* in the current state of the city. The actual distortion is then calculated as follows. A Delaunay triangulation is applied to the source positions including the image corners which divides the original image into triangular sub-images. The triangles are then deformed to fit the target positions and the contained image data is bilinearly interpolated.

Land Use Maps For the land use maps, the user can load existing images or simply draw onto an existing map by a typical paint metaphor using different colored brushes. For a more efficient workflow, we allow locking of certain areas, i.e., locked areas will not be affected by the current paint process to prevent changes. This feature turned out to be very helpful in situations, where precise maps exist for a certain area type. In that case we are able to load the existing maps and just paint additional information into the remaining areas without touching the imported data.

6. Results

We applied our method to two completely different city styles, i.e., a Roman-founded European city (Aachen, Germany, cf. Figure 1) and a planned American city (Austin, TX, USA, cf. Figure 10). While the former is dominated by two major streets that define a circular ring layout and many minor streets that behave in a very unstructured way, the latter one is mainly dominated by a completely rectangular street pattern. We also created an artificial example that demonstrates the application of several land use maps (cf. Figure 11). However, we highly recommend to watch our accompanying video, since the overall impression of the animations can be hardly expressed by a set of static images.

For the city development of Aachen over 210 years (1800 - 2010) we used 8 images of historical maps, that contained significant changes over time, and several precise items of information from books to recover the timeline of landmark buildings. With our interactive software, we could easily produce digital versions of the historical maps based on the imported street network of the current city state. The preparation for the simulation took ∼8 hours of interaction including the distortion of the 8 historical city maps, painting the existence attribute according to these maps, inserting several historic buildings, defining specific timelines for a few landmark buildings and creating several land use maps. Note, that 8 hours are not very much if one takes into account that over 50k events have been inserted to our system.

After the simulation step, we generated a 3D model for each time step by extruding the floor plans and randomly distributing trees in forest regions based on the interpolated land use maps. For the texturing of the ground we project the maps of the land use onto the basic terrain. Furthermore, we improve realism by adding some effects like a sky box, procedural clouds, shadows and ambient occlusion. This enables us to create a visually pleasing flyover while the city map development is being animated.

We also compared the interpolated result of our system with the ground truth of an existing map by successively deleting maps from the simulation. Removing a small set of maps does not change the result very much, however, if we just keep the first and the last state, our simulation will not fully match the real history as external factors like wars or industrialization are unpredictable (cf. our accompanying video). As long as the user can provide information of times with significant changes, the produced animation will much more likely fit the real city development, since our procedural interpolation ensures that the animation perfectly matches the inserted data. Furthermore, if it is only known that an extreme external factor is influencing the city growth during a certain time period, like industrialization, but no explicit data is available, we could reflect this kind of behavior by changing the global distribution function. This would be part of future work, since our current implementation fully derives the distribution function from the object timelines.

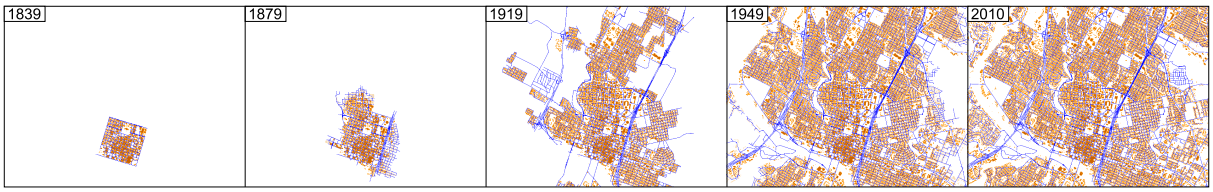

Figure 10: *Animation of Austin, TX from 1839 - 2010 based on 6 historical maps (1839, 1872, 1885, 1919, 1940, 2010).*

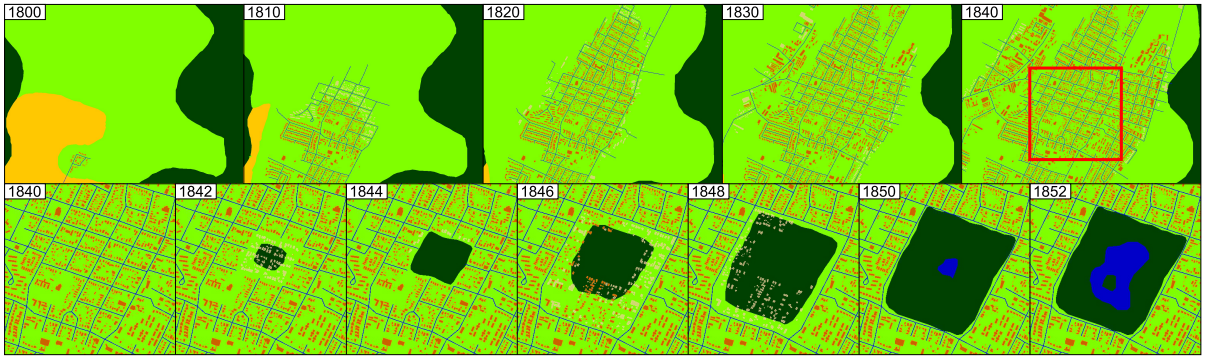

Figure 11: *Artificial example to demonstrate our method. The overall growth behavior of the city is specified by a shape layer at 1820 and 1840. Therefore, the city shape will first expand to the upper right (till 1820) before it starts growing into the other directions. The close-up (red rectangle in 1840, bottom sequence) shows the development of a park in the center of the city. The whole example was created in ∼1.5 hours including the creation of the land use maps for the years 1800, 1820, 1840, 1850 and 1852 (∼1h) and the painting of events for the years 1800, 1840 and 1852 (∼0.5h).*

In the second experiment, we generated an animation of the city development of Austin over 171 years (1839 - 2010). The animation is based on the information of 6 historical city maps. Although the street layout of Austin differs a lot from the one of Aachen, this does not affect our method since it utilizes existing street networks as input data. The overall interaction time for this example was only 4 hours, as we just used the paint metaphor to edit the timeline without inserting new objects or defining specific timelines for landmark buildings. Therefore, the workflow is even more efficient than for the Aachen example, because over 171k events have been inserted within just 4 hours. In this case, the hardest part was to undistort the images of the historical city maps due to their dimension. Although our inserted information was very sparse, the resulting information still looks authentic (cf. our accompanying video).

7. Discussion

Implementation Our animation tool is implemented in C++ using the GPU for an efficient drawing of the primitives. Since the calculation of the lower and upper bounds takes the biggest part of the computation, we have highly optimized the process by combining a breadth first traversal starting from the set E_{placed} and FIFO queues for the current working set, i.e., whenever an event calculates a better value for itself all dependent events will be enqueued for the next iteration. Thus we do not need to touch all events in each iteration, but only those that could possibly update their value. Running our application on an Intel Core i7 at 2.67GHz our algorithm takes ∼2 minutes for the animation of Aachen (210 time steps involving 50k existence events) and ∼5.5 minutes for the animation of Austin (171 time steps involving 171k existence events).

Manual Effort Currently, our application relies on several manual steps to insert the data into the simulation. Especially for the map alignment, the user has to insert around 15 to 30 correspondences in order to get a sufficiently accurate image distortion. In our first example, the biggest amount of time was consumed for the extraction of exact dates from history books which is mostly important for landmark buildings. We believe that most of the interaction processes can be automated in the future like the detection of correspondences between the historical map images and the given vector data of the current city state [cCKS*04] or the use of history databases (if existent) for detailed information on specific buildings.

Deadlocks In some situations it might happen, that the interdependencies can not be resolved, e.g., if two different houses are defined to exist at the same point in time at the same location. This is due to the fact that we do not check the input for causal correctness in advance. Therefore, we give a visual feedback in the city map viewer to efficiently detect and fix such configurations. Notice that such inconsistencies cannot result from inconsistent maps. For example, if a house appears on a map in 1950 and 1970 but not in 1960 then our procedural interpolation will build the house before 1950, destroy it between 1950 and 1960, and re-build it between 1960 and 1970.

8. Future Work

Model vs Data We currently use a simplistic model for the urban development. However this was made with the intention to demonstrate the capabilities of a purely data driven approach. A model driven approach cannot easily be generalized and adjusted to different city types (e.g. a planned US city like Austin vs. a historically grown European city like Aachen). Hence, our approach was to rather extract the characteristic dynamics of the city development from historical maps instead of adjusting abstract parameters in a synthetic model. Of course, if such a model is available (domain knowledge) this information can be integrated into our simulation by adjusting the scoring function accordingly (e.g. by increasing the priority for the extension of major roads).

Visualization We would like to utilize the key concepts of our approach to enhance our 3D visualization by animating the construction and the destruction of any entities like buildings or streets. Our paper presents an intuitive way to produce such kind of animations by uncoupling the generation of a certain *structure* from its *development* over time.

9. Conclusion

This paper proposes a novel approach for the interpolation of several city maps supported by additional historical information resulting in a plausible animation of the city development. By applying an intuitive painting metaphor, a user can efficiently insert historical events into our system regardless of whether the information is given in textual form or as an image of an old city map. We then interpolate the discrete setting in a procedural manner by determining the actual time step of a certain event taking several constraints into account like interdependencies, several scoring functions and a global distribution function. Together with a linear interpolation of the underlying land use maps the development is visualized by a virtual 3D city model. This communicates an intuition of a possible city growth to the observer which is more attractive than comparing a fixed set of historical maps.

Acknowledgement

This work was supported in part by NRW State within the B-IT Research School.

References

[BvMM11] BENEŠ B., ŠŤAVA O., MĚCH R., MILLER G.: Guided procedural modeling. *CGF (Eurographics) 30*, 2 (2011), 325–334. 3

[cCKS*04] CHIEN CHEN C., KNOBLOCK C. A., SHAHABI C., YI CHIANG Y., THAKKAR S.: Automatically and accurately conflating orthoimagery and street maps. In *In Proceedings of the 12 th ACM International Workshop on Geographic Information System, ACM-GIS* (2004), ACM Press, pp. 47–56. 9

[CEW*08] CHEN G., ESCH G., WONKA P., MÜLLER P., ZHANG E.: Interactive procedural street modeling. In *ACM SIGGRAPH 2008* (NY, USA, 2008), ACM, pp. 103:1–103:10. 3

[Fri08] FRISCHER B.: The rome reborn project. how technology is helping us to study history. In *OpEd* (2008), University of Virginia. 1

[HL02] HILLIER F., LIEBERMAN G.: *Introduction to Operations Research*. McGraw-Hill Science/Engineering/Math, 2002. 5

[HMWF03] HAMMAM Y., MOORE A., WHIGHAM P., FREEMAN C.: A vector-agent paradigm for dynamic urban modelling. In *Proceedings of 15th Annual Colloquium of the Spatial Information Research Centre* (Dunedin, New Zealand, 2003). 3

[LSWW11] LIPP M., SCHERZER D., WONKA P., WIMMER M.: Interactive modeling of city layouts using layers of procedural content. *CGF (Eurographics) 30*, 2 (2011), 345–354. 3

[LWW08] LIPP M., WONKA P., WIMMER M.: Interactive visual editing of grammars for procedural architecture. In *ACM SIGGRAPH 2008* (NY, USA, 2008), ACM, pp. 1–10. 2

[LWWF03] LECHNER T., WATSON B., WILENSKY U., FELSEN M.: Procedural city modeling. *1st Midwestern Graphics Conference* (2003). 3

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., GOOL L. V.: Procedural modeling of buildings. In *ACM SIGGRAPH 2006* (NY, USA, 2006), ACM, pp. 614–623. 2

[Ope10] OPENSTREETMAP:. http://www.openstreetmap.org/, Dec., 2010. 7

[PHL*09] PALUBICKI W., HOREL K., LONGAY S., RUNIONS A., LANE B., MĚCH R., PRUSINKIEWICZ P.: Self-organizing tree models for image synthesis. In *ACM SIGGRAPH 2009* (NY, USA, 2009), ACM, pp. 58:1–58:10. 3

[PL96] PRUSINKIEWICZ P., LINDENMAYER A.: *The algorithmic beauty of plants*. Springer-Verlag, NY, USA, 1996. 2

[PM01] PARISH Y. I. H., MÜLLER P.: Procedural modeling of cities. In *ACM SIGGRAPH 2001* (NY, USA, 2001), ACM Press, pp. 301–308. 2

[Sti75] STINY G.: *Pictorial and Formal Aspects of Shape and Shape Grammars*. Birkhauser Verlag, Basel, 1975. 2

[VABW09] VANEGAS C. A., ALIAGA D. G., BENEŠ B., WADDELL P. A.: Interactive design of urban spaces using geometrical and behavioral modeling. In *ACM SIGGRAPH Asia 2011* (NY, USA, 2009), ACM, pp. 111:1–111:10. 3

[VAW*09] VANEGAS C. A., ALIAGA D. G., WONKA P., MÜLLER P., WADDELL P., WATSON B.: Modeling the appearance and behavior of urban spaces. *CGF (Eurographics -STAR) 29*, 1 (2009), 25–42. 2

[vBM*10] ŠŤAVA O., BENEŠ B., MĚCH R., ALIAGA D. G., KRIŠTOF P.: Inverse procedural modeling by automatic generation of l-systems. *CGF (Eurographics) 29*, 2 (2010), 665–674. 3

[Wad02] WADDELL P.: Urbansim: Modeling urban development for land use, transportation and environmental planning. *Journal of the American Planning Association 68* (2002), 297–314. 3

[WMV*08] WATSON B., MÜLLER P., VERYOVKA O., FULLER A., WONKA P., SEXTON C.: Procedural urban modeling in practice. *IEEE Comp. Graph. and App. 28*, 3 (2008), 18–26. 2

[WMWG09] WEBER B., MUELLER P., WONKA P., GROSS M.: Interactive geometric simulation of 4d cities. *CGF (Eurographics) 28*, 2 (2009), 481–492. 2, 3

[Wu02] WU F.: Calibration of stochastic cellular automata: The application to rural-urban land conversions. *International Journal of Geographical Information Science 16* (2002), 795–818. 3

[WWSR03] WONKA P., WIMMER M., SILLION F., RIBARSKY W.: Instant architecture. In *ACM SIGGRAPH 2003* (NY, USA, 2003), ACM, pp. 669–677. 2