DOI: 10.1111/cgf.13897

COMPUTER GRAPHICS *forum*
Volume 39 (2020), number 1 pp. 650–671

Simulating the Evolution of Ancient Fortified Cities

Albert Mas, Ignacio Martin and Gustavo Patow

ViRVIG, Universitat de Girona, Spain
albertmas@dah-rd.com, ignacio.martin@udg.edu, dagush@imae.udg.edu

Abstract
Ancient cities and castles are ubiquitous cultural heritage structures all over Europe, and countless digital creations (e.g. movies and games) use them for storytelling. However, they got little or no attention in the computer graphics literature. This paper aims to close the gap between historical and geometrical modelling, by presenting a framework that allows the forward and inverse design of ancient city (e.g. castles and walled cities) evolution along history. The main component is an interactive loop that cycles over a number of years simulating the evolution of a city. The user can define events, such as battles, city growth, wall creations or expansions, or any other historical event. Firstly, cities (or castles) and their walls are created, and, later on, expanded to encompass civil or strategic facilities to protect. In our framework, battle simulations are used to detect weaknesses and strengthen them, evolving to accommodate to developments in offensive weaponry. We conducted both forward and inverse design tests on three different scenarios: the city of Carcassone (France), the city of Gerunda (Spain) and the Ciutadella in ancient Barcelona. All the results have been validated by historians who helped fine-tune the different parameters involved in the simulations.

Keywords: geometric modelling, modelling, polygonal modelling, modelling, human simulation, animation

ACM CCS: • Computing methodologies → Computer graphics, Mesh models

1. Introduction

Since the Persian empire and up to the modern Napoleon wars, cities have been protected by castles and fortresses, shaped by protective walls (also known as curtain walls) against enemy attacks. These constructions were not static defensive objects, but evolved to improve the defence level and to face evolution of attack weapons and strategies, while at the same time encompassing city growth. The results of these changes can still be observed nowadays in historical cities. For instance, modern European cities that had ancient defensive systems still have traces of their walls, even if these walls do not exist anymore: we can still clearly see their silhouettes drawn in their street networks.

Also, any large-scale change at the urban level must include a careful planning, and any excavation in the old neighbourhood of any old European city will find archaeological remains. Having a tool that helps to roughly estimate the past urban changes can help optimize future interventions, which is the role of urban planners.

Previous research in historical urban evolution has been mainly done in the context of archaeology, history or art history (e.g. [Duf79, Lep02]). The results in these fields are intended for improving our understanding of our past and religious, economical, political and contextual factors that shaped our world as it is now. However, in general, their results are obtained in a case-by-case basis, resulting in localized explanations that are difficult to generalize to other contexts. Only recently the emerging field of historical dynamics and its mathematical formulation, Cliodynamics [Tur11], attempt to obtain general solutions that are applicable to a wider context. However, having a consistent and reliable simulation of the evolution of ancient cities and castles is a complex problem that has not been studied deep enough. This is even more surprising if we consider their possible impact on video games, cinema, cultural heritage and even urban planning. In computer graphics, in spite of the spectacular advances in urban modelling techniques over the last years [KM06, WMV*08, STBB14], simulating evolution of cities as time-dependent objects [HMFN04, WMWG09, EBP*12, BWK14] has mainly concentrated on geometric aspects, and not on underlying historic facts that lead to such changes. Some research has been done to feed behavioural data into geometric modelling systems [VABW09, VGDA*12], but its focus is on designing and editing urban environments, not on trying to simulate their historical

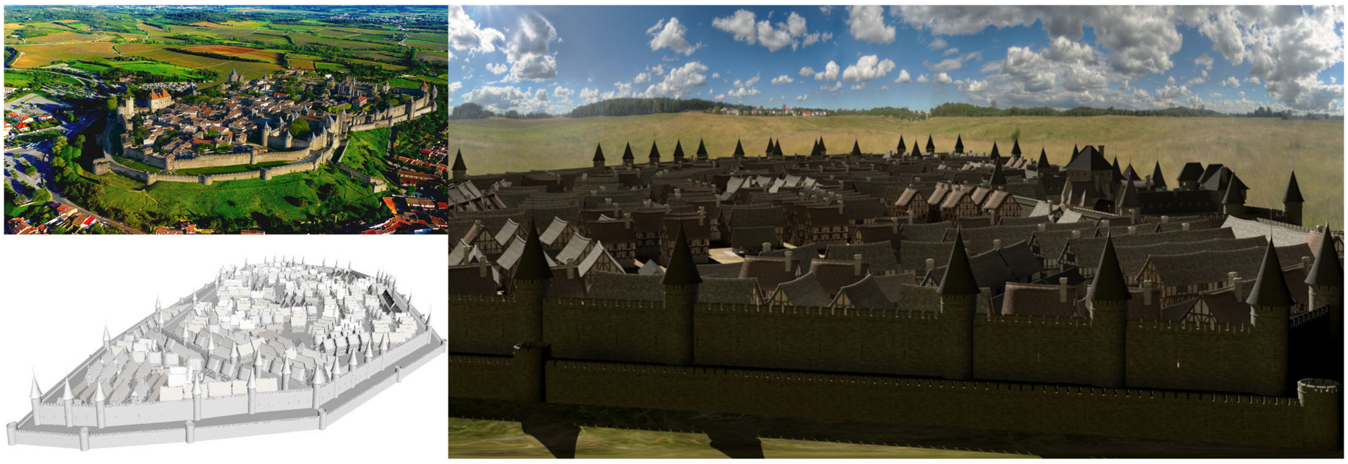


Figure 1: *Left: (top) real photo and (bottom) simulation of Carcassone. Right: a more realistic rendering.*

evolution. On the other hand, although different city models can be manually created by an artist, this would be focused only on visual realism and not on historical plausibility. Users may not have intuition about the mechanics that governed fortification construction, or knowledge of traditional shapes, sizes or proportions used in city design. Determining the precise shape that guarantees defensiveness can be an error-prone task that our system pretends to reduce.

In this work, we propose to close the gap between historical simulation and geometrical modelling of urban spaces, by modelling historic battles and the effect they produced on city layouts through their defensive walls. We model ancient historical facts (i.e. battles, and urban and castle evolution) as a succession of time-dependent events, which represent historical changes. In an interactive environment, we allow the user to define, at each iteration, a set of discrete events that describe the influence of the different evolution factors, such as city growth, creation of new walls or simulation of defence improvement against external attacks. Then the system runs by simulating the evolution of ancient cities as a dynamic process driven by these factors between two given dates presents the results and allows the user to continue interacting with the system. We can state that, what this paper demonstrates, given a general knowledge of castle evolution, it can be converted into a more concrete historically based castle evolution process.

Our main contribution is an interactive user-controllable simulation of the temporal evolution of ancient cities or castles based on events, such as city growth, wall construction or battles. In particular, our concrete contributions are:

- an interactive, agent-based system that allows the accurate simulation of real historical events, such as battles or city changes resulting from their natural population growth;
- a detailed geometrical model of Western ancient defensive structures, such as walls, towers, ravelins, bastions and many others;
- a comprehensive and accurate battle simulation environment that takes into account common situations in ancient battles.

2. Previous Work

In the field of procedural modelling, the seminal work by Parish and Muller [PM01] and Muller *et al.* [MWH*06], based on the idea of L-systems, is a key reference for procedural techniques in urban modelling. Another work based on procedural rules was presented by Sun *et al.* [SYBG02], using a template-based approach for street network generation. Chen *et al.* [CEW*08] proposed a method that uses tensor fields for the generation of street graphs. Merrell and co-workers [Mer07, MM08] proposed a procedural modelling system based on input examples that are processed, partitioned (either discretely [Mer07] or continuously [MM08]) and then used to generate new models based on the found geometric patterns. For more information, we refer the reader the surveys from Kelly and McCabe [KM06], Watson *et al.* [WMV*08], Vanegas *et al.* [VAW*10] or Smellik *et al.* [STBB14].

Considering urban evolution, Weber *et al.* [WMWG09] presented a method to grow streets based on an L-system approach, but using traffic simulations. This work was extended by Benes *et al.* [BWK14] who considered trading as a moving force, also taking the city neighbourhood into account. Vanegas *et al.* [VGDA*12] presented a method to simulate and to predict urban environment evolution using inverse modelling techniques. Emilien *et al.* [EBP*12] presented a method for the modelling and evolution of small, European villages taking into account the terrain features to increase safety, sunlight illumination or other convenience criteria. However, none of that work considered a historical setting such as we do in this paper, and they were not concerned about the evolution of cities during the turbulent ages when castles were main defensive structures.

As opposed to geometric urban modelling, which is purely computer graphics-oriented, behavioural urban modelling is intended for decision-making regarding urban policies in current and future urban areas. Some of the most important examples are the work of Alkheder *et al.* [AWS08] and Waddell [Wad02]. Vanegas *et al.* [VABW09] tried to close the gap between behavioural and geometric modelling by subdividing the urban space into a regular grid of cells, each cell with a set of associated variables that control the distributions of population, jobs, land values, road networks, parcel shape and building geometry. Their behavioural framework—which is based on a simplification of *Urban Sim* [Wad02]—and their geometrical modelling engine produce a single dynamical system that seeks behavioural and geometric equilibrium after each user-specified change. As we can see, all that work studies the evolution of socio-economical aspects of the city, allowing accurate predictions for *present* and *future* cities, but does not deal with historical battles or defence-driven urban evolutions.

Closely related to our objectives, the new field of historical dynamics and its mathematical formulation, Cliodynamics [Tur11], focuses on finding a scientific modelling of history, including its computer modelling [GMK13]. Collins [Col10] modelled victory or defeat in battle as a set of flow charts for dynamic simulation. Later on, Fletcher *et al.* [FAR*11] continued this work by presenting a simulation of the Civil War that appears to coincide with historical reality, demonstrating the validity of models proposed so far. Our work shares some common concepts, but with its main focus on the design of walled cities from scratch, and being able to produce complete models within minutes.

Medieval castle architecture and its evolution are well documented in historical literature [Lep02, Duf79]. Although each castle and city underwent different changes depending on its location, culture [Duf79], interior and surrounding terrains and available resources, some common facts can be found. For example, sieges were one of the main reasons to modify the city configuration [Gri06, Hin09]. As weapons and siege techniques evolved, defensive elements such as walls and towers had also to evolve to keep guaranteeing a safe interior for city inhabitants. However, ranging from the Roman empire [RAJ*03] to the Napoleonic wars [Hof11], the different number of war strategies and weapon types to take into account is huge. The last periods in castle evolutions, just before they were dropped as defensive structures, were popularized by Vauban's star-fortress designs [Duf85, Lep09]. Examples of star-shaped fortified cities are Geneva, Bayonne or Naarden. Other fortresses such as the Ciutadella of Barcelona were typical examples of military fortifications that did not evolve from a previous castle, but were built directly following the latest defensive structures of the time. The objective of this work is to focus on common battle settings and weapon types that could affect castle or city configurations, closely following historical developments of these structures.

Also, parts of this work are related to inverse procedural design. Talton *et al.* [TLL*11] developed a metropolis-based procedural technique that was used to generate, from some very simple targets, complex models derived from the city grammar targeted to skylines of different silhouettes, such as a whale, a shoe and even the Stanford bunny. Beneš *et al.* [BvMM11] presented a guided procedural system which allows to generate and edit a procedural model of a tree or a complex urban layout. After the original model is generated, its guides are interactively edited by means of a mass-spring model to achieve a desired layout. Bokeloh *et al.* [BWSK12] presented a system where an input shape is automatically analysed using shape understanding techniques to extract regular translational patterns. Their algebraic model of shape regularity identifies so-called *useful* degrees of freedom, which are exposed for robust real-time shape editing as controllers visible on the building surface. Demir *et al.* [DAB14] presented city-scale building proceduralization technique that uses an unorganized 3D model as input to compute a hierarchical clustering of building components, from which it extracts a context-free grammar of the urban area. With this technique, it is possible to procedurally generate structurally similar cities to the provided example. Stava *et al.* [SPK*14] studied techniques for inverse procedural modelling applied in the context of tree generation, but given the similarities between the techniques used for trees and buildings, this work is worthwhile taking into consideration. From an input model of a tree model, their system performed some pre-processing and estimated the input parameters of the developmental model, so the overall system was able to produce stochastically similar trees, also being able to produce environmentally sensitive trees models (e.g. under the influence of obstacles). Demir and Aliaga [ÄDA18] presented a method called Guided proceduralization, that, from an input building model, it converts it to a procedural editable model. Moreover, as the user specification on the target grammar changes, the system reveals different grammars of the model. Garcia-Dorado *et al.* [GDAB*17] posed an inverse algorithm where local weather is changed to achieve a given set of conditions in the city, by changing constructive parameters of the city, as well as characteristics of the terrain and the initial weather conditions. We follow their methodology for approaching this complex problem, but in the context of historical battles.

Despite of being outside of the academic world, we cannot ignore the many video games based on medieval simulations (including war simulators) [Wik19d, Wik19c, Wik19a, Wik19b]. In general, we can say that our system could benefit from their simulations or output data, but it must be noted that our system is not, and will never be, as realistic as they are. The reason for this crucial difference is that these systems are built for simulating a particular battle, or a whole war, but not an evolving city as our system does. Thus, although somewhat similar and valuable, these video games aim at a completely different objective than our system.

3. Overview

Our framework is based on a simulation that runs for a user-defined number of years, and, for each year, the city evolves controlled by a set of user-defined parameters and events. See Figure 2. The method presented in this work is based on several assumptions (see below), and controls three main aspects of the simulation: city growth, wall evolution and battles. In our system, this evolution has the only

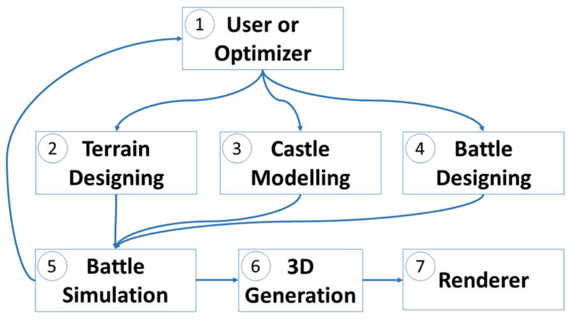


Figure 2: *System pipeline: We show a diagram of the overall pipeline of our approach.*

purpose of defining the new parameters of the city, providing the basis for city evolution. Although more sophisticated methods can be used for growth, we decided to simply generate new buildings using parameters such as direction, density factor and a building-per-year ratio.

The key element of our framework is an interactive, user-controlled loop where the user is free to redefine the evolution parameters (e.g. growth) and to introduce time-dependent events, such as sieges, battles, wall creation or reinforcements, city growth or even introduction of new historical elements such as bastions or a star-fort. Then, with these definitions, the system runs the urban/castle evolution simulation. To this end, the user uses an interactive environment where they can configure any parameter of the simulation, from city growth to battalion deployment (Figure 3). Depending on the era being simulated, the system automatically selects the correct elements in the simulation (e.g. archers or riflers, different kinds of wall reinforcements or even population redistributions, as introduced by Vanegas *et al.* [VABW09]). Each iteration of this loop is called a *turn*.

A walled city (or castle) evolution is related to its defensive capabilities against attacks. We identify weaknesses in city defences by simulating battles at user-selected dates. Simulations use two kinds of agent groups: defenders, within the castle; and attackers, on the castle surroundings. Each army defines its own agents, named battalions. There are different kinds of battalions depending on its battle skills, such as speed, defence factor and shooting accuracy, among others. These skills are randomly combined for each battalion to perform actions such as movements, shootings, climbings or target acquisitions. Each battalion takes its own decisions, ruled by its skills and the current battle status. From a technical point of view, each battalion is implemented as an agent controlled by a behaviour tree, and sharing a common blackboard memory for easy access and shared resources.

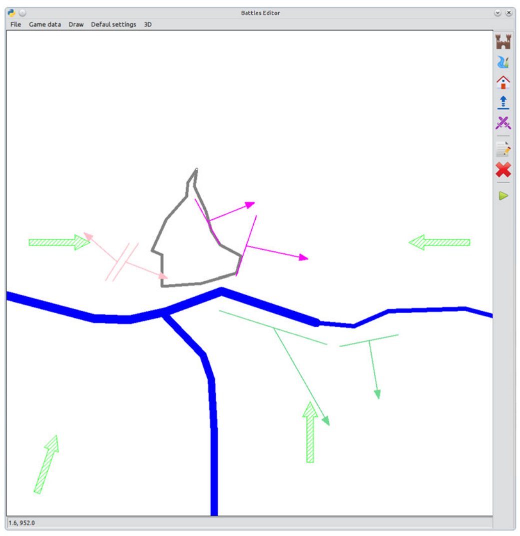


Figure 3: *User interface showing the setting for the evolution of the Gerunda medieval city showing the definition of the city growth directions.*

The battle simulation ends when any attacker battalion enters into the city by climbing a wall (or what remains after its destruction), or when all attacker battalions are defeated by the defenders. In ancient times, battles continue inside the city, but the outcome was somewhat independent of the wall geometry, and thus we do not include it in this paper. In our implementation, a given battle between attacking and defending armies is repeated several times and results are averaged. Also, from simulations where the attackers won, we obtain weak points that will guide the castle evolution, i.e. creating new towers, wall segments or a whole new wall wrapping the current city. As a last kind of castle evolution step, corresponding to the latest periods of such defensive measures, a novel algorithm creates star-shaped fortresses following Vauban's main design guidelines [Duf85, Lep09].

It is important to emphasize that this paper is about the simulation of geometric evolutions of ancient cities, and uses battles as part of that process. Most importantly, it does not aim at implementing a full combat simulator, as done elsewhere [RCCC13]. So, the battles in this paper represent an averaged scenario of the actions happening in a battlefield (e.g. advancing, firing, artillery, etc.), always taking into account the proper time setting. For instance, until the introduction of gunpowder, artillery depended upon mechanical energy to operate (e.g. catapults), and this severely limited the energy of the projectiles. However, aiming at a simulation that is able to reproduce up to the tiniest details is unfeasible, as, in the end, every city, every castle and even every wall segment are unique constructions that depended on a number of factors, such as economical and human resources, to be built. For the same reasons, the simulation of long-term sieges falls outside the scope of this paper, as the siege itself only aims at surrounding the target and blocking the reinforcement, escape of troops, or provision of supplies, but did not imply actual changes in the defensive city elements. Famine, fires and other events also happened during these wars, but again, they did not force a geometric redesign, and thus, fall outside the scope of this paper, as well as the events that follow the fall of a city wall. Another approach other than a pure geometric simulation would be much more historically and analytically based, such as reviewing the history of as many cities/castles as possible over a long period and determining a set of rules and patterns that seem to govern their growth. This would be a significant information/historical analysis rather than a geometrical simulation as we do here, but this approach probably falls outside of computer graphics and the scope of this paper.

4. Simulation Elements

The basic elements participating in our implementation are the city itself and its parts, the walls forming a castle, the battlefield and the armies. In Figure 4, we can see an example of static elements (armies are dynamic). Following the work of Garcia-Dorado *et al.* [GDAB*17], we can classify the initial input parameters of our system into three sets $\Omega = \{\omega_t, \omega_p, \omega_b\}$:

- ω_t represents the terrain characteristics for each grid cell. For example, a hill has a certain slope, which is accounted for by the movement penalty factor. These distributions can be defined by an interactive drawing tool (see Figure 4), be loaded from GIS

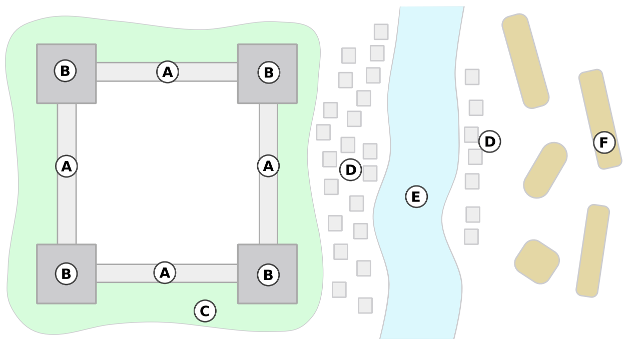


Figure 4: *Static simulation elements in the simulation landscape: walls (A), towers (B), moat (C), houses (D), rivers (E) and trenches (F).*

data or other model databases, or be generated through procedural modelling techniques. It is defined in Section 4.1.

- ω_p refers to constructive procedural parameters, described in Section 4.2, which define the city and wall geometry (e.g. types of towers, bastions, moat, etc.)
- ω_b are the initial conditions of the battle simulation. These conditions define the initial values for each army deployed in the battlefield. These conditions can be defined explicitly, procedurally or via observations. Our system uses historical data for each situation (provided by historians and art historians in our university). To generate the initial conditions procedurally, we define a safety distance equal to the maximum range a defender archer (or cannon, if available) can shoot, and randomly position all our attacking armies at that distance. See Section 4.3.

In our implementation, any of these parameters can be varied both in space and time. As an example (already shown in our forward design examples), the type of tower can vary for different years, as well as the introduction of defensive bastions to repel artillery fire. In our simulations, we set the values initially and then execute the battle simulations.

4.1. Battlefield

The battlefield ω_t is a regular grid of cells c_{ij} where attacker battalions are deployed; it controls the troops' movements. Each grid cell has a height h_{ij} and a movement penalty p_{ij}, which depend on current terrain conditions, e.g. dry, mud and flooded. When a battalion enters a cell, depending on the movement penalty and its height, it is slowed down proportionally to the penalty factor p_{ij}. In addition, there are other battlefield elements:

- **Rivers**: They are similar to moats, but a river does not have to wrap the castle. As they are wider and deeper than moats, they infer a stronger movement penalty to any unit that attempts to traverse them. They can be overcome the same way as moats, see below.
- **Trenches**: A trench is any kind of terrain element that gives an extra defensive factor to attack battalions. When an attack archer unit moves to its target, it first searches for the closest trench, moving there to get some defensive factor against castle shots. If there are reachable targets from the trench, the archer unit remains on this cell. Otherwise, it will advance looking for other closer trenches leading to reachable targets. Trenches are created randomly along the battlefield for each battle simulation.
- **Houses**: Cities can grow outside its walls. If that happens, a house external to the city wall offers to attackers an extra defensive factor, like a trench. Later on, houses will also affect wall expansion (see Section 6).

Finally, as we can see, the environment (terrain, water bodies) is fully considered: Figure 11 shows an automatic placement of walls taking into account the river, as it would happen in reality. Areas to enclose are partitioned into disjoint sets, each walled independently. Also, different kinds of terrain are considered by the movement penalty factor p_{ij}, accounting for slopes, different consistencies and any other terrain aspect. All these elements are defined by the user though the input file, but nothing prevents a future implementation to allow the creation of some of them on the fly (e.g. trenches), as is done by covering moats during a siege, see below.

It is important to notice that the whole battlefield is nothing more than a terrain with some extra features that could be modelled with standard procedural techniques [EMP*03, STN16], thus alleviating the burden of defining them manually. For instance, rivers could be created either manually, or with growth or erosion techniques [EVC*15], while trenches could be simply defined parallel and at a distance to the closest castle wall. In any case, it should be noted that, as usual with procedural models, the more automation is put in the process, the less control the user has on the system evolution, which in this case could be a problem if historical accuracy is an issue. In any case, the full exploration of the automatic battlefield definition possibilities is left as future work.

4.2. Castle

In our implementation, a castle or walled city is defined by its external defensive elements: ω_p

- **Walls**: The castle wall is the main defensive system, and defines the city shape. The castle wall is composed of wall segments $w_i \in \omega_p$, forming a closed poly line that defines it. Each wall segment w_i has a top gateway where archers are deployed to shoot attackers, while they are protected by merlons. For the attacking army, walls are the main target. Walls can be climbed by the infantry or destroyed by artillery. Therefore, wall heights and thicknesses are key features in battle simulations.
- **Towers**: A tower joins two walls (i.e. at some vertices of the poly line). They can be square or rounded, depending on which century they were created. Defence archers and artillery battalions can be deployed on a tower. A tower is a strong point, and therefore, it is never targeted to be climbed or destroyed in battle simulations.
- **Bastions**: A bastion is the latest in tower evolution, since most of them were constructed over original towers. A bastion goal was to cover all 'blind' zones around a castle that were generated by square or rounded towers (see Figure 5).
- **Moat**: If it is used, a moat surrounds the entire castle with the objective of making attackers' movements more difficult, and thus, perilous. In our implementation, moats are optional. They

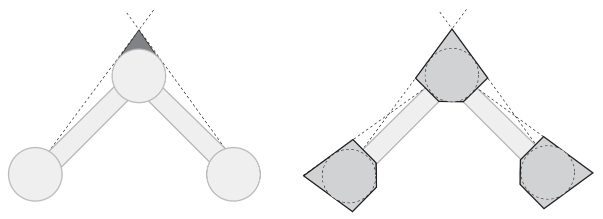


Figure 5: *Left: in dark grey, the blind zone produced by rounded towers. Right: a bastion removes all blind spots.*

are defined by width and depth, and may or may not contain water.

4.3. Armies

As mentioned in Section 3, there are two kinds of armies in ω_b: defenders and attackers. Each army is structured in battalions, in turn, containing a set of units of different types: infantry, archers, artillery and siege towers for the attacking army, and archers and artillery for the defending army. A unit may contain a variable number of soldiers of a given type. For instance, artillery units may contain a single catapult or cannon. Each type of battalion simulates a specific type of action, and the units used in battle depend on the historic context.

- **Infantry**: They advance on the battlefield to climb the closest wall (see Section 5.4). They are fast, defenceless and cannot shoot.
- **Archers**: Although throughout this paper, we consistently use the word *archers*, we are actually referring to long range shooters, using weapons such as arches, crossbows or muskets (introduced during the 15th century). On the battlefield, archers advance until a defender battalion is in range for shooting. In the castle, archers stay at their positions searching for the closest enemy in range. The target acquisition is performed randomly but weighted inversely with distance. After every shot, archers need a given recharging time (see Section 5.3). Archers on the battlefield are defenceless, but strongly protected on the castle due to wall and tower merlons.
- **Artillery**: According to each period, it is formed by devices such as catapults (less energetic shots) or cannons (more destructive shots). In general, artillery has a very restricted movement and is very slow recharging between shots. Also, artillery has high protection, either in the battlefield or in the castle. The goal of attack artillery is to destroy castle walls to help the infantry climbing into the castle (see Section 5.5). The goal of defensive artillery is to shoot attacker battalions, selecting their targets in the same way than defensive archers, with a higher preference for attack artillery.
- **Siege towers**: Siege towers are built on the battlefield to adapt their heights to the castle walls. Therefore, they waste battle simulation time, depending on the required height, until they are ready. Then they choose a path to the closest castle wall, and they advance slowly on the battlefield towards that point. Siege towers are populated by archers, so when castle battalions are reachable from the siege tower, they shoot such as normal archers, but they are protected by the tower structure. Obstacles, such as moats, are removed by a special, heavily protected unit called *turtle*, which cannot shoot. The turtle advances along the siege tower path until it reaches a moat, and then it begins to remove it (e.g. with a bridge or filling it with earth), requiring a number of turns proportional to the moat characteristics.

5. Battle Simulations

The battle simulation goal is to find the castle weaknesses for improvement of its defensive capabilities. In this work, we do not consider battles inside the castle itself, so our simulations stop when a wall is climbed. This might appear to be a drastic decision, as it basically ends the simulation. We realized that the battle could have continued inside the city and the attacking army could still be defeated, but this would not have most likely changed the geometrical aspects, which are the main objective of our simulation. Thus, we stop when all necessary geometric information is gathered. Also, this is surely not an unreasonable assumption, as this was exactly what happened at several sieges in the past, such as the Siege of Lisbon, in October 24th, 1147, when Moorish rulers surrended the crusader's siege. Our battle simulation also ends in case the whole attacker army is defeated.

Whenever the castle is defeated, a weak point is generated where it was climbed. As already explained, a given battle is repeated several times and results are 'averaged'. The outcome of such simulations is a set of Gaussian-like probability distributions, from which the breach points with the highest probability are selected. Each of these repetitions uses randomly selected parameters, with different outcomes. The selected vulnerabilities are the result of several runs (from one run in the video, for demonstration purposes, up to 50 runs in our final experiments). Finally, after a user-defined number of simulations, we obtain a set of weak points on the walls that will be used to improve the castle, as presented in Section 6. The simulation is ruled by a turn-based system, where each battalion decides its following action, described in the next subsections.

5.1. Artificial intelligence

Each agent in our system uses a *behaviour tree* [CD19] (also known as a decision tree) to describe changes between a finite set of tasks in a modular fashion. A behaviour tree is a mathematical model of plan execution, often used in video games and robotics. We choose them because of their ability to create very complex tasks composed of simple tasks, without worrying how the simple tasks are implemented. For detailed definitions, as well as example code, we recommend the interested reader to consult any of the excellent books on the topic [Sch04, MF09, DaG17]. In particular, the work by Champandard and Dunstan [CD19] is highly recommendable as a general, but excellent introduction to the topic.

A behaviour tree is usually represented as a directed tree in which the nodes can be either the root, the internal control flow nodes or the execution nodes, which are the leaves of the tree. The execution of a behaviour tree starts from the root which sends *ticks* with a certain frequency to its children. A tick is an enabling signal that allows the execution of a child. When the execution of a node in the behaviour tree is allowed, it returns to the parent a status which can be: *running*, if its execution has not finished yet; *success*, if it has achieved its goal; or *failure*, otherwise. Our implementation is based on the Python library *behavior3py* (https://github.com/behavior3/behavior3py) by Marzinotto *et al.* [MCSg14], which uses a blackboard as a shared memory to keep local and/or general information among agents. This blackboard is used at the behaviour tree node level, the agent level and the whole army level, to provide global goals to all the agents simultaneously.

5.2. Movement

Movement actions are applied to the attacking infantry, archer and siege tower units. As explained in Section 4.1, battalions move cell by cell on the battlefield. The number of cells that a battalion can advance depends on the battalion speed factor and the movement penalty p_{ij} for the given cell. Each battalion follows its own path to its target (see Section 4.3), avoiding obstacles such as other battalions or castle structural objects, and recalculates its path if the target is no longer available (e.g. a unit that has been destroyed). In our current implementation, the path is computed trying to minimize deviations from the direction to the target position, controlled by the IA module. However, any other criterion could be used, such as the well-known A* algorithm [Sch04].

5.3. Shooting

The shooting action is applied to archers, artillery and siege towers. This action consists of the following steps:

- Search for the set of available targets and choose one randomly, weighted by distance. Available targets are those that are in attack range. The attack range depends on the maximum shooting distance and the angular shooting range. The defender archers have an angular shooting range restriction related by the merlons coverage. The attacker archers have no angular restriction. Artillery also has a reduced angular range because it cannot easily move to aim the target, in the castle or in the battlefield.
- Execute the shooting. To calculate the shooting vector (see Figure 6), the battalion precision skill factor is used with the maximum shooting distance to get a cosine distribution over the shooting ray. Then a direction is sampled from this distribution, resulting on a final hit or miss on the target. Wind could also be included to affect the precision of shots, but we could also assume part of precision for the shooter.

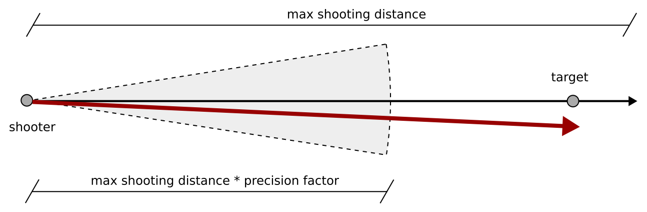


Figure 6: *Shooting method using the precision factor over the maximum shooting distance: each shooter uses a cosine distribution around the direction towards its selected target, and always within its maximum shooting distance.*

- If the target is hit, its defensive factor is compared by the shooting power factor, deciding if the target was killed or not.
- Wait to reload the weapon, defined as a number of turns.

To simulate artillery shots, we have to take into account that real artillery bullets do not explode when they impact, as they simply were iron balls (for cannons), sometimes filled with shrapnel, or boulders (for catapults), shot directly at soldiers. The actual effect of these projectiles then was to 'drill a hole' in the approaching units, killing everything in their path from the moment they reached the soldier's level until they stopped. Because the soldier positions inside the battalion are not considered individually, a sample algorithm is used to know how many soldiers are killed when a bullet hits a battalion. For this, we know the battalion cell location $b_c \in \omega_p$ in the battlefield, its size in soldiers b_s and each soldier standard volume V_s. Then, a ratio r_k is calculated considering the battlefield cell cube volume V_c and the volume associated with the projectile path V_p:

$$
\begin{aligned}
r_k &= \frac{b_s V_s}{V_c}, \\
n_k V_s &= r_k V_p, \\
n_k &= \frac{b_s V_p}{V_c},
\end{aligned}
$$

where n_k is the number of killed soldiers. The path volume V_p is calculated using a cylindrical profile of radius equal to the bullet diameter (or linearly increasing if the bullet is made of shrapnel). This profile is approximated by a set of cylindrical (or conical) cell-sized segments with length equal to the length that passes through the cell volume, until it reaches the ground. In that latter case, the cylinder length will be the distance between the cell boundary and the intersection with the ground.

5.4. Climbing walls

When an infantry battalion reaches a wall, it starts climbing it. Climbing is simulated by a virtual ladder where each soldier steps upwards at each turn. Climbers can be killed by any defender archer with enough angular range from its position. In addition, the closest defender archers to the ladder are gathered into a new kind of unit named *thrower*. The *thrower* is a temporary battalion that throws things, such as rocks or boiling pitch (resin), over the climbers. This throwing action is performed as a special shooting in the ground direction. The throwing hits the first climbers on the ladder, from top to bottom, killing them, up to a user-defined threshold. The thrower reload time is long and it is related to the number of units in the defending battalion. If any of these units is killed, reload time increases. If a climbing battalion reaches the top of the wall (representing the start of the fight over the walls), the simulation ends, storing the position of the ladder as a wall's weak point.

5.5. Breaking walls

5.5.1. *Stone walls*

When a battle simulation starts, a grid is generated for each wall. When artillery is aiming at a wall, it randomly chooses between one of the top wall tiles, weighted by the shoot impact factor. This factor is calculated by $e * \cos(\alpha)$, where e is the shot energy and

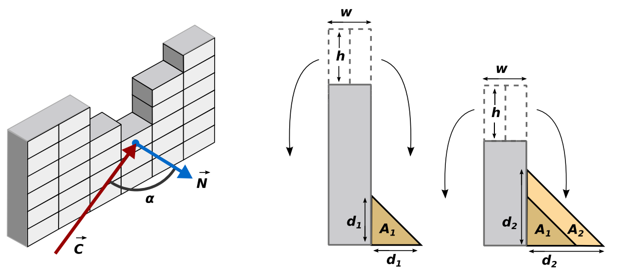


Figure 7: *Wall breaking by artillery shots on wall grid (left). When a grid cell is broken, its material falls down creating a slope (centre and right), where w is the wall thickness and h is the tile height in the cross sections.*

α is the angle between the shot vector $\overrightarrow{C}$ and wall normal $\overrightarrow{N}$ (see Figure 7, right). The impact value reduces the wall resistance at the corresponding grid cells proportionally to the cosines of the incident angle and the wall normal at that point. When a cell resistance is 0, it falls and is converted into rubbles, being accumulated at the bottom of the wall, as shown on left of Figure 7, where we can see that rubbles are simulated as a slope, which is represented as a wedge with a volume equal to the volume of the fallen rubbles. Afterwards, a small percentage of the fallen rubbles are redistributed to the columns on the sides (20% in our examples).

When the height d_n of the accumulated rubbles is equal to the current wall height, artillery discards the lower part of the wall as target, and a gateway through the wall is created. The attacker movement over rubbles receives a penalty depending on the slope angle and height. When an infantry battalion reaches a wall with a gateway, they advance through rubbles to reach it, finishing the simulation if they are not killed by the castle archers.

5.5.2. *Wodden walls*

From the 2nd to the 12th century, in the majority of historical settlements, just a wooden wall was used, except in the particular case where walls were built by the Romans in previous centuries. Wooden walls can be more easily destroyed by fire but also rebuilt faster, so we implemented them as a simple grid where each cell has a strength parameter, which is gradually (linearly) reduced by enemy attacks, e.g. by fire. When the strength reaches 0, we consider this element ceases to exist, no longer stopping any attack.

6. Geometric Evolution

When the city grows or when a weak point is detected after a simulation battle, the castle evolves by creating new walls and towers. These kinds of evolutions are explained in the next sections. Finally, the latest kind of evolution in walled defensive structures, the star-fortress, is explained.

6.1. Castle growth evolution

6.1.1. *Castle wall elements*

One of the main features that define the weakness of a wall is its length. Short walls can be easily defended due to the closer towers, as towers improve the angular attack range of archer defenders.

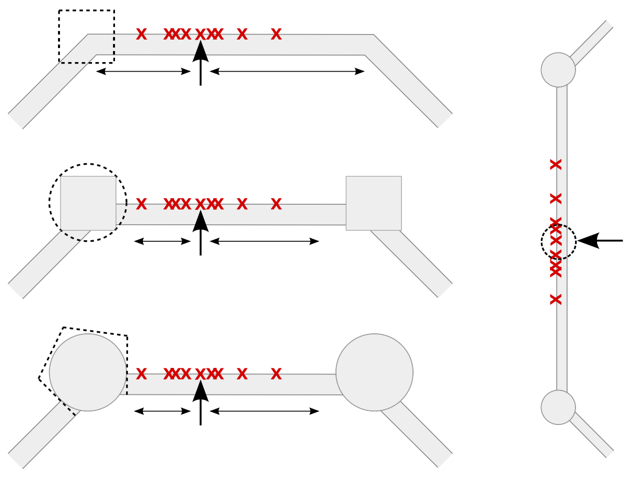


Figure 8: *Types of curtain wall evolutions from the weakpoints (marked in red). The weakpoints are clustered to get the closer tower or the wall splitting point. Then a new tower is constructed (dotted lines).*

Therefore, long walls should be split with new towers. The weak points resulting from the battle simulations give information that is used to split the walls (see below), and then to create a new tower, or bastion. The construction of new towers was relatively inexpensive, but bastions required the addition of larger structures, see Figure 5. A tower can be created as squared or rounded, depending on the current simulation era, and the same for bastions. For each type of tower and bastion, there is an associated historical range of years (and locations) when they were used. However, there were overlaps in time between the different usage periods, so we use a weighted random selection within these overlapping periods. In this case, the weights are proportional to the temporal distance between the sampling date and the overlapping period boundaries. For non-overlapping periods, a direct selection is performed.

There are three conditions to take into account when a wall needs to be improved with a tower or bastion:

- If the wall segment has any of its endpoints without tower, the closest segment end (without tower) is selected to create a new tower or bastion (see Figure 8, top left). Bastions represent the last kind of evolution, before the use of star forts.
- If the weakest point is close to a tower, the tower must evolve to its next stage, that is, from squared to rounded, and from rounded to bastion (see Figure 8, middle and bottom left).
- Finally, if the weakest point is far from any tower, and if the wall is long enough to be split, a new tower or bastion is created (see Figure 8, right).

If no wall can be evolved with one of these conditions, a city growth evolution is performed (see next section), requiring further expansions of the wall in the future to protect these new structures.

6.1.2. *Whole castle wall*

As mentioned above, our objective is to simulate the evolution of ancient cities, and the way that affected their shapes. Given that any

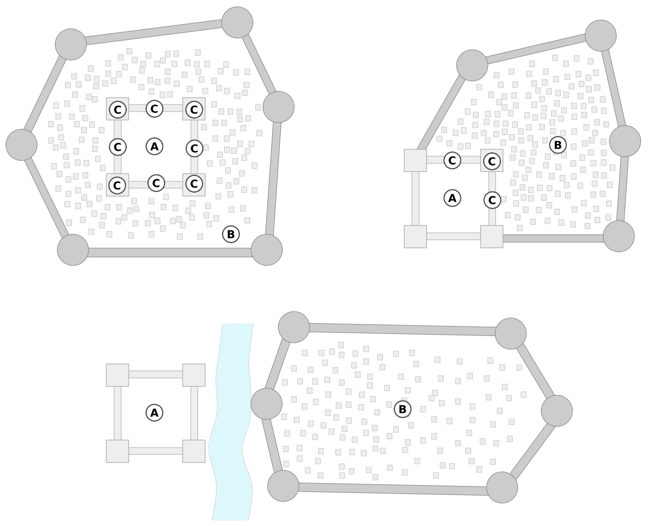


Figure 9: *Three examples of city growth. For each one, we see the initial castle (areas with A labels), the city expansion (B labelled areas) and the walls and towers rendered useless after the evolution (structures labelled with C).*

city growth simulation algorithm can be used [WMWG09, EBP*12, BWK14], and that this evolution is performed in peaceful times (i.e. does not depend on any strategic planning), we decided to use a simple city growth algorithm directed by house sampling from a set of user-defined evolution patterns, such as a regular growth around a prescribed direction (e.g. with a cosine distribution), or the restriction imposed in ancient times from houses to be built in areas that would weaken the defence, such as very close to the walls. In general, an urban evolution pattern is defined as an origin zone, a growth direction, a building creation rate per year and a time range. When the method requires a full city evolution (see Section 6.1.1), new wall segments are created. The new walls wrap the houses created from the last city evolution, completely enclosing them and following the new constructed area convex hull, plus a small offset outwards. There are three cases to take into account for the new wall:

- The new wall surrounds completely the old one: In this case, the old wall becomes useless for battle, so the new one replaces it and is the only target for the attackers (see top left of Figure 9).
- The new wall intersects the old one: A 2D Boolean union is used to join both walls (old and new), creating the final one (see top right of Figure 9). The towers of the new wall segments are created taking into account the current year of the simulation. The towers related with the old wall segments are reused, and are not evolved. A particular case appears when a new tower is going to be created too close to an old one. In this case, the new tower is not considered, the old one is upgraded (if required), and the new castle shape is updated to connect to the old tower. In addition, the method avoids too short or too large walls, removing wall vertices or splitting walls, respectively.
- The new walls do not intersect with the old ones. This happens when there are new buildings far from the main structure, or when the city growth is discontinued by a river (see Figure 9, bottom). In this case, a new independent walled structure is created surrounding the isolated houses.

In the first two cases, the obsolete defensive structures are ignored during battle, while in the third case, all new structures are now potential targets, as well as the old ones.

6.2. Star fort

The latest type of castle evolution was the star fort, bastion fort or trace italienne, which is a fortification that evolved firstly in the mid-15th century in Italy, after the apparition of gunpowder, when cannons were perfected enough to start dominating the battlefield. In general, it was mostly used for castles, but also for cities like Tvrđa (Croatia), in 1861; Geneva (Switzerland), in 1841; and Palmanova (Italy) and Naarden (Netherlands), still existing nowadays. This kind of fortification defines a star-shaped construction around the castle or city to improve its defences against the combined attack of cannonballs and climbing soldiers, see Figure 10 (bottom). It is important to mention that star forts represent the last evolution step, which did not evolve any further before the decline of walls as defensive elements. Please, refer to the Appendix for details of its geometric construction.

6.3. 3D geometry generation

The last step in our pipeline is the generation of 3D geometry. In our current implementation, this is done by a process of simple extrusion and model replacement. The first stage, once the simulation is complete and the floorplan is ready, is to extrude its main components (i.e. walls, towers, houses) to have a volumetric model. Then, in a second stage, we replace the volumetric models by detailed 3D counterparts, most of them taken from an artist-created library (The Castle Creator library, from Daz3D). This replacement was implemented with standard procedural techniques, such as the CGA language developed by Müller *et al.* [MWH*06] and implemented in the commercial product CityEngine, which amount to a simple repetitive subdivision along the walls, inserting for each subdivided segment a wall piece from the library, or replacing the tower geometry by its artistic counterpart.

7. Battlefield Design

Up to now we have described our forward modelling tool, where users want to design an ancient city model and simulate realistic battles during different time periods. However, our system also supports inverse design, and it is able to change the initial battle conditions or the characteristics of the battlefield to cause the battle to evolve as desired. Uses of these design options are shown in the results section (Section 8).

7.1. Inverse design

As mentioned, our system also builds upon the concept of inverse procedural modelling [TLL*11, BWSK12, SPK*14, GDAB*17], providing a novel inverse modelling tool for designing historical battles (e.g. for movies or video games). Given a procedurally generated model, our system finds out how to alter the model or the initial conditions so as to produce a user-specified battle outcome. As battles are very complex to simulate, it is quite difficult to predict and/or control their outcome. Therefore, we used an optimization-based system to explore the search space and find a suitable solution that exhibits the desired battle outcome. We have tested Metropolis-Hastings and simulated annealing optimizers, but we obtained the best results with a variant of the global optimization algorithm by Mas *et al.* [MMP18]. See below.

As in the work of Garcia-Dorado *et al.* [GDAB*17], the optimization mode for the inverse calculations can be any of:

- *Error optimization*, where the system tries to minimize an error function $E(*)$ that describes the desired outcome. We used a simple function given by:

$$E(p, A, D) = ||p - p^{\text{target}}|| \\ +\Omega\left(\sum_{a\in A}\omega_a|I_a - F_a| + \sum_{d\in D}\omega_d|I_d - F_d|\right),$$

  where p is the position where the castle was breached, $||x||$ is the length of vector x, A is the set of units in the attacker army, D is the same for the defender army, $\omega_{\{a|d\}}$ are user provided weights for the loss of an attacker unit a or defender unit d and Ω is a global weighting factor to control the overall influence of the losses term.
- *Cost minimization*, where the user defines an objective value η for a function $E(*)$ and defines a cost function $C(*)$ to minimize. This might be of interest, e.g. when a game designer wants to perform the minimum changes possible to a good enough simulation, while controlling a constructive variable. In our implementation, we used a simple conditional to decide whether the optimization should optimize only $E(*)$, if its value is above the user-provided threshold th; or $\lambda C(*)$, if $E(*) <= th$, where λ is a weighting constant chosen to guarantee that $\lambda C(*) <= th$. For instance, as a simple test, we verified it with the same inputs as the above error optimization function, setting $E(p) = ||p - p^{\text{target}}||$ and $C(A, D) = \sum_{a\in A}\omega_a|I_a - F_a| + \sum_{d\in D}\omega_d|I_d - F_d|$, with $\lambda = th/(\sum_{a\in A}\omega_a I_a + \sum_{d\in D}\omega_d I_d)$. The results of this particular test are similar to the ones we got with the weighted combination, above and are omitted in this presentation.
- *Constrained optimization*, where constraints are used to define a problem, something frequently needed for the other modes. In our implementation, constraints are simply introduced as penalty terms in the error function $E(*)$, each multiplied by a user-provided weighting factor.

Mas *et al.* [MMP18] presented a global optimization algorithm specifically tailored for inverse design problems in the domain of reflector design, but it can be used for more general problems, as we did here with a high degree of success. To perform the optimization, the method they propose is based on constructing a binary tree in which each tree node represents a battle where the objective function must be evaluated. At each construction step of the tree, a previously created node is chosen stochastically by using heuristics based on the evaluation of the node and the up-to-date statistical information on the surrounding tree nodes. Once chosen, the selected node is replaced by two new child nodes. These nodes contain the same parameter range as their parent but one, which is split into two subranges. For each new node, we use a heuristic, based on its ancestors, to choose which parameter range to split. Since the algorithm evaluates the nodes after they are created, and because it uses a greedy breadth-first search algorithm, the full tree structure is not needed. The process stops when a node evaluation result is below a user-determined termination threshold. However, when the chosen node is close enough to a minimum, or when the parameter space size is smaller than a user-defined threshold, a local optimization method (i.e. Hooke & Jeeves) is used to converge in a rapid manner. In this case, the chosen node becomes a tree leaf. If the minimum found by the local optimization process is below the user-determined termination threshold, the process stops. Otherwise, it continues by choosing new tree nodes. This optimization method shows a convergence to a solution in fewer steps than most other classic optimization methods, and also avoids many local minima. It must be noted, though, that this method is not able to guarantee finding a global minimum, and it finds the first minimum that is below a user-defined threshold. For more details, refer to the paper by Mas *et al.* [MMP18].

8. Results

The goal of this section is to evaluate plausibility by comparing to real environments, and modelling expressiveness by demonstrating the simulation of time-dependent phenomena. For the plausibility, we conducted tests to evaluate the similarity of our simulations to real data from three different scenarios: the city of Carcassone (France), which still exists nowadays as a walled city that can be effectively compared to our results; the city of Gerunda (Spain), which is one of the most well-documented cities in respect to its historical evolution; and the Ciutadella in Barcelona, a star-shaped fortress that only remains as descriptions and drawings in old documents. For the modelling expressiveness, we tested how a castle adapts to its city growth, changing its shape along time without considering any battle simulation. Then, we run many battle simulations on different examples to see how a castle adapts its defences to different kinds of attacks. All the results, both synthetic and real, have been validated by historians who helped fine-tune the different parameters involved in the simulations.

8.1. City growth only

To check city growth, we have used the fictitious example shown in Figure 10. Each figure shows the result of surrounding the city with a new wall. From the simple castle (A) the city evolves (to the North-West), wrapping it with a new wall (B) that joins to the old one, creating new rounded towers at the new wall segment endpoints and at the join points. After this, the city grows again in the same direction, forcing the expansion into a new set of walls (C). This time, the new towers are squared and rounded, because the evolution time falls in the range shared between both types of towers. Then, the city grows again in the opposite direction, creating the new wall shown in (D), with rounded towers and bastions. In the last step, the city has been wrapped by a star fort (E), whose generation also transforms the external rounded towers into bastions. Note also that there are bastion pairs without a ravelin or bastions without lunettes, mainly because there is not enough available space, or because the angles between bastions would create self-intersecting or twisted ravelins or lunettes.

8.2. Gerunda (1050–1400 CE)

Another example of evolution without battles is shown in Figure 11. This simulation represents the evolution of the ancient city of Gerunda, based on the real historical facts in the range from 1050 to 1400 CE. In this case, input data are the ancient roman castle shape and actual wall measures (e.g. wall widths and tower dimensions), and three vector patterns for the city growth. From the first step with the roman castle (A), the city started growing at the sides, following the river bounds (B). This happened in a period of 10–20 years around 1376 CE. Note that the original Roman castle did not have towers for each wall vertex, mainly because they were not common at this era. Then the city grew to the West (bottom in the figure), to the other side of the river, generating a second structure separated from the original one (C), at year 1386 CE. The bottom image is the real map of Gerunda medieval city in 1386 CE, included for comparison.

8.3. Battle simulation

In Figure 12, there is a simple example of battle simulation results. From an initial city that grows in all directions and a surrounding castle, the battle algorithm simulates the average result of many battles from all directions simultaneously. Table 1 presents a summary for this example. The simulation has been structured in waves, where each wave is a set of battle simulations. Each wave is executed at a specific year that is used to choose the corresponding tower upgrade. Also, in the table, there is a relation of the different units used for each simulated battle in each wave. The army size increases for each wave, simulating the requirement of more battalions to defend and attack a larger castle. In addition, we have used siege towers for the first two waves, and artillery for the last one, simulating battles in those eras where they were common. When attackers are close to the middle of long walls, they can be attacked by many defenders along the wall and from the adjacent towers. Even if the towers are far from each other, there are enough defenders to kill many of them. Observe that attackers very close to any tower have the advantage of being protected by the blind spots of the towers themselves. Therefore, the tower upgrades give better results than wall splitting.

8.4. Gerunda (5th–12th centuries)

Figure 13 shows the simulations based on many real battles against the city of Gerunda between the 5th and 12th centuries. The simulation starts at the evolution point shown in Figure 11. The initial city shape at the top was attacked from random directions many times. In this case, the city did not grow anymore, so only rounded towers and bastions are created from the battle results. In Table 1, there is a summary of this example, structured in waves. In our simulation, the armies do not change their configurations between waves, only the attack directions. The defending army is composed by 1400 archer units and 200 cannons. The attacking army is composed by 3000 infantry units, 1000 archers and

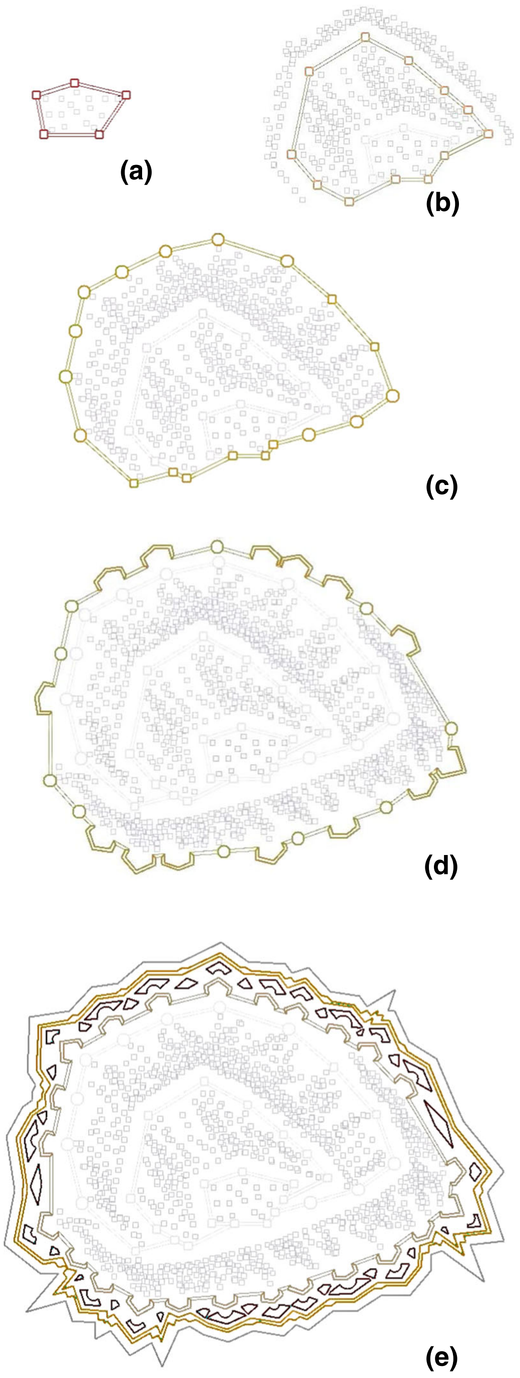


Figure 10: *Example of city evolution with a final star fort. Starting from an initial population, a first wall is built (usually a wooden or stone wall) (A). After some time, the population growth forces the construction of houses outside the wall, which is a security problem from the defensive point of view, as houses can be used a trenches. Thus, a wall expansion is made (B) and the process repeats itself over the years (C–D). Observe that, depending on the years of wall reinforcement, new elements are introduced such as round towers (C), bastions (D) or a star fort (E).*

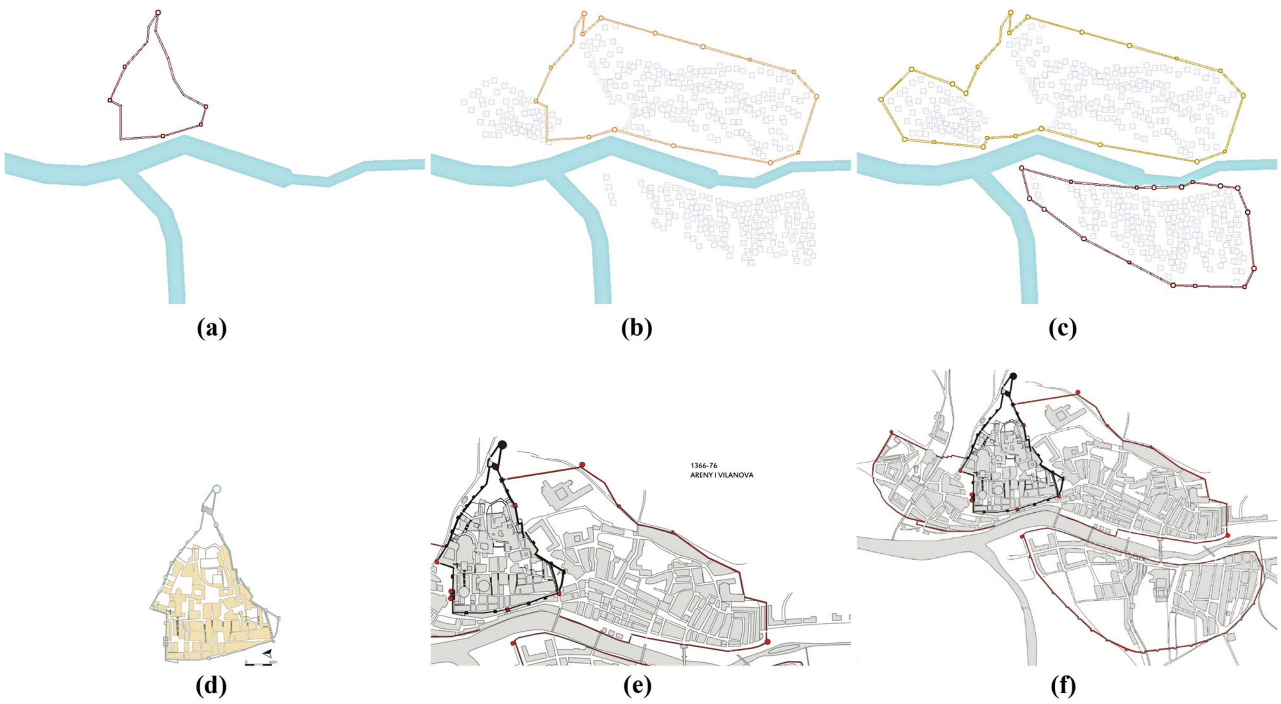


Figure 11: *Simulation of Gerunda medieval city evolution between years 1050 and 1400 (A–C), and the real Gerunda medieval city at year 785 (D), detail of the wall finished at year 1366 (E) and the finished wall at year 1386 (F).*

30 cannons. The results show that the zones closer to the river are more protected than the others, so no tower is evolved to bastion in these zones, which is to be expected since the river is a natural protection. The results also show that the parts of the wall with sharper angles tend to have more bastions, as attackers usually attack earlier the more convex vertices of the original wall. Also, the concave parts are more easily defended by adjacent walls and towers.

Table 1: *Results and units used in evolution simulations. When battles occurred, we use the following acronyms: I (Infantry), A (Archers), C (Artillery), S (Siege towers). Empty cells correspond to Pacific urban expansions and new walls being created.*

				Attackers				Defenders		
Figure	Iter #/Year	battles	results	I	A	C	S	A	C	timing (minutes)
12	1st	40	Three squared towers upgraded to rounded	800	200	—	4	350	—	18
12	2nd	20	One new rounded tower New wall	1200	300	—	4	700	—	17
12	3rd	30	One squared tower upgraded to rounded One rounded tower evolved to bastion New wall	1200	300	20	—	1000	50	29
11(A)	1050		Initial State							
11(B)	1376		Pacific expansion, new wall							
11(C) 13(top)	3/1400	10	Expansion, new wall, rounded towers	1900	1200	—	—	550	—	1
13	1638	10	Four rounded towers evolved to bastion. 6 new bastions.	3700	1000	36	—	1400	200	2
13(middle)	1818		Final State							
16(A)	100		Initial State							111
16(B)	460	80	Expansion, 12 new rounded towers	2800	700	—	—	2600	—	
16	508	10	None	2600	800	—	—	2600	—	7
16	725	10	None	2950	700	—	—	2600	—	7
16	1209	10	None	2700	600	—	—	2600	—	7
16	1226	10	None	3000	750	—	—	2600	—	7
16(C)	1239	30	Seven new rounded towers	2800	700	—	—	2600	—	24
16(D)	1250		Pacific expansion, final state							

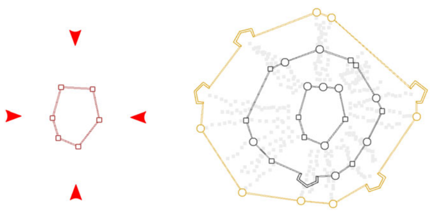


Figure 12: *Battle simulation example. The initial walled city at top is attacked many times from four different directions. After the simulation, the castle is upgraded to include star fort elements (right).*

8.5. Ciutadella fortress (Barcelona)

A more regular example of star-shaped castle, following the Ciutadella fortress in Barcelona, is shown in Figure 14. This castle starts with an inner house distribution and creates a surrounding wall with bastions. Then, the star fort is created automatically, this time without lunettes.

8.6. Carcassonne (100–1250 CE)

The last example is based on the city of Carcassonne (France) (see Figures 16 and 1). The battle simulations are based on real historical data [VlD53]. However, there is a large discrepancy among historians about the actual figures of soldiers [Ram14] in the different armies, and in some cases, the information of the army compositions is, simply put, non-existent. As a consequence, we decided to use, for the attacker army, data from a well-documented later battle (the 1355 battle when Edward the Black Prince sacked Carcassonne) [Ram14], and used these figures plus a 10% variance. For the defensive forces, we used Viollet-le-Duc estimation of the minimum number of soldiers needed to defend Carcassone (1323 [VlD53]), and doubled it. The time range starts at year 100 CE (see Figure 16A) until year 1250 CE (see Figure 16D), and the system is initialized with the bounding wall around the city. For each wave, as there are not important geographical features that would impede movement, the attacking armies came from all directions, many times, creating new towers at each wave.

As before, the wall did not evolve like in other examples because the city did not grow, which allowed focusing on the city defence improvement, creating new towers along the years and battles and delaying the creation of a new surrounding wall until much later. Note that Carcassonne had very high walls, which improved the defence factor in comparison with the other examples. See Table 1 that contains the results of battle simulations for different years. After the first battle, new towers were added, obtaining a castle with a very good defensive factor. On the following battles, the attacking army was defeated in all simulations until the city grew and a new wall was required (see Figure 16C). Then, the city fell in the next simulation, and our algorithm decided to add new towers and a new surrounding wall. The simulations have been performed with attacks from many directions and an army size of 2800 infantry units and 700 archer units. The defending army is composed by 2600 archers. We have not used any artillery for the armies. In Figure 15, we can see the resulting 3D model. For illustrative purposes, we have used

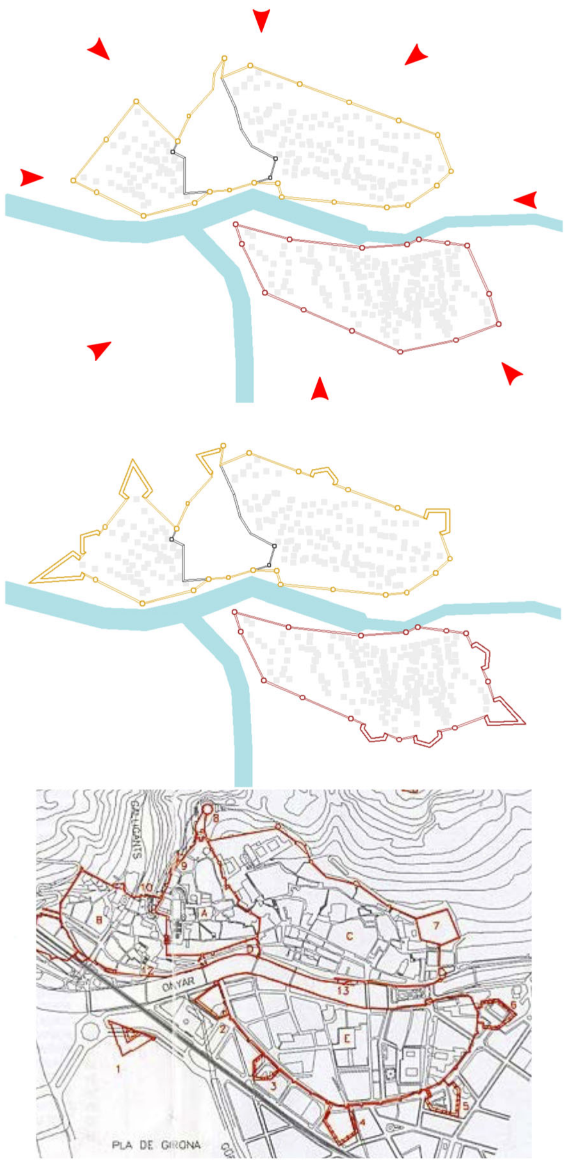


Figure 13: *Gerunda's battle simulation between 5th and 12th centuries, when it was attacked from all directions (top), resulting in an improved, star fort defensive wall (middle). This can be compared with the documented state at year 1638 (bottom).*

a 3D model library to represent the castle objects: walls, towers and houses.

8.7. Consistency

At Figure 17 (left), we can see the result of running multiple times (200 in this figure) the same battle simulation with the same initial parameters. In Figure 17 (right), we can observe the corresponding histogram of distances to an arbitrary reference point (called *target*). The position that shows the higher number of breaches is chosen as the final breaching point for this simulation.

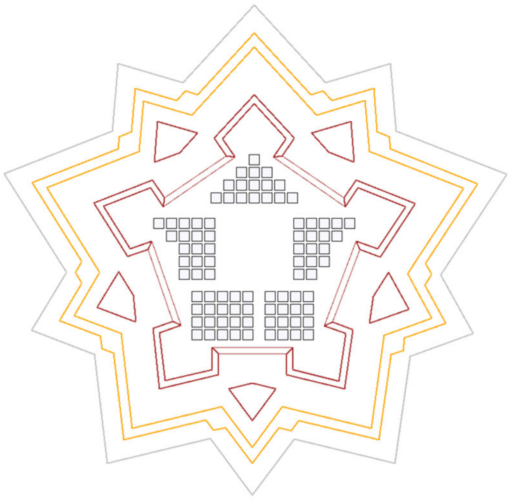
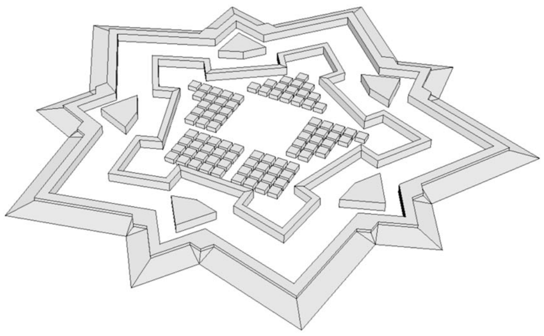
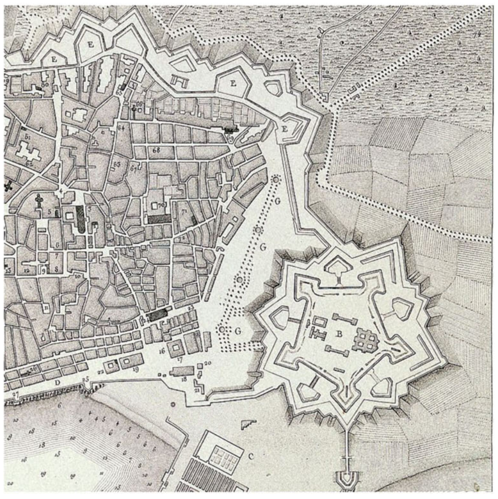




Figure 14: *Simulated star-shaped fortress as a floor plan (top) and 3D view (middle), based on the Ciutadella in Barcelona (bottom) at year 1806.*

8.8. Parameter influence

In Figure 18 (top), we can see how changing the urban expansion settings affects the future evolution of the defensive walls. We should not forget that walls were built to protect the people in the city; thus, household distribution clearly is a key governing parameter of city growth. In Figure 18 (middle), we can see the influence of the temporal events, which can be either a change in the dates of battles, or a change in the range of years when each type of architectonic structure was used. In Figure 18 (bottom), we can see the effect of changing the initial location of the attacking forces (left: uniform, right: from north).

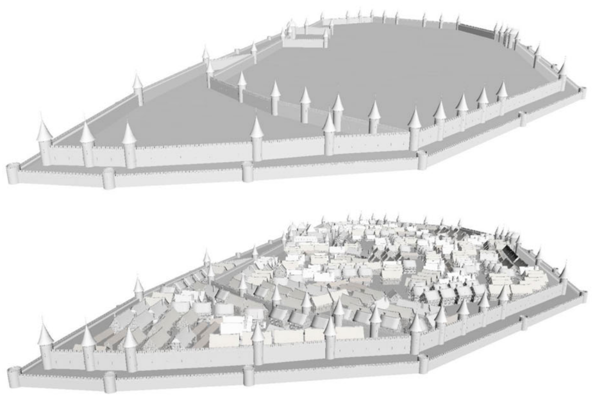
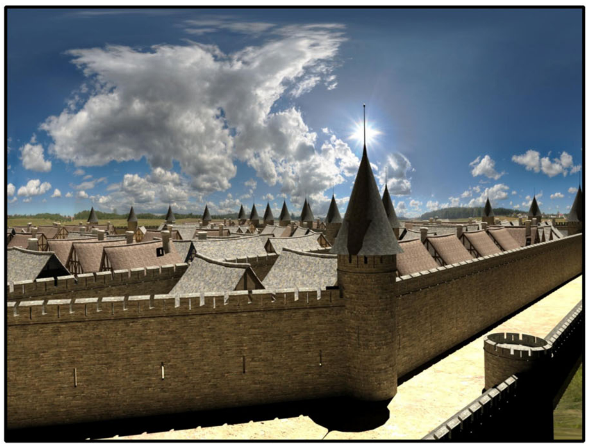



Figure 15: *3D model of the medieval city of Carcassonne obtained from our simulation. Top: walls only. Middle: Including city houses. Bottom: a full rendering of the resulting model. Assets from Daz3D's Castle Creator library.*

8.9. Inverse design

An example of inverse design can be seen at Figure 19, where the events of the siege of Gerunda in 1809 have been recreated [PD50]. At that battle, the city walls were breached at the northeast side, as shown by a red arrow in the figure. The results clearly show a high agreement with the actual starting positions and directions. Each of these starting positions lead to a breach in the defences no farther than 10 m from the actual position. Inverse simulations took a few minutes to solve the whole inverse problem (see below).

8.10. Timings

To analyse the time needed for a successful simulation with our system, we must differentiate three different stages: simulation,

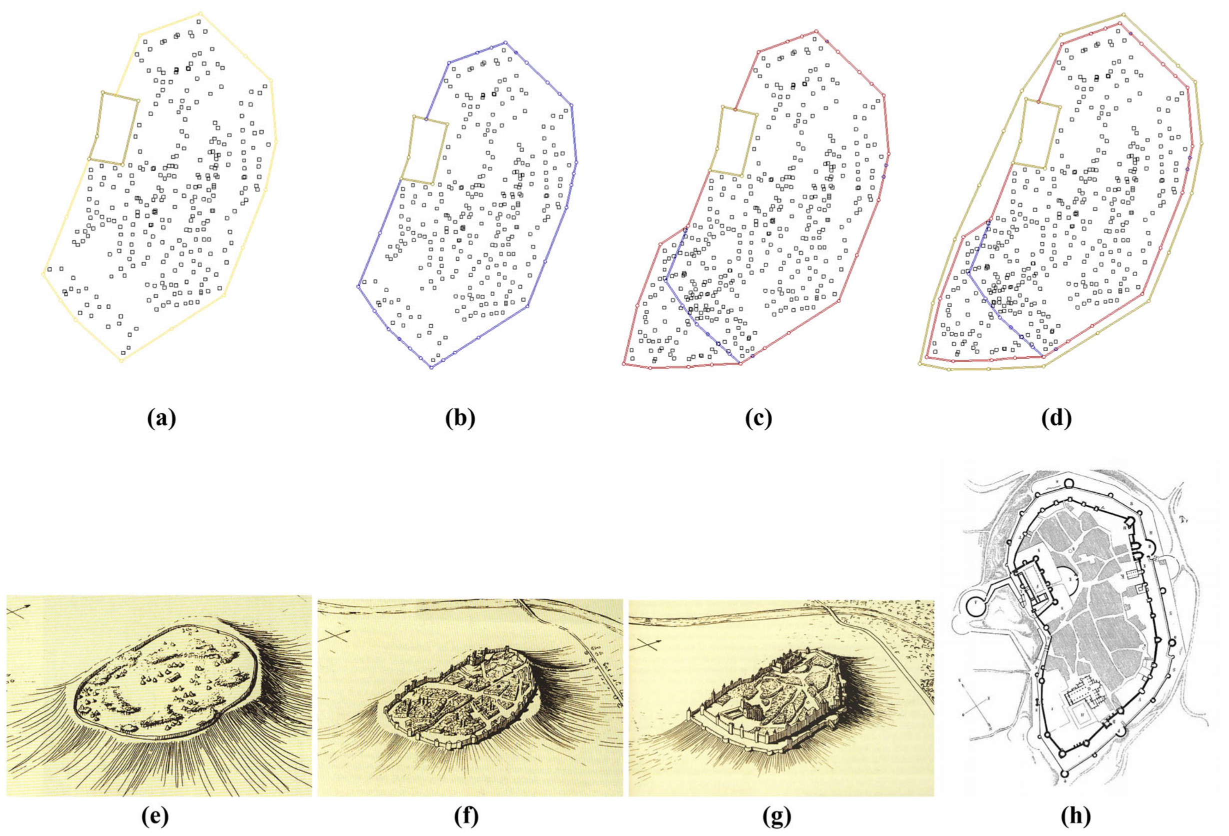


Figure 16: *Top row: Simulation results based on Carcassonne, at years 100 CE (A), 460 CE (B), 1238 CE (C) and 1250 CE (D). Bottom row: drawings of the castle at roughly the same years (E–G) and a map of the current shape of the city (H).*

parameter setting and historical information gathering. The first one, simulation, is by far the fastest one: as a reference, each battle simulation takes, in our implementation, between 8 and 50 s, but this value depends on the size of the armies, and the total time also depends on the times each battle is repeated. The accompanying video is an accurate recording of a working session, and all timings are faithful to the application. Also, we can consider that:

- the system can work without rendering, which is considerably faster, about one-third of the time;
- the same battle is repeated (with randomized parameters) several times, which can be distributed over the available CPU cores;
- our current implementation runs on unoptimized Python code.

The second stage, parameter setting, is where the user sets the parameters of the simulation. As mentioned, the presented application allows the user to interact with the system after each simulation is ended. In general, the whole simulation set-up takes a few hours, even for the most inexpert users. Of course, here is where our inverse simulation may take the role of inferring some missing parameters. For the inverse design part, the example in Figure 19 took 1708 s to complete, performing a total of 144 battle simulations (11 s on average each), of which 100 resulted in an attacker win (i.e. with a valid distance to compute). Summing it all up, our current implementation is not interactive, but it is easy to see that it can become so by introducing some further optimizations.

The final stage, historical information gathering, really depends on the available historical information. Both for Carcassone and Gerunda, there is a considerable amount of information about their historical battles, and, especially in the case of the latter, there is abundant information about the army compositions, locations and movements. However, this information is often scarce, and in some cases, even non-existent. Thus, this stage cannot be allotted a fixed time unless all the information was already gathered. Otherwise, finding accurate information may take an unpredictable amount of time.

9. Discussion

9.1. Usefulness for urban planners

As mentioned in Section 1, having a tool that helps to roughly estimate the past urban changes can be of high interest to urban planners, who have to plan urban interventions, sometimes at large scales.

9.2. Usefulness in movies and games

Artists usually focus, when creating any model, on visual realism only and not on historical plausibility. However, adding some historical rigour would be beneficial not only to designers, but also users themselves may benefit from the added knowledge. The system we

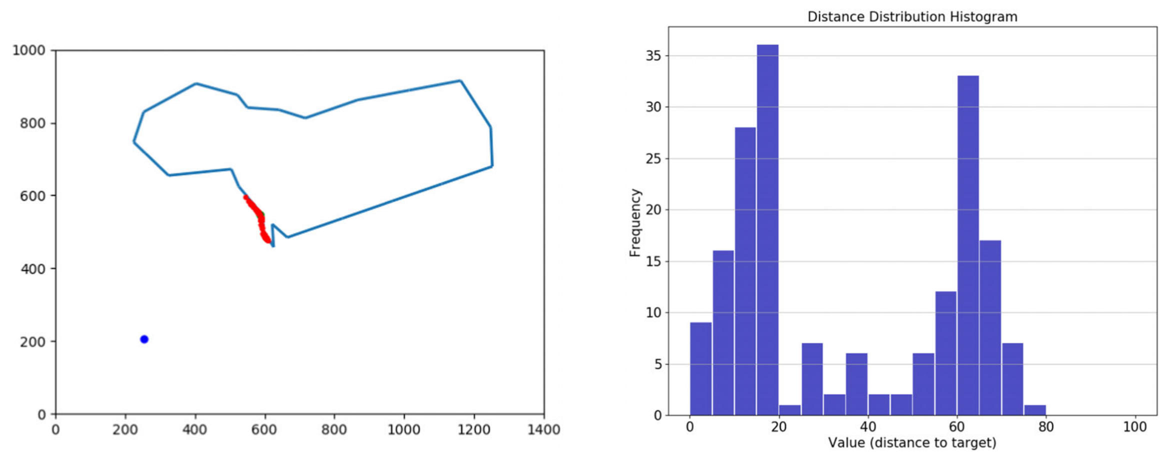


Figure 17: *Consistency of battle simulations. The same battle was repeated 200 times, and the red dots (left) show the positions where the wall was breached. Right: Histogram of the (absolute) distance values between a reference position (called target) and the battle results.*

present in this paper could be used to determine the precise shape that guarantees defensiveness, avoiding obvious historical inaccuracies in a transparent way.

9.3. Comparison to military simulations

In the current state, we do not believe that a formal comparison to existing military simulation software [dMV10, TCTG13, RCBV14, Hog14] is adequate, because the output and goals are too different. On the one hand, to the best of our knowledge, we propose the first geometry-oriented historic simulation system and our main output are two-dimensional floor plans and three-dimensional cities that change over time. Other simulation systems cannot produce this output. On the other hand, we designed a simulation system that focuses on recreating visually plausible urban layout for an ancient city. We did not attempt to tune the simulation to exactly match existing cities nor do we simulate variables that have little visual importance (e.g. long-term sieges). We plan to address this in future work.

9.4. Comparison to video games

As mentioned, there are many video games that share some characteristics with our system, such as the *Age of Empires* series, the *Total War* series and the *Totally Accurate Battle Simulator*. However, there is a crucial difference that cannot be ignored: its purpose. While many video games have an urban or historical setting (or both), their sole objective is to be fun for the player. Instead, our framework is intended to be used as a tool with different application domains: for historians, it may act as a speculative aid to propose historical models of the evolution of any given city. For computer graphics practitioners, it provides a reliable framework to generate plausible 3D models of ancient cities.

9.5. Predictability and controllability

An important aspect to consider is the predictability of our system. In general, the main castle weak points can be identified intuitively, usually being associated with long walls, which are harder to protect. However, the battle outcome is a combination of several factors, including the castle geometry, the battalion units and the battlefield configuration, besides the randomness included in the simulation process. For instance, depending on how the attacker units are deployed and configured, they could intensively attack a non-weak castle point, making it fall in the long term. In fact, in some situations, this is exactly what happened, like in the year 1240 CE when Carcassonne fell because the invading forces concentrated their efforts on the southern tip of the wall (see Figure 16). Our simulations are able to correctly predict this outcome, given an accurate army distribution, from an historic point of view. A tightly related issue is the controllability of the outcomes of the simulation, which strongly depend on the user for the positioning of the attacking forces. An accurate positioning will result in a likely historical outcome, but a wrong input from the historical point of view can result in undesired results. In this sense, our system can be used by historians to pose plausible historical scenarios and observe their outcomes, allowing them to validate their assumptions, thus closing the loop between the simulation and geometric evolution with the needs of scholars doing research in historic events. In any case, at each iteration, the user is completely free to add, remove or modify any event and its controlling parameters in our simulation, like in the example of the battles described above.

9.6. Architectural styles

Another important point is how different architecture styles are taken into account by the system. Although we have implemented Vauban's structures, there exist many other different patterns that appeared in different regions and at different eras in history. For instance, Philippe Auguste and his successors (who also inspired Edward 1st of England) developed a specific castle style in the XIIth century, rediscovering and following the knowledge and expertise of romans. Passive defence (in depth defensive system) that relied on the accumulation of obstacles to protect defenders was progressively transformed into active defences with rounded towers and other structures. As our system simply instantiates a different

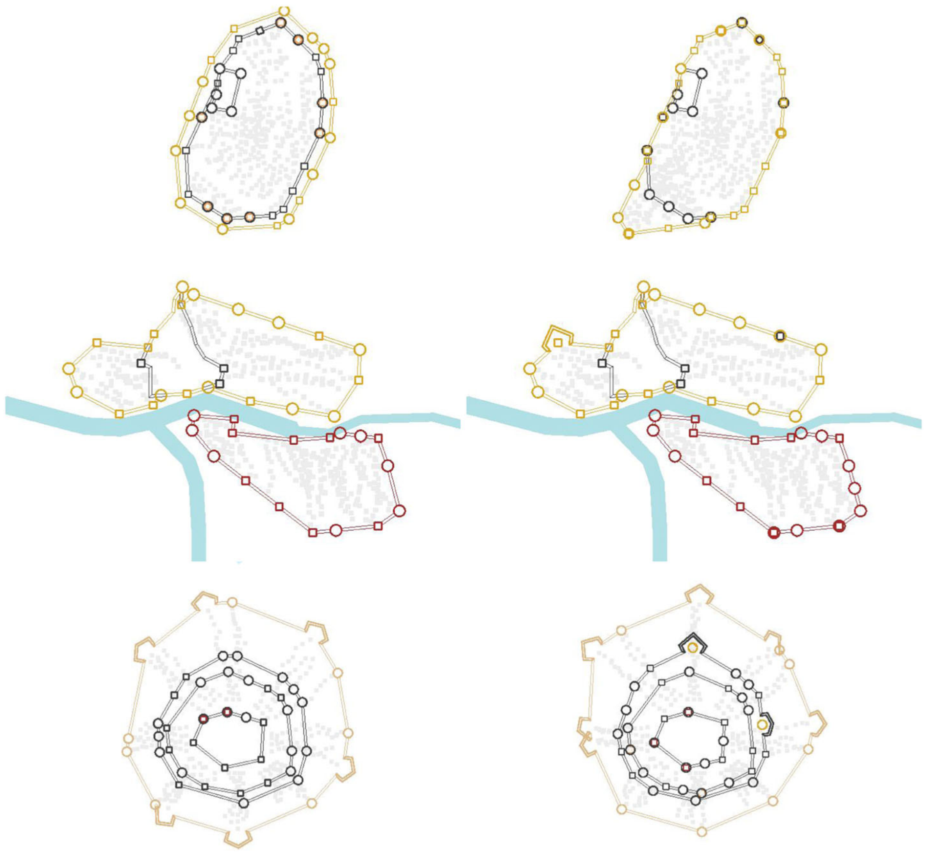


Figure 18: *Effect of parameter variation on the evolution outcomes. Top: By changing the urban expansion settings (Carcassone), we can substantially change the shape of the final city. Middle: Changing the years (or, conversely, the historical setting parameters), the user can control the type of defensive elements used to protect the walls (Gerunda), from square to round towers, to bastions. Bottom: By changing the directions where attackers come from, the user can control the overall distribution of defensive elements (left: uniform attacks, right: all attacks from the north).*

type of structure according to the corresponding historical time, it can easily accommodate such styles. Passive defences, for instance, would be instantiated the same way a moat is built, by selecting the corresponding cells and labelling them accordingly, and the system would incorporate it seamlessly into the simulation process. On the other hand, our current implementation is heavily oriented towards European cities. The generalization to non-European cities, different architectural styles (e.g. Japanese or Chinese castles), different armies and different defences is possible with the current system, but its study is left as future work.

9.7. Extensibility

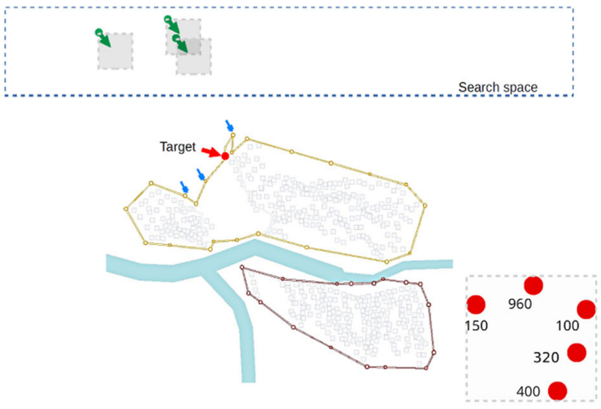
Our system is easily extensible with new events and elements. In our implementation, we have designed patterns to create new events to add into the event list to be checked at each temporal iteration, as described in Section 3. For new geometric elements (e.g. a new tower or a new defensive element), their integration comes in a similar way, by adding them to the construction pipeline and defining their properties (e.g. the movement penalty factor, as described in Section 4.1). Finally, for dynamic elements such as armies, new entities can be easily added and instantiated by the user simply by adding them to the available possibilities and letting the user select them from the corresponding menus, as shown in Figure 3.

9.8. Economic/social/other aspects

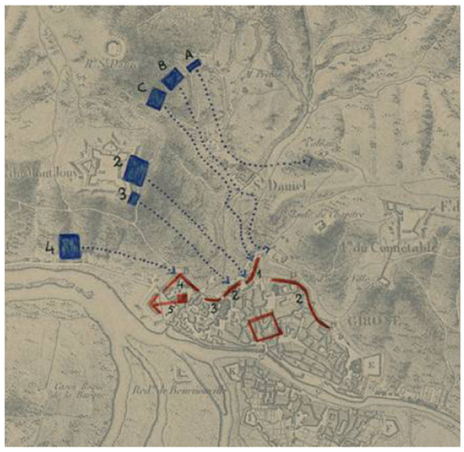
We are mainly interested in the geometrical aspects, and thus, delays due to lack of resources (economical or of any kind) are simply shifts in time and do not imply any geometric alteration. Also, in this system, the user specifies what happens and when. Thus, these aspects only imply not improving the defences at a given time, but at a different one. This hardly represents a big change for our algorithm, which is focused on the geometry. However, for a more complete socio-economical simulation, these aspects could be incorporated as explained in Section 3.

9.9. Number of parameters

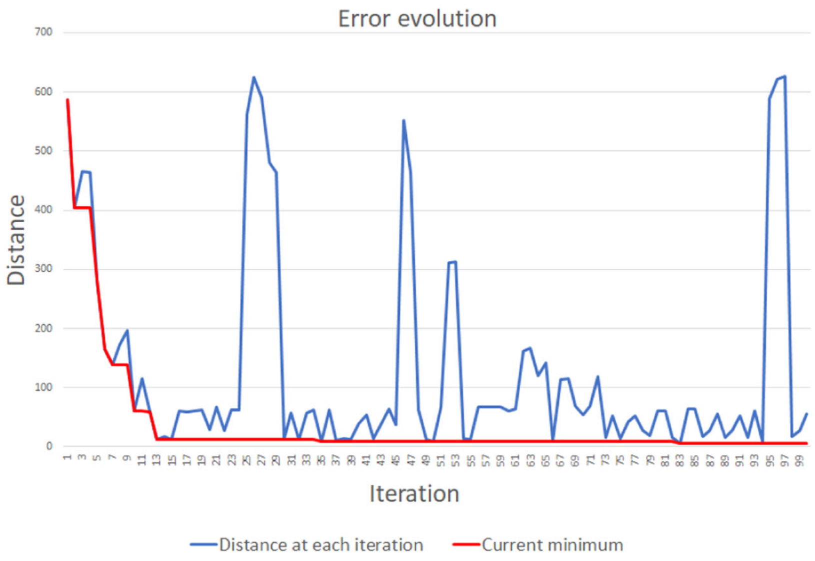
This is, perhaps, the biggest drawback of our system, but fine-grained control is crucial for an accurate simulation (e.g. number



(a)



(b)



(c)

Figure 19: *The result of an inverse design. (A) the global objective is that the city falls where the target red dot is, following the historical events of the siege of Gerunda in 1809. The upper rectangle is the optimization area, and the three green arrows correspond to the three starting points with the best scores. At each starting location, the armies were laid out as shown in the inset (lower right corner). Blue marks show where the defensive cannons were positioned. (B) Reference historical information [PD50] (blue attackers, red defenders). (C) The evolution of the error with the number of iterations. Observe that the absolute minimum is reached at iteration 83.*

of units and attack direction for each battalion). Providing a simplified interface would result in unrealistic settings which could be used without problems in a video game, but completely unacceptable in a historic study. Thus, we decided to provide the users with the maximum control possible, writing a comprehensive user manual to help them. Actually, this is exactly why the inverse system we describe above should be used. In our tests, we have barely scratched the surface of the possibilities of inverse design for this problem, and we believe that, if formulated correctly, even the socio-economical unknowns can be estimated from an inverse mode.

9.10. Wall evolution as a pure geometrical model

Battles were complex processes that strongly depended on many factors, like the sieges on Barcelona before 1714, when the northern part of the city was attacked for the first time. The reason was, according to military treatises, that at the XVIIth century, rifles started to be preferred over artillery. Barcelona's walls were very strong against artillery on its northern side, but was more vulnerable to fusiliers. This type of changes is difficult, if not impossible, to be included with a pure-geometrical model. Instead, these changes are trivial because they only imply a different input from the user.

9.11. Limitations

As we already mentioned, our implementation leaves out a number of important evolution factors. A new set of rules, constructions and agents could be defined to get more accurate city evolutions. First of all, cultural, economic, politic, resource-related or social aspects could be incorporated to affect the outcome of an event. Also, behavioural simulation aspects can be easily accommodated in our system following the description of Vanegas *et al.* [VABW09], but its implementation is left as future work, too. With respect to our fortifications, we would like to include the possibility of having rivers through the city, introducing new castle restrictions. With respect to the battle simulations, we would like to take into account the city surrounding bridges, with new attack strategies. Another aspect to consider would be cities close to the sea, where they were attacked by naval forces. Also, new parametric algorithms to construct other star fort elements, such as *horn works* or *tenailles*, could be considered. Finally, new battalion agents, such as cavalry and miners, could be included in the simulation.

10. Conclusions

In this work, we have presented a novel method to simulate ancient city evolution considering time-dependent, user-defined events that change its natural evolution (i.e. city growth), such as battles, sieges and fortification changes. In our implementation, the city grows around the castle from a set of user-defined parameters, and becomes surrounded by new castle walls when it is required by defensive purposes. At each iteration, the user is free to redefine parameters, or add events that will affect any further evolution. Behavioural simulations are easy to accommodate in our framework, following the interactive scheme introduced by Vanegas *et al.* [VABW09]. An agent-based battle simulation is used to find the castle weak points against attacks. These weak points are used to improve the walls, towers or bastions. The latest evolution in our system transforms a castle into a star-shaped fortress following Vauban designs.

We would like to emphasize that the system we have presented is a particular case of a more general study, where constraints evolve to redefine new procedural parameters for the models, and our castle simulation is one specific application to a larger way of solving similar problems, which abound in procedural generations of man-made environments.

Finally, we would like to emphasize that this is the first paper that attempts to connect geometric modelling with history, which is an extremely complex target. Every city is unique and has its own history, and even experts find it difficult to draw patterns general enough for more than a single city. Thus, with this paper, we are attempting a daunting task, clearly entering into uncharted territory, but which could foster new research in directions never attempted before.

Acknowledgements

We would like to thank Pierre Poulin for his help proofreading this manuscript. This work was partially funded by the TIN2011-14860-E and TIN2017-88515-C2-2-R projects from Ministerio de Ciencia, Innovación y Universidades, Spain. Some images in this manuscript used the Castle Creator library, from Daz3D (daz3d.com/).

Appendix A: Control Parameters

Given the overall complexity of the system and the high number of controllable parameters, we recommend the interested reader to read the provided Supporting Information.

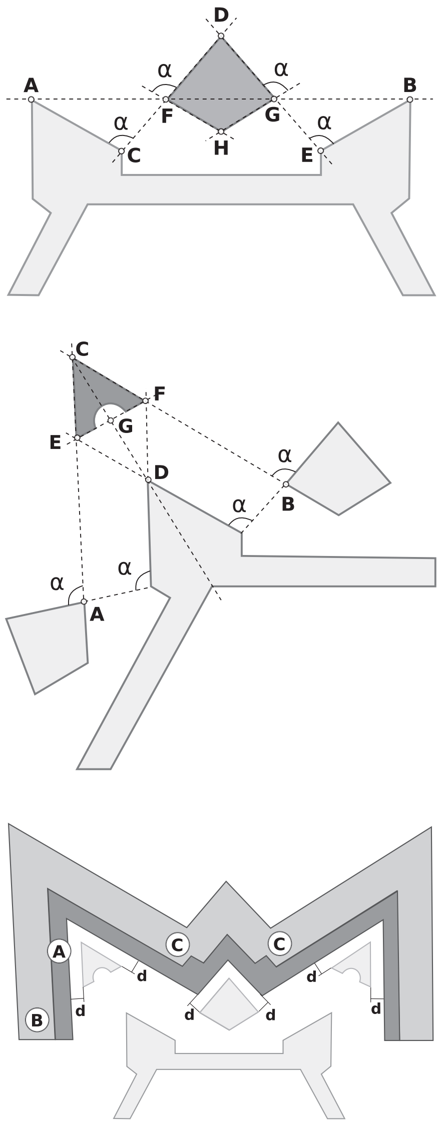


Figure A1: *Star-shaped fortress. Top: Ravelins. Middle: Lunettes. Bottom: Covert way, composed by the curtain wall (A), the glacis (B) and the places of arms (C).*

Appendix B: Star Fort

As mentioned, this kind of fortification defines a star-shaped construction around the castle or city to improve its defences against the combined attack of cannonballs and climbing soldiers. Vauban considered three main kinds of new constructions: *ravelins*, *lunettes* and *covert way*. Ravelins are diamond-shape constructions placed between two bastions, and constructed after their geometry. See

Figure A1 (top), where points F and G are calculated intersecting the segment $\overline{AB}$ with segments $\overline{CD}$ and $\overline{ED}$. The latter ones are the respective bastion sides $\overline{AC}$ and $\overline{BE}$ at a user-defined angle, close to the perpendicular. Finally, from points F and G, two segments, parallel to the bastion sides, are intersected to get the point H.

Lunettes are placed in front of bastions. They require a bastion and its related ravelins, as shown in Figure A1 (middle). From the rear ravelin flanks, two vectors are traced in parallel to the bastion flanks to get point C. Then, the bastion flanks are extended and intersected with the segments $\overline{AC}$ and $\overline{BC}$ to get points E and G, used to construct the rear lunette flank. Finally, the bastion axis is intersected with segment $\overline{EG}$ and a half circle with a user-defined radius is constructed on the back.

The covert way is the wall that wraps lunettes and ravelins, as shown in Figure A1 (bottom). It is placed at a user-defined distance d from both elements, creating the star-shaped structure that gives it its name. In addition, another wrapping structure named *glacis* is calculated following the same method. The glacis is a slope around the fortress that improves defence against artillery. Finally, *places of arms*, a small defensive position, are placed on the non-convex vertices as small squares adapted to the covert way angles.

Except for ravelins, the other star fort components are optional.

References

[AWS08] Alkheder S., Wang J., Shan J.: Fuzzy inference guided cellular automata urban-growth modleing using multi-temporal satellite images. *International Journal of Geo-Information Science 22*, 11–12 (2008), 1271–1293.

[BvMM11] Beneš B., Šťava O., Měch R., Miller G.: Guided procedural modeling. *Computer Graphics Forum 30*, (2011), 325–334. https://doi.org/10.1111/j.1467-8659.2011.01886.x

[BWK14] Beneš J., Wilkie A., Křivánek J.: Procedural modelling of urban road networks. *Computer Graphics Forum 33*, 6 (2014), 132–142.

[BWSK12] Bokeloh M., Wand M., Seidel H.-P., Koltun V.: An algebraic model for parameterized shape editing. *ACM Transactions on Graphics 31*, 4 (July 2012), 78:1–78:10.

[CD19] Champandard A., Dunstan P.: *The Behavior Tree Starter Kit*. 2019, pp. 27–46. http://doi.org/10.1201/9780429055058-3. [Online; accessed 19-January-2020].

[CEW*08] Chen G., Esch G., Wonka P., Müller P., Zhang E.: Interactive procedural street modeling. *ACM Transactions on Graphics 27*, 3 (2008), 1.

[Col10] Collins R.: A dynamic theory of battle victory and defeat. *Cliodynamics 1*, 1 (2010), 1–24.

[DAB14] Demir I., Aliaga D. G., Benes B.: Proceduralization of buildings at city scale. In *2014 2nd International Conference on 3D Vision* (December 2014), IEEE. URL: https://doi.org/10.1109/3dv.2014.31.

[DaG17] DaGraca M.: *Practical Game AI Progrmming*. Packt Publishing, Birmingham, UK, 2017.

[dMV10] d'Eça Moraes Vaz P. A.: *Ancient and Medieval Battle Simulator*. Master's thesis, Universidade Técnica de Lisboa, Portugal, 2010.

[Duf79] Duffy C.: *Siege Warfare: The Fortress in the Early Modern World, 1494-1660* (1st edition). Routledge and Kegan Paul Books, London, 1979.

[Duf85] Duffy C.: *The Fortress in the Age of Vauban and Frederick the Great, 1680-1789 (Siege Warfare, Vol 2)* (1st edition). Routledge and Kegan Paul Books, London, 1985.

[EBP*12] Emilien A., Bernhardt A., Peytavie A., Cani M.-P., Galin E.: Procedural Generation of Villages on Arbitrary Terrains. *Visual Computer 28*, 6-8 (June 2012), 809–818. Special Issue - CGI 2012.

[EMP*03] Ebert D., Musgrave F., Peachey D., Perlin K., Worley S., Mark W., Hart J.: *Texturing and Modeling: A Procedural Approach* (3rd edition). Elsevier Inc., Saint Louis, CA, 2003.

[EVC*15] Emilien A., Vimont U., Cani M.-P., Poulin P., Benes B.: Worldbrush: Interactive example-based synthesis of procedural virtual worlds. *ACM Transactions on Graphics 34*, 4 (July 2015), 106:1–106:11.

[FAR*11] Fletcher J. B., Apkarian J., Roberts A., Lawrence K., Chase-Dunn C., Hanneman R. A.: War games: Simulating collins' theory of battle victory. *Cliodynamics: The Journal of Theoretical and Mathematical History 2*, 3 (2011), 1–24.

[GDAB*17] Garcia-Dorado I., Aliaga D. G., Bhalachandran S., Schmid P., Niyogi D.: Fast weather simulation for inverse procedural design of 3d urban models. *ACM Transactions on Graphics 36*, 4 (April 2017), 1–19.

[GMK13] Grinin L., Markov A., Korotayev A.: On similarities between biological and social evolutionary mechanisms: Mathematical modeling. *Cliodynamics: The Journal of Theoretical and Mathematical History 4*, 2 (2013), 1–45.

[Gri06] Griffith P.: *The Vauban Fortifications of France (Fortress)* (1st edition). Osprey Publishing, London, 2006.

[Hin09] Hindley G.: *Medieval Sieges and Siegecraft* (1st edition). Skyhorse Publishing, London, 2009.

[HMFN04] Honda M., Mizuno K., Fukui Y., Nishihara S.: Generating autonomous time-varying virtual cities. In *Proceedings of the 2004 International Conference on Cyberworlds* (Washington, DC, USA, 2004), CW '04, IEEE Computer Society, pp. 45–52.

[Hof11] Hofschroer P.: *Prussian Napoleonic Tactics 1792-1815 (Elite)*. Osprey Publishing, London, 2011.

[Hog14] Hoganson K.: Computational history: Applying computing, simulation, and game design to explore historic events. In *Proceedings of the 2014 ACM Southeast Regional Conference* (New York, NY, USA, 2014), ACM SE '14, ACM, pp. 18:1–18:6.

[KM06] KELLY G., MCCABE H.: A survey of procedural techniques for city generation. *ITB Journal 14* (2006), 87–130.

[Lep02] LEPAGE J.-D. G. G.: *Castles and Fortified Cities of Medieval Europe: An Illustrated History*. McFarland and Company, Jefferson, NC, 2002.

[Lep09] LEPAGE J.-D. G. G.: *Vauban and the French Military under Louis XIV: An Illustrated History of Fortifications and Strategies*. McFarland, Jefferson, NC, 2009.

[MCSg14] MARZINOTTO A., COLLEDANCHISE M., SMITH C., ÖGREN P.: Towards a unified behavior trees framework for robot control. In *2014 IEEE International Conference on Robotics and Automation (ICRA)* (May 2014), pp. 5420–5427.

[Mer07] MERRELL P.: Example-based model synthesis. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2007), I3D '07, ACM, pp. 105–112.

[MF09] MILLINGTON I., FUNGE J.: *Artificial Intelligence for Games, Second Edition* (2nd edition). Morgan Kaufmann Publishers Inc., San Francisco, CA, 2009.

[MM08] MERRELL P., MANOCHA D.: Continuous model synthesis. *ACM Transactions on Graphics 27*, 5 (December 2008), 158:1–158:7.

[MMP18] MAS A., MARTÍN I., PATOW G.: Heuristic driven inverse reflector design. *Computers & Graphics 77* (2018), 1–15.

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., VAN GOOL L.: Procedural modeling of buildings. *ACM Transactions on Graphics 25*, 3 (2006), 614–623.

[PD50] PLA DALMAU J.: *Colección de cartas y estados gráficos para el estudio de las operaciones militares que tuvieron lugar durante los Sitios de Gerona de 1808-09*. Girona : [el autor], 1950.

[PM01] PARISH Y. I. H., MÜLLER P.: Procedural modeling of cities. In *SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (2001), pp. 301–308.

[RAJ*03] RICE R. S., ANGLIM S., JESTICE P., RUSCH S., SERRATI J.: *Fighting Techniques of the Ancient World (3000 B.C. to 500 A.D.): Equipment, Combat Skills, and Tactics* (1st edition). Thomas Dunne Books, London: Greenhill, 2003.

[Ram14] RAMSAY J. H.: The strength of english armies in the middle ages. *The English Historical Review 29*, 114 (1914), 614–623.

[RCBV14] RUBIO-CAMPILLO X., BLE E., VALDÉS P.: The role of centurion in the roman legion: A computer simulation of battle tactics. In *Proceedings of Ancient Warfare International Conference 2014* (2014).

[RCCC13] RUBIO-CAMPILLO X., CELA J. M., CARDONA F. X. H.: Development of new infantry tactics during the early eighteenth century. *Journal of Simulation 7*, 3 (August 2013), 170–182.

[Sch04] SCHWAB B.: *Ai Game Engine Programming (Game Development Series)*. Charles River Media, Inc., Rockland, MA, 2004.

[SPK*14] STAVA O., PIRK S., KRATT J., CHEN B., MĚCH R., DEUSSEN O., BENES B.: Inverse procedural modelling of trees. *Computer Graphics Forum 33*, 6 (September 2014), 118–131.

[STBB14] SMELIK R. M., TUTENEL T., BIDARRA R., BENES B.: A survey on procedural modelling for virtual worlds. *Computer Graphics Forum 33*, 6 (2014), 31–50.

[STN16] SHAKER N., TOGELIUS J., NELSON M. J.: *Procedural Content Generation in Games* (1st edition). Springer Publishing Company, Incorporated, Cham, 2016.

[SYBG02] SUN J., YU X., BACIU G., GREEN M.: Template-based generation of road networks for virtual city modeling. In *VRST '02: Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (New York, NY, USA, 2002), ACM, pp. 33–40.

[TCTG13] TURCHIN P., CURRIE T. E., TURNER E. A. L., GAVRILETS S.: War, space, and the evolution of old world complex societies. *Proceedings of the National Academy of Sciences 110*, 41 (2013), 16384–16389.

[TLL*11] TALTON J. O., LOU Y., LESSER S., DUKE J., MĚCH R., KOLTUN V.: Metropolis procedural modeling. *ACM Transactions on Graphics 30*, 2 (April 2011), 11:1–11:14.

[Tur11] TURCHIN P.: Toward cliodynamics – An analytical, predictive science of history. *Cliodynamics 2*, 1 (2011), 1–24.

[VABW09] VANEGAS C. A., ALIAGA D. G., BENEŠ B., WADDELL P. A.: Interactive design of urban spaces using geometrical and behavioral modeling. *ACM Transactions on Graphics 28*, 5 (December 2009), 111:1–111:10.

[VAW*10] VANEGAS C. A., ALIAGA D. G., WONKA P., MÜLLER P., WADDELL P., WATSON B.: Modelling the appearance and behaviour of urban spaces. *Computer Graphics Forum 29*, 1 (2010), 25–42.

[VGDA*12] VANEGAS C. A., GARCIA-DORADO I., ALIAGA D. G., BENES B., WADDELL P.: Inverse design of urban procedural models. *ACM Transactions on Graphics 31*, 6 (November 2012), 168:1–168:11.

[VlD53] VIOLLET-LE DUC E.-E.: *La cite de Carcassonne*. Gutenberg: Editions Albert Morange, 1853. http://www.gutenberg.org/ebooks/18940

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

[Wik19a] WIKIPEDIA: Age of Empires—Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Age%20of%20Empires&oldid=920272696, 2019. [Online; accessed 10-October-2019].

[Wik19b] Wikipedia: Command—Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Command&oldid=899630949, 2019. [Online; accessed 10-October-2019].

[Wik19c] Wikipedia: Medieval: Total War — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Medieval%3A%20Total%20War&oldid=917923496, 2019. [Online; accessed 10-October-2019].

[Wik19d] Wikipedia: Rome: Total War—Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Rome%3A%20Total%20War&oldid=917467468, 2019. [Online; accessed 10-October-2019].

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

[WMWG09] WEBER B., MÜLLER P., WONKA P., GROSS M.: Interactive geometric simulation of 4D cities. *Computer Graphics Forum 28* (2009), 481–492.

[ÄDA18] LLKE Demir, ALIAGA D. G.: Guided proceduralization: Optimizing geometry processing and grammar extraction for architectural models. *Computers & Graphics 74* (2018), 257–267.

Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Data S1

Video S2

Video S3

Video S4