The 8th International Symposium on Virtual Reality, Archaeology and Cultural Heritage
VAST (2007)
D. Arnold, F. Niccolucci, A. Chalmers (Editors)

Populating Ancient Pompeii with Crowds of Virtual Romans

Jonathan Maïm[1], Simon Haegler[2], Barbara Yersin[1], Pascal Mueller[2], Daniel Thalmann[1] and Luc Van Gool[2]

[1]Virtual Reality Laboratory, EPFL, Switzerland [2]Computer Vision Laboratory, ETHZ, Switzerland

Abstract

Pompeii was a Roman city, destroyed and completely buried during an eruption of the volcano Mount Vesuvius. We have revived its past by creating a 3D model of its previous appearance and populated it with crowds of Virtual Romans. In this paper, we detail the process, based on archaeological data, to simulate ancient Pompeii life in real time. In a first step, an annotated city model is generated using procedural modelling. These annotations contain semantic data, such as land usage, building age, and window/door labels. In a second phase, the semantics are automatically interpreted to populate the scene and trigger special behaviors in the crowd, depending on the location of the characters. Finally, we describe the system pipeline, which allows for the simulation of thousands of Virtual Romans in real time.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling. I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Animation.

1. Introduction

EPOCH [EPO] is a network composed of about a hundred european cultural institutions whose goal is to improve the use of Information and Communication Technology for Cultural Heritage. Recently, a project has been conducted to revive the ancient city of Pompeii. In the work presented here, Pompeii has been virtually reconstructed based on archaeological data and crowds of Virtual Romans have been introduced in its streets and houses to simulate life before the eruption of volcano Mount Vesuvius in 79 A.D.

We show how a populated virtual version of ancient Pompeii (see Figure 1) has been created, starting with a set of archaeological maps. Our contribution is twofold: (1) We present how the procedural modelling tool *CityEngine* is used to automatically generate an annotated city model based on archaeological input data. (2) We introduce an automatic process that reads the city semantics and induces special behaviors in the crowd of Virtual Romans, depending on their location in the city. We have empirically defined each behavior as a series of scripted actions.

Our paper is organized as follows: In Section 1.1 we introduce previous work in both procedural city modelling and real-time crowd behavior simulation. A system overview of the paper is given in Section 1.2. In Section 2 we present the procedural modelling system *CityEngine*, able to generate a city and attach semantic data to the geometry. In Section 3 the extraction of the semantics from the annotated city model and its subsequent insertion into the navigation graph vertices is explained. Section 4 describes the runtime pipeline of the crowd simulation. In Section 5 we show the results we have obtained, and finally, in Section 6 we summarize this paper and discuss future work.

Figure 1: *A crowd of Virtual Romans simulated in a reconstructed part of Pompeii.*

1.1. Related Work

The city of Pompeii has been studied previously in the LIFEPLUS cultural heritage project related to augmented reality [PSO*05]. The objective was to augment a cultural site

of Pompeii with Virtual Romans. Devlin and Chalmers proposed a perceptually valid method to visualize Pompeii frescos [DC01]. Several cultural heritage projects related to virtual reconstructions that include crowd simulation have been done on other ancient sites [DEY03, FS07, RFD05].

For the creation of 3D models of existing cities, several approaches exist [BBJ*01]. Today, the most widespread method is the use of GIS data, *e.g.*, 2D maps of building footprints in combination with data from airborne sensors [TSSS03, SMVG02]. Despite the impressive results generated with these photogrammetric systems, they are not suited for the reconstruction of partially destroyed archaeological sites, as the missing features and textures have to be added manually or through mobile mapping techniques.

A second method uses GIS data in combination with *procedural modelling*, the latter based on shape grammars [SG71] and L-systems [PM01]. Shape grammars have a long history in analysis and construction of architectural designs [KE81, Dua02]. A shape grammar is composed of a set of production rules which operate directly on shapes (labeled lines and points). By iterative application of these rules, more and more detail is added to a model. In these earlier works, the derivation process was usually done manually or by computer with human supervision.

With the introduction of *Split Grammars* by Wonka *et al.* [WWSR03], shape grammars drew the attention of the Computer Graphics community. Geometric split operations are used as a basic operation to hierarchically subdivide building facades. Based on this approach, Mueller *et al.* [MWH*06] recently introduced the *CGA Shape* grammar, a complete shape grammar framework amenable to computer implementation. The approach includes essential features like a rule syntax, practical shape definition, context-sensitivity, occlusion handling and global synchronization of splits. Complementary to shape grammars, cellular textures [LDG01] can be used to compute brick patterns, while generative mesh modelling allows to generate complex manifold surfaces from simpler ones [Hav05, GBHF05].

In the domain of crowds, various methods have been studied on how to control their behavior by adding semantic data to their virtual environment. Musse *et al.* [MBCT98] classified the environment information into points of interest. Obstacle positions were labeled to avoid collisions with the environment. Sung *et al.* [SGC04] developed a *painting interface* to draw the regions where special events should occur. Based on these regions, their scalable behavioral model was used to have characters react according to a probabilistic action selection mechanism. Yersin *et al.* [YMdHC*05] added semantic information to the environment in real time, allowing the user to interact with crowds through high-level instructions. Shao and Terzopoulos [ST05] developed an autonomous pedestrian model where virtual humans individually plan their actions based on different maps of their environment.

More recently, a work similar to the one presented in this paper has been proposed by Da Silveira and Musse [dSM06]. They introduced a semi-automatic city generation process where semantic information such as population density was used to create the environment. However, they were limited to a fixed number of different buildings, previously stored in a repository. Moreover, although they exploited semantic data to choose where to put which kinds of buildings, this information was not further used to populate the generated environment or apply corresponding behaviors.

Figure 2: *Left: the generated Pompeii districts. Right: Virtual Romans instantiated from seven templates (male and female nobles, plebeians, patricians and a legionary).*

1.2. System Overview

The main goal of our work is the real-time simulation of a crowd of Virtual Romans exhibiting realistic behaviors in a reconstructed district of ancient Pompeii (Figure 2). In an offline process, the city is first automatically reconstructed and exported into two different representations: (1) a high-resolution model for rendering purposes and (2) a low-resolution model labeled with semantic data. A second preprocessing step is the extraction of the semantics to be used during the real-time crowd simulation. Figure 3 displays an overview of the various pipeline stages.

In Table 1, we summarize the different semantics embedded in the city geometry and the associated behaviors. For this paper, we decided to concentrate on this small set since we want to (1) explain the underlying mechanisms in great detail, and (2) emphasize these specific behaviors in the companion video. However, Table 1 is not exhaustive, and many more behaviors could easily be added.

There are several buildings in the city model where Virtual Romans can enter freely. Some of them are labelled as shops and bakeries, and the characters entering them acquire related accessories, *e.g.*, oil amphoras or bread. These accessories are directly attached to a joint of the virtual character's skeleton, and follow its movements when deformed. We can attach accessories to various joints, depending on their nature. Here, we illustrate this variety by empirically placing

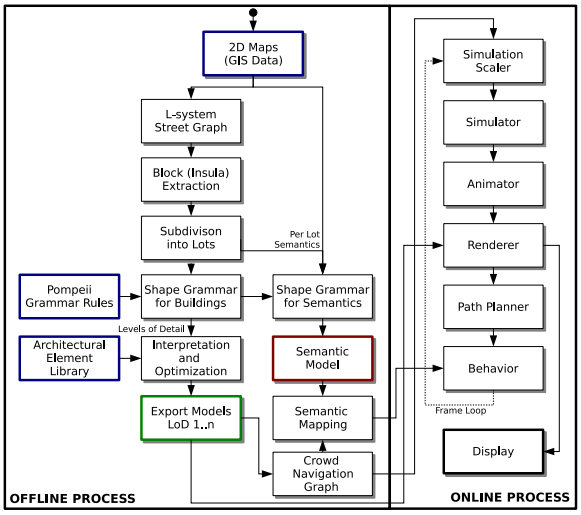

Figure 3: *The overall pipeline consists of an offline and online part. The generation of the city model and semantics is done in an offline preprocess, where also the navigation graph is created and the behavior mapping is achieved. The online part deals with the scene rendering and the execution of the behavior actions based on character location.*

the amphoras in the hands or on the heads of the romans, depending on their social rank.

The idea of rich and poor districts is based on age maps that were provided by archaeologists taking part in the EPOCH project. These maps show the age of buildings in the city. Although we do not yet have the building textures to visually express these differences, we have decided to install the rich Roman templates in the most recent districts and the poor people in older ones. From this, virtual characters know where they belong and while most parts of the city are accessible for everybody, some districts are restricted to a certain class of people: rich Romans in young areas and slaves in poor zones. When the Virtual Romans wander in the city, they may pass near an open door or a window. In this case, we make the characters slow down and look at them.

geometry semantics	behavior	actions
shop	get amphora	walk inside, get out with amphora.
bakery	get bread	walk inside, get out with bread.
young	rich	only rich people go there.
old	poor	only poor people go there.
door	look at	slow down, look through door.
window	look at	slow down, look through window.
	stop look at	accelerate, stop looking.

Table 1: *Summary of semantics and associated behaviors.*

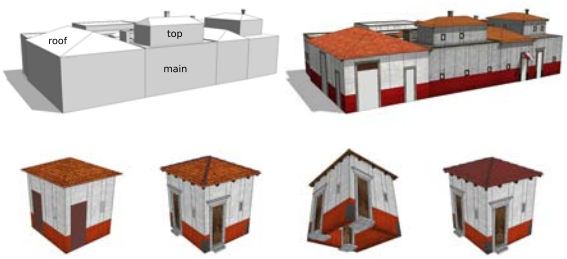

Figure 4: *The Pompeii shape grammar. Top row: On the left an intermediate state in the grammar derivation process after the volume assembly is shown. On the right we show the final subdivided facades. Bottom row: the Pompeii grammar rules contain built-in levels-of-detail which can be controlled with a single variable* `LOD`*. From left to right with increasing complexity:* `LOD=0` *(quads only),* `LOD=1` *(door and window elements),* `LOD=2` *(thick walls) and* `LOD=3` *(individual roof tiles). The results presented in this paper make use of* `LOD=0` *and* `1`*.*

2. Semantic Reconstruction of Pompeii

The ancient site of Pompeii has been extensively studied due to the good preservation of the ruins [Esc79, Ric88, WH94]. Pompeii contains centuries of architectural evolution: from the Italic model of 4^{th}-3^{rd} century B.C. to that of the 1^{st} century A.D. Imperial Rome. The Pompeiian townhouses vary greatly in size and elaboration, from two rooms to large buildings with many rooms and courtyards. The city is divided into blocks, where the individual houses were built contiguously.

2.1. Grammar Rules

Our Pompeii model was created with an extended version of the CityEngine presented in [PM01, MWH*06]. The grammar rules, based on [MVUG05], were enhanced with levels-of-detail capability (see Figure 4), semantic information for the crowd engine and more detailed textures and geometry. The Pompeii grammar rules consist of a combination of volumes and a repetitive subdivision scheme. In addition, the rules contain a set of parameters, *e.g.*, building dimensions, facade proportions, door widths, which are defined as single values or as ranges of values with upper and lower bounds defined by archaeological findings [MVUG05]. The main grammar components are as follows:

1. Each building possesses a *main* volume which is extruded from the footprint polygon. An optional *top* volume represents the second floor. An additional volume called *roof* is placed on top. If the roof is flat, then the top face of either *top* or *main* is used instead.
2. The polygonal faces (facades) of the volumes are labeled with an orientation, *e.g.*, *front* for a face bordering to the street, and tested on occlusion with neighbouring build-

ings. The occluded areas are marked off and will not receive any detailed geometry in the final grammar interpretation step.
3. The resulting raw facade areas are subdivided in a repetitive way: facade → floor → tile → window/door/... → architectural elements.
4. The subdivision is completed with the instantiation and fitting of manually designed architectural elements from the CityEngine library.

In our Pompeii shape grammar structure, the semantic and geometric data for the crowd is collected from several places: the street graph and building footprints allow to constrain the navigation of the characters, while window and door positions are extracted as additional behavior controls for the "look at" action (see Table 1), as illustrated in Figure 5. All these structures contain semantic labels from which the expected behavior will be derived.

Figure 5: *A building contains invisible geometry (checkerboard) which is used to store the semantic information.*

2.2. City Construction

In order to automatically generate the city, the first step in the offline process (Figure 3) is the creation of the *2D maps*. The *Street Graph*, *Block Extraction* and *Lot Subdivision* are executed based on satellite, land/water, elevation, age and population density maps. From these steps, we get a 2D city model composed of polygonal building footprints which are annotated with all relevant data from the maps, *e.g.*, the color value of the land-use maps at the centroid of the footprint [MVUG05]. For the results presented in this paper, we defined five different building types which are mapped to a color code on the corresponding land-use map (Figure 6 (upper right)).

When the shape grammar rules are executed on the footprints, the above mentioned gray value level is applied as input to the grammar and after the *Interpretation and Optimization* process, the name of each node in the exported city model gets a suffix corresponding to the actual building type. For convenience, as later detailed in Section 3, we store the semantic information in a separate scene file, apart from the actual rendered geometry. The navigation graph is then created, based on the scene geometry. This process is not further detailed here, since fully developed in [PdHCM*06,PGT07].

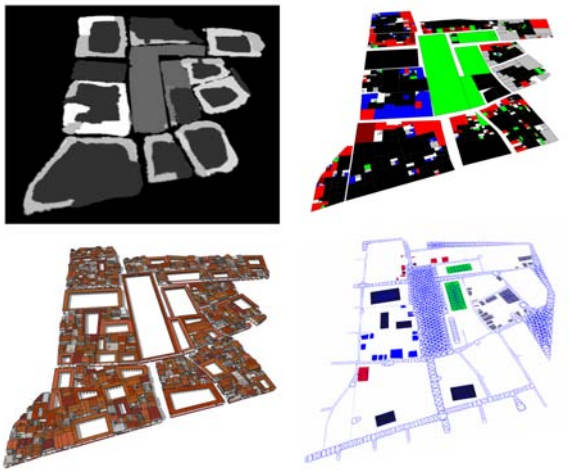

Figure 6: *Four stages in the offline preprocess. Upper left: the manually designed map with five different gray levels for the building types. Upper right: output of the semantic part of the grammar (labeled footprints). Lower left: the output of the geometric part of the grammar. Lower right: the generated navigation graph from the combined grammar output.*

3. From Semantic Map to Behaviors

From the semantic labels created in the city generation (Section 2), a simple file is extracted, containing the data strictly necessary to identify places where specific behaviors should be applied. This file is only composed of labeled building footprints and quads representing windows and doors.

3.1. Extract Geometry Semantics

In the crowd engine, motion planning is based on a navigation graph [PdHCM*06, PGT07] automatically generated from the 3D Pompeii geometry model in an offline preprocess. This navigation graph is represented with circular areas - graph vertices - where it is possible for characters to navigate without colliding with the environment (Figure 6 (lower right)). If two graph vertices intersect, characters can freely move from one to the other. From this, we can find paths between different areas of the city. Thus, at runtime, there is no direct interaction between the Romans and the city geometry. Instead, they interact with the navigation graph. In order to have them behave differently given their surroundings, the semantic data contained in the geometry need to be transfered into the navigation graph vertices. This is achieved in an offline preprocess (Figure 3) where the behavior data and their location are output into a dedicated script.

Depending on the behavior we want to trigger and its location, graph vertices are identified in different ways. For instance, we would like people to slow down and look at the doors and windows when walking past them. In such a situation, we need to identify the graph vertices that are within a certain distance from the doors and windows. For

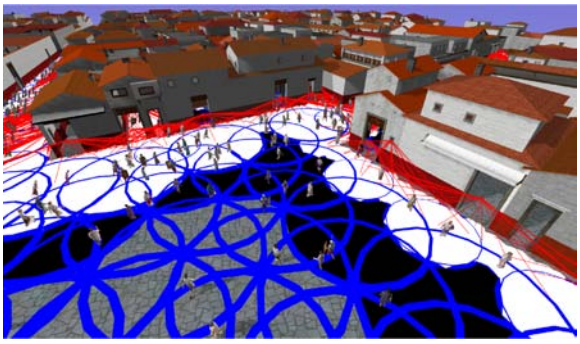

Figure 7: *Graph vertices are marked with special behaviors: "look at" (in white), "stop looking at" (in black), and "target point" (in red).*

the shop and bakery buildings, the Virtual Romans should be able to find their way to get indoors. In this case, we have to find the graph vertices that are contained inside the building footprint; outside vertices, even if close to the building, should not be considered. We have designed an algorithm to read the simplified city model and automatically extract its semantic data to identify the influenced navigation graph vertices. This automatic process is detailed in Algorithm 1:

Algorithm 1: Semantic Mapping.

Data: set of simple geometries with semantic labels, set of graph vertices with their position and radius.
Result: subset of graph vertices GV where to apply corresponding behavior.

```
1 for each geometry g do
2     extract semantic label s from g.name
3     switch s do
4         GV = identifyGraphVertices(s, g)
5         computeBehaviorParameters(GV, s)
```

In our specific scenario of Pompeii, we have developed two methods to identify the graph vertices (line 4) which will adopt a special behavior. The first one, used for window and door semantics, given a range r, returns all the graph vertices that are at a maximum distance of r to the vertices of the geometry g. The second method, exploited for shop and bakery building footprints, detects all the graph vertices contained inside the footprint geometry g. For this particular function, we can reduce the problem to a 2D approach (working on the OpenGL XZ plane). Based on [Bou87], we easily determine when a point p is contained in the geometry vertices by computing and summing the angles $v_i\widehat{pv_{i+1}}$ between each pair of geometry vertices (v_i, v_{i+1}) and the graph vertex position p. If the sum of these angles equals 2π, then the graph vertex is contained inside the polygon. In our particular Pompeii case, the buildings all have simple footprints which allow to exploit this algorithm.

We illustrate the result of our semantic mapping in Figure 6 (bottom right). As shown in Algorithm 1 (line 3), it is possible to treat each semantic label with different methods, depending on their location and effects. At line 5, there is also a second important function *computeBehaviorParameters* that can be developed differently, depending on the semantic label and the corresponding wished behavior. The purpose of this method is detailed in the next section.

3.2. Semantics to Behavior

We know that each semantic label corresponds to a specific behavior. For instance, the window and door semantics trigger a "look at" behavior that makes Virtual Romans slow down and look through the window (see Table 1). To keep our crowd engine as generic as possible, each graph vertex triggering a special behavior also receives a series of variables used later on for parameterization. For our example, this means that each graph vertex associated with this behavior should make Romans look through the window or the door. In order to know exactly where Romans have to look, each of these graph vertices also receives a target point, computed as the center of the window/door quad. Thus, for each behavior, it is possible to compute specific parameters during the graph vertex identification in Algorithm 1. More specifically, at line 3, we test the semantic labels to know which parameters need to be computed, and how the graph vertices should be identified.

In our Pompeii scenario, only the "look at" semantics requires extra parameters. Since a graph vertex may be close to several windows or doors at the same time, we make sure to save a set of target points (one per door/window) as its behavior parameters. When a Virtual Roman crosses such a graph vertex, he chooses among the target points the closest one facing him. For other semantics, no parameter is required. However, the engine has been developed to accept variables in any number.

Finally, this process outputs a script describing which behaviors apply to which vertices and with which parameters. This script is later used at the initialization of the crowd simulation to assign behaviors to graph vertices.

3.3. Long Term vs Short Term Behaviors

There are many behaviors that can be triggered when a Virtual Roman passes over a graph vertex. Some of them are permanent, *i.e.*, once they are triggered for a Roman, they are kept until the end of the simulation, while others are temporary: once the Roman leaves their area, the behaviors are stopped. For instance, a Roman entering a bakery acquires some bread and will keep it when leaving the bakery until the end of the simulation. However, a Roman passing close to a window will slow down to look through it until he is too far away and then resume a faster walk.

The permanent behaviors are not complex to manage. Once triggered, they modify parameters within the Roman data that will never be set back to their previous value. For temporary behaviors however, it is important to detect when a Roman leaves an area with a specific behavior, and set his modified parameters back to normal values. We have chosen to treat this specific case as follows: when Algorithm 1 is called, for each temporary behavior b, a new behavior $stopb$ is created as its opposite. To identify the graph vertices that should trigger $stopb$, we iterate over all vertices invoking b, and check the behavior of their neighbors. If a vertex triggering b has a neighbor that is not triggering b, we have found a vertex that borders the b area. We thus flag it with $stopb$. Note that it is possible for a graph vertex to trigger several behaviors. An example of the "look at" behavior is illustrated in Figure 7.

4. Online Crowd Simulation

At runtime, it is very important to optimize the different pipeline stages to be able to simulate crowds in real-time. Efficient crowd simulation is obtained by targeting computing resources where the attention of the user is focused. The online simulation pipeline in Figure 3 is composed of six important steps and receives two major inputs: the camera view frustum, and a navigation graph filled with Virtual Romans. This graph is used for crowd motion planning [MYMT07] and as a general-purpose structure to hierarchically process Virtual Romans throughout the pipeline.

The **Simulation Scaler** is the first stage of the pipeline. User focus is determined by simple rules that allow to spread computing resources throughtout the environment. Depending on the camera frustum, each graph vertex is categorized with two scores :

- *A representation score*, determined by finding the distance from the vertex to the camera and its eccentricity from the middle of the screen. This score determines the Virtual Roman representation.
- *A score of interest*, resulting in an environment divided into regions of different interest (ROI) [MYMT07]. For each region, we choose a different motion planning algorithm. Regions of high interest use accurate, but more costly techniques, while regions of lower interest exploit simpler methods.

A special scoring is applied to invisible vertices: those containing no Romans are directly frustum culled without scoring and not further considered in the current frame. The invisible vertices filled with at least one character are still kept, but for them, only the ROI score needs to be computed; no rendering is needed for these Romans, but they still require a minimal simulation to move along their path.

The purpose of the second stage of the pipeline, the **Simulator**, is to ensure that each Roman comes closer to its waypoint. Indeed, each Virtual Roman stores a waypoint, which is the position of its next short-term goal to reach. Depending on the ROI, path smoothing is applied, and efficient internal book-keeping of the Virtual Romans on the underlying navigation graph is done.

The third stage, **Animator**, is responsible for animating characters whichever the representation they are using. We use three different representations decreasingly costly to render and animate : *deformable meshes* which are deformed in real time using a skeleton and a skinning technique, *rigid meshes* whose derformation are precomputed, with respect to an animation repertoire [UdHCT05], and *impostors*, extensively exploited in the domain of crowd rendering [TLC02, DHOO05, MR06]. The animation process is similar for all representations: depending on the animation time, the correct keyframe is identified and retrieved. Then, each representation is modified accordingly.

The **Renderer** stage represents the phase where draw calls are issued to the GPU to display the crowds. Shadows are also handled at this stage using a shadow mapping algorithm [RSC87]. The whole process thus becomes a two-pass algorithm: first, deformable, rigid meshes, and impostors are sequentially rendered from the point of view of the main light. Then, the process is repeated from the point of view of the camera. Roman accessories are also rendered in this stage. The renderer is the last stage of the current frame. The next pipeline stages operate on the subsequent frame.

The **Path Planner** stage performs the collision avoidance between Virtual Romans. Due to a complexity in $O(n^2)$, it runs at a significantly lower frequency than the previous stages. Regions of high interest, typically in the vicinity of the camera, are treated with a long-term collision avoidance strategy, while other ROI are treated with short-term algorithms [MYMT07].

Finally, the last stage of the crowd pipeline is the **Behavior**. In our simulation, crowd behavior is updated individually for each Roman, based on the navigation graph vertices. The first step is to retrieve a list of graph vertices for each possible behavior. This is efficiently achieved, due to optimized data structures. Note that it is possible for a graph vertex to be associated with several behaviors. For each behavior, the Behavior stage iterates over its graph vertices, gets the associated Romans, and applies the corresponding actions to them. As detailed in Section 1.2, individual behaviors are mostly expressed through accessories acquisition and animation changes. Such changes do not need to be applied at high frequencies, and a Virtual Roman usually takes more than 1s to cross a graph vertex. Thus, for a realistic crowd simulation, a frequency of 10Hz for the Behavior stage is sufficient.

5. Results

The accompanying video demonstrates the results of our work in reviving the ancient city of Pompeii. First, its streets

and buildings have been automatically reconstructed and annotated. Second, they have been populated with crowds of Virtual Romans, presenting several different behaviors, and thus offering a realistic and varied simulation (Figure 8).

The city reconstruction is based on a CGA Shape grammar with nine different facade designs derived from archaeological drawings. It contains 4 levels-of-detail (LOD), 16 different wall types and 3 roof styles. Of this grammar 16 variations were automatically generated by combining the facades and roofs with specifically designed color palettes. This number could be arbitrarily increased, but practical aspects of the rendering limited the usable number of materials. The CGA Shape grammar has proven its great flexibility, for instance during the optimization of the levels-of-detail for the rendering process.

As for the crowd, seven human templates have been exploited to create the paper images as well as the companion video: male and female nobles, plebeians, patricians and finally, a male legionary. These seven templates are instantiated several hundred times to generate large crowds. To ensure a varied and appealing result, per body part color variety techniques are exploited. The simulation of the crowd

Figure 8: *Crowds of Virtual Romans in a street of Ancient Pompeii.*

in the city has been achieved with an Intel core duo 2.4 GHz 2 Gb RAM and a Nvidia Geforce 8800 GTX 768 Mb RAM. The crowd engine is mainly implemented in *C++*, but to ease the definition of behavior actions, we use the *Lua* scripting language. One of its main advantages is the fast test/fix cycles while programming behavior functions. The city part used for the simulation (illustrated in Figure 6, bottom left) is composed of about 700,000 triangles and 12 Mb of compressed textures. For the crowds, combining the different LOD, it is possible to simulate in this environment 4,000 Romans, *i.e.*, about 600,000 triangles and 88 Mb of compressed textures, with real-time performance (30 fps in average). Note however that we have reduced the number of Romans to 2,000 in the video sequences, due to the requirements of the real-time capture software.

6. Conclusion and Future Work

Based on archaeological data, we have presented the different steps of our work to generate the ancient city of Pompeii and populate it with Virtual Romans. Thanks to the semantic data labeled in the geometry, crowds are able to exhibit particular behaviors relatively to their location in the city. Our results and companion video show that we are able to simulate several thousands virtual characters in the reconstructed city in real-time. The use of a procedural technique for the creation of city models has proven to be very flexible and allows for quick variations and tests not possible with manual editing techniques. For instance, it allowed us to modifiy the land-use maps and recreate the whole model with updated semantics in about 20 minutes. It was also very easy to reduce the model complexity for the crowd engine by omitting some grammar rules, *e.g.*, replacing some window primitives with simpler versions.

One possible follow-up work would be to make the Virtual Romans interact with the model, *e.g.*, opening doors. This would allow to create more intelligent and varied behaviors for crowds. From an archaeological point of view the simulation would also benefit from a validation of the Virtual Roman behavior based on historical data. Finally, to reduce the complexity of collision avoidance between characters, investigating spatial hashing [SGG*07] could be benificial in our architecture.

Acknowledgments The authors would like to thank Mireille Clavien for designing the Virtual Romans and for her great help in this project. Thanks to Damien Maupu for his contribution in the EPOCH project, and to the anonymous reviewers for their comments. The crowd algorithms have been supported by the Swiss National Research Foundation. The integration and application to cultural heritage have been sponsored by the *EC IST Network of Excellence EPOCH*. The procedural city modelling has been supported in part by *EPOCH* as well and by *EC IST Project CyberWalk*.

References

[BBJ*01] BIRCH P., BROWNE S., JENNINGS V., DAY A., ARNOLD D.: Rapid procedural-modelling of architectural structures. In *Proc. VAST* (2001), pp. 187–196.

[Bou87] BOURKE P.: Determining if a point lies on the interior of a polygon, 1987.

[DC01] DEVLIN K., CHALMERS A.: Realistic visualisation of the pompeii frescoes. In *Proc. AFRIGRAPH 2001* (2001), pp. 43–47.

[DEY03] DIKAIAKOU M., EFTHYMIOU A., Y. C.: Modelling the walled city of nicosia. In *Proc. VAST* (2003), pp. 61–70.

[DHOO05] DOBBYN S., HAMILL J., O'CONOR K., O'SULLIVAN C.: Geopostors: a real-time geometry / impostor crowd rendering system. In *SI3D: symposium on interactive 3D graphics and games* (2005), pp. 95–102.

[dSM06] DA SILVEIRA L. G., MUSSE S. R.: Real-time generation of populated virtual cities. In *Proc. ACM VRST* (2006), pp. 155–164.

[Dua02] DUARTE J.: Customizing mass housing: A discursive grammar for siza's malagueira houses. *Phd dissertation, Department of Architecture, Massachusetts Institute of Technology, Cambridge, MA* (2002).

[EPO] EPOCH: Excellence in Processing Open Cultural Heritage. http://www.epoch-net.org/.

[Esc79] ESCHEBACH H.: Zur Entwicklung des Pompeianischen Hauses. In *Wohnungsbau im Altertum (Diskussion zur archaeologischen Bauforschung 3)* (1979), pp. 152–161.

[FS07] FRISCHER B., STINSON P.: The Importance of Scientific Authentication and a Formal Visual Language in Virtual Models of Archeological Sites: The Case of the House of Augustus and Villa of the Mysteries. In *Interpreting the Past: Heritage, New Technologies and Local Development* (2007), p. 49.

[GBHF05] GERTH B., BERNDT R., HAVEMANN S., FELLNER D. W.: 3d modeling for non-expert users with the castle construction kit. In *Proc. VAST* (2005), pp. 49–57.

[Hav05] HAVEMANN S.: *Generative Mesh Modelling*. PhD thesis, UB Braunschweig, 2005.

[KE81] KONING H., EIZENBERG J.: The language of the prairie: Frank Lloyd Wright's prairie houses. *Environment and Planning B 8* (1981), 295–323.

[LDG01] LEGAKIS J., DORSEY J., GORTLER S.: Feature-based cellular texturing for architectural models. In *Proc. ACM SIGGRAPH* (New York, NY, USA, 2001), ACM Press, pp. 309–316.

[MBCT98] MUSSE S. R., BABSKI C., CAPIN T., THALMANN D.: Crowd modelling in collaborative virtual environments. In *Proc. ACM VRST* (1998), pp. 115–123.

[MR06] MILLAN E., RUDOMIN I.: Impostors and pseudo-instancing for gpu crowd rendering. In *Proc. GRAPHITE* (2006), pp. 49–55.

[MVUG05] MUELLER P., VEREENOOGHE T., ULMER A., GOOL L. V.: Automatic reconstruction of roman housing architecture. In *Recording, Modeling and Visualization of Cultural Heritage* (2005), pp. 287–298.

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

[MYMT07] MORINI F., YERSIN B., MAÏM J., THALMANN D.: Real-time scalable motion planning for crowds. In *Proc. Cyberworlds* (2007), p. to appear.

[PdHCM*06] PETTRÉ J., DE HERAS CIECHOMSKI P., MAÏM J., YERSIN B., LAUMOND J.-P., THALMANN D.: Real-time navigating crowds: scalable simulation and rendering. *CAVW 17*, 34 (2006), 445–455.

[PGT07] PETTRÉ J., GRILLON H., THALMANN D.: Crowds of moving objects: Navigation planning and simulation. In *Proc. IEEE ICRA* (2007).

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

[PSO*05] PAPAGIANNAKIS G., SCHERTENLEIB S., O'KENNEDY B., AREVALO-POIZAT M., MAGNENAT-THALMANN N., STODDART A., THALMANN D.: Mixing virtual and real scenes in the site of ancient pompeii: Research articles. *CAVW 16*, 1 (2005), 11–24.

[RFD05] RYDER G., FLACK P., DAY A.: A framework for real-time virtual crowds in cultural heritage environments. In *Proc. VAST* (2005), pp. 108–113.

[Ric88] RICHARDSON L. J.: *Pompeii. An architectural history.* Johns Hopkins University Press, 1988.

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. In *Proc. ACM SIGGRAPH* (1987), pp. 283–291.

[SG71] STINY G., GIPS J.: Shape Grammars and the Generative Specification of Painting and Sculpture. In *Proc. IFIP Congress* (1971), pp. 125–135.

[SGC04] SUNG M., GLEICHER M., CHENNEY S.: Scalable behaviors for crowd simulation. *Computer Graphics Forum 23*, 3 (2004), 519–528.

[SGG*07] SUD A., GAYLE R., GUY S., ANDERSEN E., LIN M., MANOCHA D.: Real-time navigation of independent agents using adaptive roadmaps. In *Proc. ACM VRST* (2007), p. to appear.

[SMVG02] SCHOLZE S., MOONS T., VAN GOOL L.: A generic 3d model for automated building roof reconstruction. In *ISPRS Commission V Symposium 34* (2002), pp. 204–209.

[ST05] SHAO W., TERZOPOULOS D.: Autonomous pedestrians. In *Proc. ACM SIGGRAPH/EG SCA* (2005), pp. 19–28.

[TLC02] TECCHIA F., LOSCOS C., CHRYSANTHOU Y.: Image-based crowd rendering. *IEEE CGA 22*, 2 (2002), 36–43.

[TSSS03] TAKASE Y., SHO N., SONE A., SHIMIYA K.: Automatic generation of 3d city models and related applications. In *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* (2003).

[UdHCT05] ULICNY B., DE HERAS CIECHOMSKI P., THALMANN D.: Crowdbrush: interactive authoring of real-time crowd scenes. In *Proc. ACM SIGGRAPH/EG SCA* (2005), p. 3.

[WH94] WALLACE-HADRILL. A.: *Houses and Society in Pompeii and Herculaneum*. Princeton University Press, 1994.

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

[YMdHC*05] YERSIN B., MAÏM J., DE HERAS CIECHOMSKI P., SCHERTENLEIB S., THALMANN D.: Steering a virtual crowd based on a semantically augmented navigation graph. In *Proc. VCROWDS* (2005).