URBAN PROCEDURAL MODELING FOR REAL-TIME RENDERING

M. Carrozzino [a, b], F.Tecchia [b], M.Bergamasco [b]

[a] IMT Institute for Advanced Studies – m.carrozzino@imtlucca.it
[b] PERCRO, Scuola Superiore Sant'Anna, Pisa, Italy - (carrozzino, tecchia, bergamasco)@sssup.it

KEY WORDS: Virtual Reality, Real-time Rendering, 3D Graphics, Procedural Modelling, Urban Modelling

ABSTRACT:

This paper presents the architecture of the City Modelling Procedural Engine (CMPE), a system for the automatic / semi-automatic reconstruction of virtual models of cities. Although the entire data flow can automatically proceed from the beginning to the end, in each stage the manual intervention of the user is possible to correct mistakes caused by the automatic process, to optimize the results, or to introduce further details. The CPME is integrated in XVR, a framework for the development of VR applications. So far the CMPE engine is still in progress, but many of the main modules have been developed and some interesting conclusions may be outlined both on the quality and on the performance side.

1. INTRODUCTION

An increasing interest has been growing in latest years in the field of procedural generation of urban environments, as this technique can be advantageously exploited both for research and commercial purposes. For instance, the visualization of maps and cities has lately become very interesting popular, as the recent spreading of GPS receivers at cheap prices, to be used as a navigation aid in cars, has stimulated the interest in the market of high-precision maps. The software used by the large majority of navigation devices, whose complexity is or can be compared to that of palmtop computers, usually shows a 2D flat or a "fake 3D" perspective representation of road maps. Indeed, a full 3D visualization of the maps, regarding not only roads but also the surrounding environments, could be very helpful to assist the visual association between the displayed map and the real location. Another application which rapidly gained a large success is Google Earth (http://earth.google.com) , which displays satellite images, aerial images and vector maps projected onto a simple 3D representation of the terrain they are associated. In general, urban planning for analysis, insight, and decision-making, will more and more make a thoroughly use of virtual cities, in the same way as in video games, especially in the context of driving simulations, and in the movie industry.

Whether the application requires a high visual detail or not, modelling a whole city is a long and tiring job from a manual approach point of view. A much more time-effective technique for city modelling may consist in semi-automatic generation of 3D models; this is the technique presented in this paper, where the first and coarser model levels of detail are generated automatically, starting from a set of parameters which may consist in maps or even textual descriptions. Afterwards, a manual intervention of modellers should be foreseen to have a better control on the final results, especially in terms of visual pleasantness and realism.

2. RELEVANT WORK

A great deal of research on the topic of virtual cities is currently ongoing in several universities and departments. One of the most important points of reference on the web is the Virtual Terrain Project (http://www.vterrain.org) which, since 1998, has been hosting a huge amount of articles, images and software tools to "foster the creation of tools for easily constructing any part of the real world in interactive, 3D digital form". The tools and their source code are freely shared to help accelerating the adoption and development of the involved technologies. Currently the project is still far from proposing a unified approach; rather it is a collection of algorithms, taxonomies and applications which, although well categorized, are not yet fully integrated. However, VTP is one of the most interesting and challenging project in the sector.
A remarkable work in the field of the procedural generation of cities is City Engine, the evolution of the work proposed by Parish and Mueller (2001). The engine uses a procedural approach based on L-systems to model cities. From various image maps given as input, such as land-water boundaries and population density, the City Engine generates a system of highways and streets, divides the land into lots, and creates the appropriate geometry for the buildings on the respective allotments. A texturing system based on texture elements and procedural methods allows adding visual definition to the buildings.
Another noticeable work regards Large Urban Environments (Browne, 2004), a comprehensive approach which takes in account both modelling and rendering issues.

The steps for the creation of a virtual city can be resumed in the definition of the following entities:

- Urban Zone: information about geographical and, optionally, social-statistic features of the terrain, usually defined in a GIS system
- Road Network: commonly, but not always, present in GIS systems
- Block definition: very rarely this information is present in the GIS, usually it is deduced from other data
- Lots: blocks are divided into lots with methods ranging from heuristics to pattern instantiation

- Buildings: several methods exist: fully procedural (shape grammars), semi-procedural (declarative modelling) or manual (geometric modelling).

As for the first two entities, input data may be easily available if related to the United States (in the Tiger/line format, http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html), whilst it is sometimes more difficult to be found for other countries, where there is still not such an integrated effort. However, elevation maps under the Digital Elevation Models (DEM) format of several regions overall the world are today commonly available. The most common sources for road data are centerlines, which are described by vector data formats.
Nevertheless, most of the existing data bases are proprietary and not freely available. Therefore it seems convenient to establish a procedure to create such data by commonly available data, for instance extracting information from an existing set of raster maps, or aerial photographs, which are considerably easier to find, either directly in a digital format or after a scanning process from physical maps. Building the vector set from a raster source involves processing the road network raster image, either by hand or with the assistance of an automated computer application.
When passing to 3D, basically two main approaches are available to model roads starting from 2D vectorial data: the road geometry is draped on top of the terrain (faster, but Z-fighting may occur) or it can be embedded into the terrain (no visual issues, but computationally expensive). With either approaches, an algorithm is needed to convert road centerlines (raw vector data) to a full 2D/3D representation and to date there are not established algorithms for doing this.

As far as the blocks and buildings modelling is concerned, the first issue to address is spatial location. A possible approach is the automatic extraction of footprints features from the raster images, as in the case of the road network. Buildings modelling can be carried out using classical techniques. Although manual modelling leads to the most pleasant and realistic results, it is often desirable to perform a parameterization of the buildings properties so as to quickly create 3D models with a reasonable degree of realism. The problem is commonly addressed providing a very rough detail (box-like) for large datasets, increasing the level of detail for some noticeable buildings. Conversely, in the aforementioned work of Parish and Mueller (2001) buildings are generated by means of specialized L-Systems which provide three possible styles: commercial, residential or skyscrapers. Roofs are built from templates. Facades and windows, along with other details, are modelled with procedurally composed textures. This means that photographs of real buildings are taken and subsequently decomposed in elements (windows, balconies, gates etc.) and re-composed adapting them to the surfaces of the generated geometry. This leads to good visual results, even if certain repetitiveness is easily perceivable. The VTP addresses the problem limiting the possible footprint shapes to rectangular, circular or polygonal. In the case of an arbitrary (but forcedly convex) polygonal shape, the roof can only be flat.

The concept of split grammars, a specialized type of grammar operating on shapes, is introduced in (Winka, 2003). Its suitability for the automatic modelling of buildings stems from the fact that restrictions have been carefully chosen so as to strike a balance between the expressiveness of the grammar (i.e. the number of different designs it permits) and its suitability for automatic rule selection. The objects manipulated by the grammar are parameterized basic shapes.

All the examined solutions address often very efficiently, but separately, the single problems involved in the generation of a virtual city. Nevertheless, only commercial solutions are available for an integrated approach which takes in account all of the involved issues. Even in this case, they are often devoted either only to the modelling aspects (favouring visual quality) or the real-time aspects (favouring performances).

3. THE RENDERING FRAMEWORK

The XVR framework (Carrozzino, 2005), jointly developed by PERCRO and VRMedia s.r.l., is a fully integrated environment devoted to the development of VR applications, based on a VR-oriented scripting language specifically dedicated to 3D graphics, 3D sound and, in general, many other typical VR components.
XVR is actually divided in two main modules: the ActiveX Control module, which hosts the interface for web browsers, and the XVR Virtual Machine (VM) module, which contains the technology core, such as the 3D graphics, audio and physics engines, the multimedia engine and all the software modules managing the other built-in XVR features. It is also possible to load additional modules which offer advanced functionalities not directly available. The XVR scripting language allows specifying the behaviour of the application, providing the basic language functionalities and the VR-related methods, available as functions or classes. The script is then compiled in a bytecode which is processed and executed by the XVR-VM.
The integrated 3D engine, built on top of OpenGL, allows to manage the visual output not only on a standard graphical window (either web or local hosted), but also on more advanced devices such as Stereo Projection Systems and Head Mounted Displays. The engine uses state of the art algorithms of culling, simplification, normal mapping and image caching to achieve good real-time performances even with high-complexity models.

Many applications built with XVR make use of 3D urban models. These may include vehicle simulators, crowd behaviour simulators and, lately, also an innovative methodology (ISEE) dealing with the access to information related to Cultural Heritage (Pecchioli, 2008).

Figure 1 – The ISEE application

In this work, interactive 3D models reproducing the main features of corresponding real environments are used in order to map relevant spatial zones to "pieces" of information. The innovative aspect lies in the use of 3D Gaussians both to map the information-related zones and the current view of the user;

this yields to automatically obtain a measure of the "spatial relevance" of information, defined based on its location in the world and on the position/orientation of the user in the 3D space. This application represents a typical case where the visual fidelity is not the main issue: rather, the possibility of rapidly generating even roughly defined 3D models, with a sufficient geographical correspondence with the real places, is a key-aspect in order to quickly prototype the databases to be accessed with this methodology.

4. THE ENGINE

The City Modelling Procedural Engine (CMPE) is a tool for the semi-automatic generation of 3D urban environments, designed to support as many inputs as they are available, like vector maps, raster maps, DTMs, aerial photographs, text descriptions, trying to provide the most coherent output with the provided input. Although the entire data flow can automatically proceed since the beginning to the end, in each stage the manual intervention of the user is possible to correct mistakes caused by the automatic process, to optimize the results, or to introduce further details.

The CMPE main objective is to allow the creation of huge 3D urban datasets, suitable for real-time rendering, which can be:

- quickly obtained with limited user interventions, so as to speed up the creation process (either finalized to a prototype or to the final model);
- stored in a limited amount of memory, so as to allow the transmission over the network of related data even in low bandwidth conditions.

An additional software library (PVRlib) implements the features related to the procedural generation of simple architectural entities (starting from basic geometries like cylinders, boxes and spheres, to more complex shapes like arches, columns, capitals) which can be used to easily add minor refinements to the final model or, embedded in the XVR scripting language, to model more complex architectural entities (like churches, temples, monuments) not directly characterizable by means of simple parameters (fig.2).

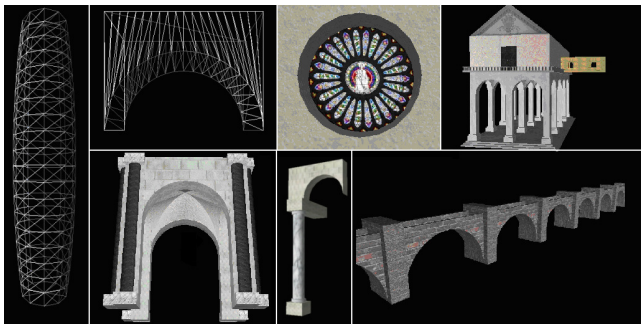

Figure 2 - Examples of PVRlib objects

As a general overview, the CMPE work flow (fig. 3) can be sketched as follows:

- automatic extraction from raster maps of relevant features related to the road network or to the block footprints (this stage may not be performed if data are already in vectorial format);
- generation of data structures for the general management;
- generation of the 2D road network (includes, if possible, railways and rivers);
- generation of the 3D road network;
- identification of the parameters for the subsequent stages (blocks and buildings);
- generation of the 3D blocks and of the 3D buildings;
- storage of the global 3D model in the AAM format, either to directly feed the XVR real-time rendering engine, or to be manually refined within 3D modellers such as 3D Studio Max;
- manual modelling of the details (either with the 3D modelling software or directly within XVR, through the PVRLib) and export to AAM;
- XVR Rendering.

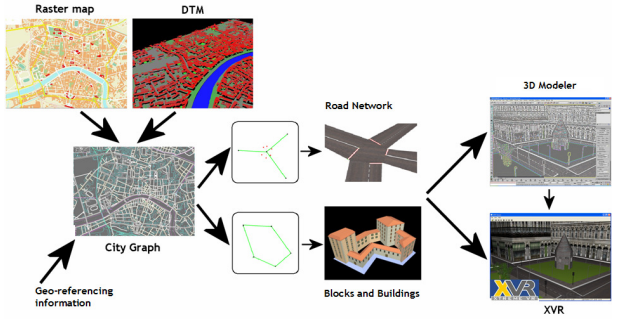

Figure 3 - A hi-level scheme of CMPE workflow

4.1 Automatic extraction of road networks and building footprints

In the current implementation, the CPME input data consists in:

- a 2D raster map of the city to be modelled;
- a DTM of the same region (optional);

with planned support to vector maps. If the DTM is included, both maps must be geo-referenced in order to be correctly co-located.

The first step consists in a simplification of the raster image (which may come from digital archives or even from scanning, therefore it may contain inscriptions, labels or other "noisy" elements, see fig. 4a) to clean the map and make it easier the features extraction. The stages to be performed:

- reduction of the colours to a constant number (which may be user-specified);
- elimination of the "isolated" pixels (noisy pixels produced by image compression or anti-aliasing), assigning to them the mean of the adjacent pixels colours;
- identification of the most frequent colours (let N be the number of these colours) and further reduction of the image colours to N;
- elimination of the labels, if present, as in the case of isolated pixels;

This stage leads to the production of a raster map that, even if not perfectly clean, is much handier for the subsequent stages. This allows a great number of sources which are commonly available and free (like, for instance, scanned maps) to be used

as input data. However, user intervention is possible for small manual corrections.

The second stage consists in the vectorization of the map. The result of this stage is a graph composed of nodes (corresponding to crossroads, curves etc.) and links (which are segments connecting two nodes). Particular attention must be put on squares, which cannot be treated as simple crossroads if wanting, in the next stages, to add details or other architectural elements like fountains etc.

Nodes and links are therefore categorized as follows:

Nodes	**Links**
Blind Alleys	Streets
Crossroads	Railroads
Curve Nodes	Watercourses
Square Vertices	

Table 1. Nodes and links categorization

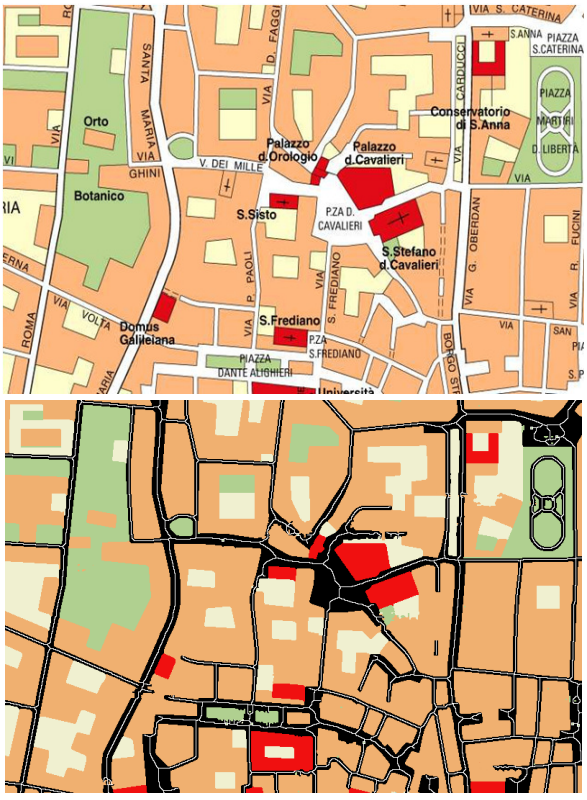

Figure 4: (a) – A sector of a possible input raster map
(b) – The map skeleton overimposed on the cleaned map

To vectorize the map, the identification of the above mentioned relevant features, i.e. nodes and links, is performed by means of a process of *skeletonization* (fig. 4b), a form of thinning aiming to extract a region-based shape feature representing the general form of an object (Blum, 1967). The skeleton represents both local object symmetries and the topological structure of the object. Therefore it is a good starting point for the extraction of vector features from a 2D raster map (Gold, 2001).
The map is then processed in order to identify the relevant points which will be subsequently treated as nodes. The first step is to recognize crossroads, then the blind alleys and finally the curves. In order to get a restricted number of nodes (so as to limit the geometrical complexity of the final 3D model of the road network) the curve bending shall exceed a pre-defined user-specified bending angle. If this happens, the point where bending exceeds the limit is marked as a curve node. In Figure 5a crossroads are highlighted in red, blind alleys in blue and curve nodes in green. In this stage, nodes coordinates are still stored in the picture reference system. If the picture is geo-referenced, the real coordinates will be computed afterwards; otherwise a generic reference system is used.
The obtained graph is then optimized by removing redundant nodes and links, likely to be the result of approximation errors in the previous stage (links connecting blind alleys are removed, crossroads connected by very short links are collapsed, etc.)

The next feature to identify is **roads width**. Different cases may occur:

- blind alley or curve branch: it is enough to measure the width in a certain number of points (usually the extremes and the medium points) along the road orthogonal direction;
- roads ending in crossroads: the extreme points, in this case, cannot be used, as the orthogonal direction should point to another road, therefore the measured width is commonly non significant (it could result even in the intersecting road length). In this case, a series of semicircles is traced starting from the crossroad until the borders of the road are reached.

A particular case is represented by **squares** (fig. 5b). Automatically recognizing squares is not straightforward, as they can be sometimes mistaken by crossroads. An attempt is made by comparing a link width with its length. If their ratio is nearly 1, the link is likely to make part of a square. In this case, a further search is performed to find the nodes related to the adjacent links. The convex envelope of these nodes constitutes an acceptable approximation of the square shape. A better approximation is given by considering the convex envelope of an enriched set of nodes, including additional nodes created on the width boundaries of the roads at their connection with the square (*fake nodes*).
With a similar technique, and with the search of the minimal closed loops in the graph, the boundaries of blocks are identified (fig. 5c). As appears from the picture, block footprints currently do not take in account internal shapes, like courtyards; they are not automatically retrieved in this stage, even if they can subsequently be manually added.

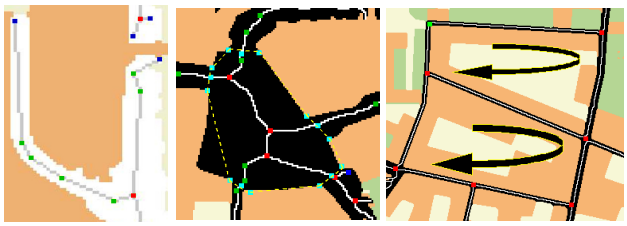

Figure 5: (a) – Nodes highlighting,
(b) – Squares features extraction,
(c) – Blocks features extraction

At the end of the whole process, the complete graph is composed of:

- a set of nodes, characterized by a coordinate pair (x,y) and a type;

- a set of links, characterized by the related nodes, a type and a width;
- a set of squares (collections of connected nodes defining an area lying on the road network);
- a set of blocks, (collections of connected nodes defining an area lying outside the road network);

This City Graph (CG) has an almost 1:1 correspondence with a vector map, which is therefore easily built. The vector map can be displayed as an overlay on the raster map, so as to allow the user to manually modify the nodes position and/or to add new nodes, in order to refine (Figure 6) the obtained results and to correct the possible mistakes so far made. It is worthy of note that user interventions are always optional, hence the algorithm is able to generate an equivalent vector map directly from the raster map.

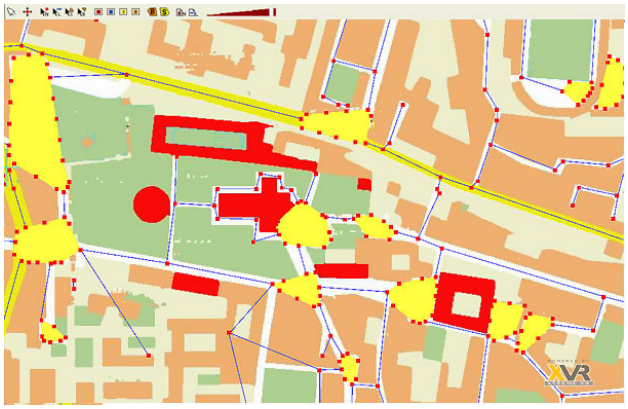

Figure 6:– Manual refinement of the City Graph

The user in this stage can also add labels or additional types to the City Graph elements for the subsequent automatic generation of 3D data. For instance a certain portion of the map can be labelled as downtown or suburbs, so as to provide information about buildings style and size. In the same way, blocks can be marked as **parks**, either because user-identified or because of map colours. Finally, the user can label blocks as **monuments**; in this case the automatic generation of the block might access additional geometrical data either manually or procedurally modelled.

If data are geo-referenced and a DTM is also present, the graph nodes are updated with the addition of the z coordinate. This allow defining also the morphology of the terrain and, if present, the height of buildings. Otherwise the city is considered to by lying on a plane and buildings heights will be pseudo-randomly generated on the basis of qualitative attributes. The next stage consists in the generation of a polygonal mesh representing the road network.

4.2 Generation of the 3D road network

The simplest method to address this issue consists in converting links in quads, using the information related to the links extremes in terms of coordinates and widths. However this would lead to overlapping polygons, resulting in annoying z-fighting effects. Therefore a more refined tessellation (fig. 7 a,b,c) must be done on crossroads (3 or more roads meeting). The opposite problem may occur in curve branches (2 roads meeting) where the simple method could leave uncovered areas, depending on the bending angle. In this case the tessellation should create new triangles to cover these areas (fig. 57 d).

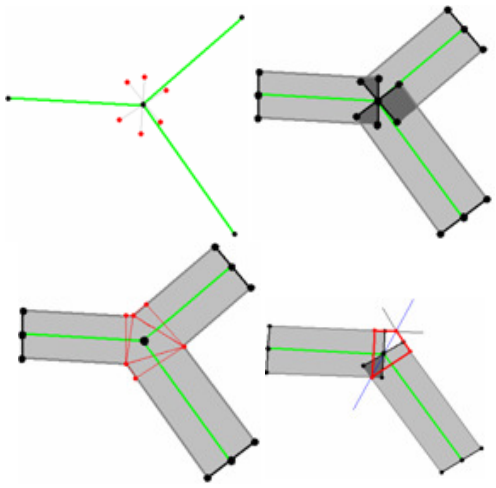

Figure 7: (a,b,c) – Crossroad tessellation,
(b) – Curve tessellation

Subsequently textures, with different traffic signs, centre lines and track signs, are applied to roads, depending on their type, width and importance (fig. 8).

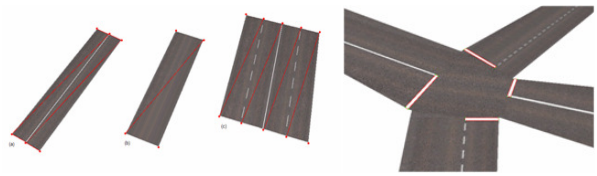

Figure 8 – Road texturing

4.3 Generation of the 3D buildings

The next step is to generate blocks and buildings starting from data contained in the vectorial map. A block is identified by a *line of block*, which represents its perimeter. The engine can generate blocks in two modalities: Perimetral and Mix.

The Perimetral mode simply generates a courtyard inside the perimeter. The thickness of buildings is taken so as to avoid edifices overlapping. The Mix mode is more complicated; the block is completely subdivided in buildings based on the angles formed by the perimeter lines. The size and the height of buildings, if not specified in the City Graph, are pseudo-randomly calculated in order to differentiate buildings inside the same blocks (Fig. 9).

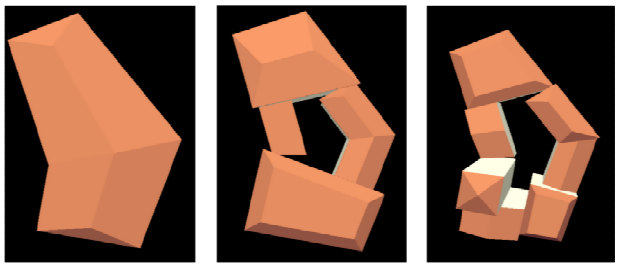

Figure 9 – Different building shapes from the same block

The shape and the appearance of the building can be as well defined and modified, for instance treating the ground floor differently from the other ones or allowing to stack floors of different size (like in skyscrapers, for instance). Several types of roofs have been implemented which can be manually set or automatically chosen from labels and block type (Fig. 10).
In addition to the code-based procedural generation, an authoring/editing tool has been developed to assist the user if a manual intervention is needed to fine tune the model. The tool has advanced procedural capabilities oriented to the buildings modelling, therefore by setting few parameters it is possible to create complex models. Dedicated features are available to model pavements and arcades. The appearance of buildings can be enhanced by applying textures of portals, doors, balconies and windows (Fig. 10). The procedure follows the one described in (Parish, 2001). Work is in progress to generate some of these features as geometry rather than textures (for instance balconies, ridgecaps etc.). It is also possible to have more than one representation to be stored as different LODs.
Parks are treated as a separate case; the minimal LOD is given by a green polygon, but additional entities will be easily added (procedurally generated trees or billboards, from external libraries). The set of entities will be completed by additional vertical traffic signs and stoplights, lamps, benches and other urban furniture. Also in this case rules can be defined to automatically add these features; otherwise they can be manually specified.

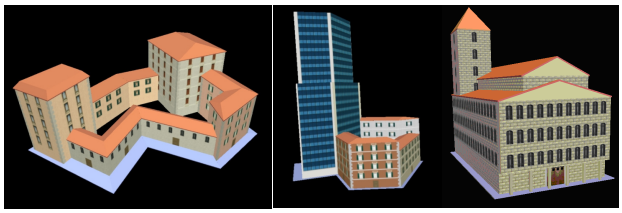

Figure 10 – Textured buildings

All these geometrical entities can be directly generated inside XVR or exported in the AAM format, in order either to be used inside XVR or to be imported into 3D modeller programs where the model can be improved and enriched by:

- correcting possible mistakes;
- refining geometries and textures;
- adding new elements;
- perform shaders-based photo-realistic rendering and exporting static lighting features on textures

A detailed CMPE block diagram is presented in fig. 11.

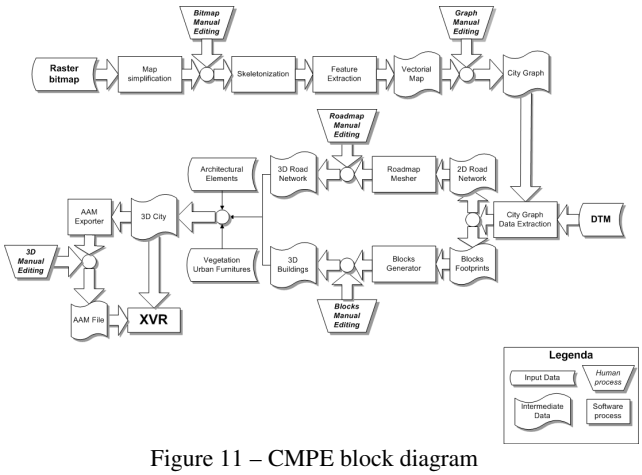

Figure 11 – CMPE block diagram

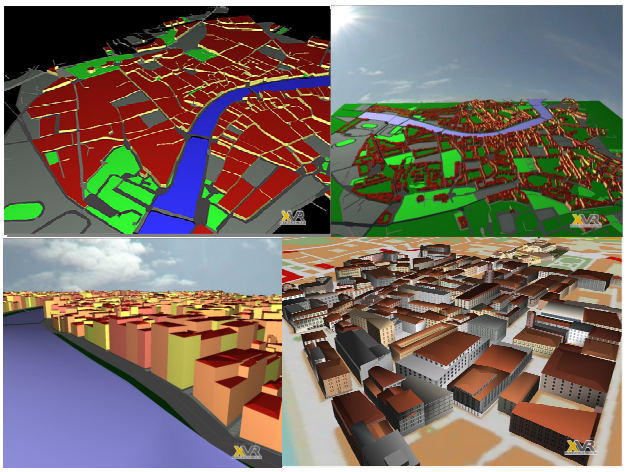

Figure 12 (a) – Simplest LOD for a City Graph of Pisa
Figure 12 (b,c) – More detailed LOD for the same City Graph
Figure 12 (d) – Sample of a textured high-detail LOD

5. RESULTS

Being CMPE still under development, results are only partially significant. However, some tests were made to investigate the possible uses of the engine. Being natively XVR a web-oriented technology, one of the foreseen uses of CMPE is related to internet-based applications. Therefore, measuring CMPE performances in terms of time required for geometry generation, is interesting to understand if it is suitable for on-demand rendering urban models in networked environments. A comparison should occur between a scenario where a 3D urban model of a given complexity is ready to be downloaded and another one where the 3D model is generated on the fly on the base of a raster map downloaded from network.

In the case of on-the-fly modeling, the total needed time is:

$$t_{OTF} = t_{JPG} + t_{CG} + t_{3DM}$$

where t_{JPG} is the downloading time of the raster map from network, t_{CG} is the time needed for the generation of the City Graph, t_{3DM} is the time needed for the procedural generation of the 3D model.

In the case of network downloading, the corresponding time is:

$$t_{DL} = t_{DL} + t_{FL}$$

where t_{DL} is the downloading time of the 3D road network model from network, and t_{FL} is the loading time of the 3D model file in memory.

In the following, network times are calculated considering the best network performances possible (i.e. full bandwidth). The tests were performed on an Intel Pentium 2M notebook, equipped with an ATI 9600 and 512 MB RAM. The models where generated based on maps of increasing size, resulting in increasingly complex geometry:

Map Resolution (pixel)	512	1024	2048	4096
# of links	92	379	1152	3168
# of nodes	156	632	1876	5159
# of triangles	9479	84939	280380	770790

Table 2 – City Graph complexity vs. map size

Fig. 13a shows that the time needed to have the model ready for rendering is slightly shorter in the case of on-the-fly modeling with respect to Ethernet network downloading, while it is considerably better if compared with DSL 4Mb network downloading. Network incidence is almost negligible in the case of on-the-fly modeling, thus DSL and Ethernet offers roughly the same performances.

The efficiency of on-the-fly modeling is more evident (fig. 13b) when the generation of the City Graph (which is by far the most time-consuming operation) is performed on the server side. In this case it is not possible to generate virtual cities from any map (unless the client does not send a *request* to the server by sending the desired raster map, but in this case uploading time should be taken in account) and the detail is not so easily parameterizable.

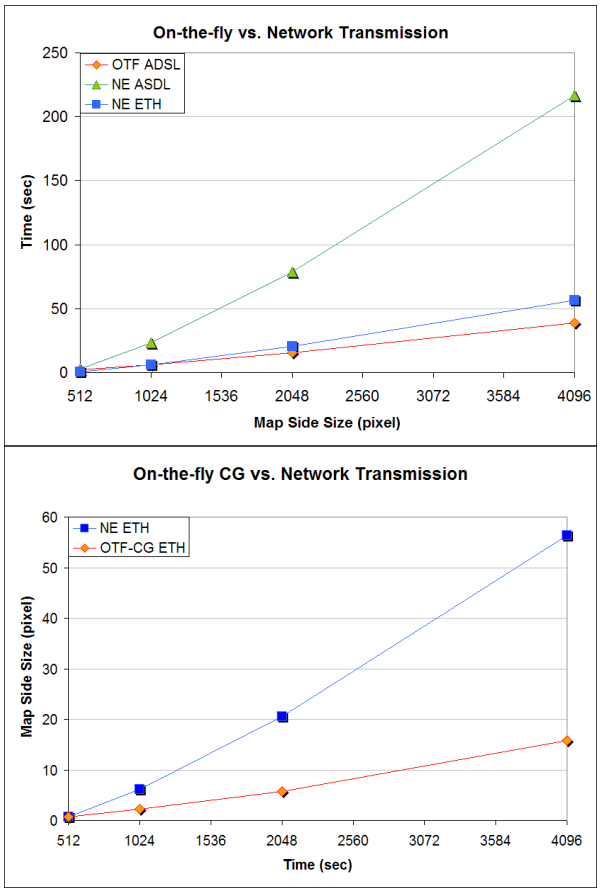

Figure 13: (a) – Performances for CG generation
(b) – Performances for CG download

However, using pre-computed City Graphs reduces the needed time for visualization to:

$$t_{OTF} = t_{CGT} + t_{3DM}$$

where t_{CGT} is the needed time to *download* the City Graph, rather than to generate it. In this case t_{OTF} is considerably smaller than t_{DL} also in broadband network circumstances. A network application could therefore provide a wide choice of City Graphs whose data may be transmitted and locally evaluated in order to generate the 3D geometry. It could also be considered to implement network services which build City Graphs *on demand* having as input data a raster map received by the client side.

6. CONCLUSIONS

We have presented the City Modeling Procedural Engine, a tool for the rapid automatic generation of very complex virtual urban environments, suitable for real time rendering and requiring minimal user interventions. With respect to existing procedural engines, we introduced the possibility of manual refinements in every stage of the work flow, in order to ensure a better control on the final results, usually a weakness of procedural modeling algorithms.

We believe that this mixed strategy can produce interesting results: procedural techniques are used to deal with the quantitative aspects intrinsic in complex virtual environments and, optionally, then manual user actions are possible to refine the generated models from a qualitative point of view. This allows for mistakes or undesired features, resulted from the procedural generation, to be fixed at earlier stages of the process. Besides, as not all the desired components of a complex VE can be described in terms of procedures, manual refinements may be not only advisable but, indeed, needed to produce the expected results.

Many improvements may be performed on the City Modeling Procedural Engine:

- a single, unified interface collecting all of the engine modules in order to provide an integrated framework able to manage the whole workflow;
- a deeper support to **vector road networks**;
- a more extensive support to procedurally generated **vegetation** and **urban furniture**;
- addressing the problem of modeling the whole transport network (including **railroads** and subways), along with **watercourses**, and related elements;
- the integration with the Crowd Rendering Module, jointly developed by PERCRO and UCL, in order to bring life to the generated virtual cities. A cross-research could be performed investigating issues related to the behaviour of the city elements, meaning not only human crowds but also vehicles etc. ;
- the algorithms which rule the automatic generation of the road network, blocks and buildings may be improved to achieve better performances. In particular the map skeletonization results in being the bottleneck of the whole process, therefore alternative approaches may be tried to reduce the processing times;
- due to time constraints and to poor availability of maps and textures, the styles of the generated elements are currently chosen among a few. In order to enrich the range of models, the realization of additional modules for various styles (downtown, suburbs, commercial areas, industrial areas), possibly differentiated depending on countries, is advisable;

- a more intensive use of state-of-the-art techniques for photo-realistic rendering is foreseen, exploiting XVR support to shaders, in order to improve the visual quality of the models without excessively increasing their geometrical complexity. Shaders should also be used to create ex-novo procedural or composite textures, to be used as materials for building or vegetation as in (Zalesny, 2001), having as input data small samples or simple procedural rules.

Another interesting issue to keep into account is the interface to CityGML (http://www.citygml.org) an information model for the representation of 3D urban objects defining not only with their geometrical, topological and appearance properties, but also semantical properties. Since this promises to become an interesting open standard for storing and sharing city models, future work on CMPE will consider the opportunity of realizing models complying with this format.

7. REFERENCES

Blum, H (1967) A transformation for extracting new descriptors of shape, in: W. Whaten Dunn (ed.), *Models for the Perception of Speech and Visual Form*. MIT Press, Cambridge, Mass., pp.153-171

Browne S.P., Willmott J., Wright L.I., Day A.M., Arnold D.B. (2001) Modelling and Rendering Large Urban Environments, in *Proceedings of EGUK 2001*, UCL, London

Carrozzino M., Tecchia F., Bacinelli S., Cappelletti C., Bergamasco M. (2005) Lowering the development time of multimodal interactive application: the real-life experience of the XVR project, in *Proceedings of ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACE 2005, June 2005

Gold, C., Thibault, D (2001) Map Generalization by Skeleton Retraction, in *Proceedings of 20th International Cartographic Conference (ICC 2001),* pp. 2072-2081, 2001

Parish Y. I. H., Müller P. (2001), Procedural Modeling of Cities, In *Proceedings of ACM SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, New York, Annual Conference Series, ACM, pages 301-308

Wonka P., Wimmer M., Sillion F. (2003) , Instant Architecture, William Ribarsky, *ACM Transaction on Graphics*, 22(3):669-677, July 2003.

Zalesny A, der Maur D.A., Van Gool L. (2001), Composite Textures: emulating building materials and vegetation for 3D models, , *Proceedings of the 2001 conference on Virtual reality*

Pecchioli L. Carrozzino M., Mohamed F. (2008) ISEE: Accessing Relevant Information By Navigating 3d Interactive Virtual Environments", in Proceedings of the 14th International Conference on Virtual Systems and Multimedia, IEEE VSMM 2008, pages 326-331, Limassol, Cyprus