The 12th International Symposium on Virtual Reality, Archaeology and Cultural Heritage VAST
(2011)
M. Dellepiane, F. Niccolucci, S. Pena Serna, H. Rushmeier, and L. Van Gool (Editors)

Reconstructing and Exploring Massive Detailed Cityscapes

Enrico Gobbetti[1] Fabio Marton[1] Marco Di Benedetto[2] Fabio Ganovelli[2]

Matthias Bühler[3] Simon Schubiger[3] Matthias Specht[3] Chris Engels[4] Luc Van Gool[4]

[1] Visual Computing, CRS4, Italy [2] Visual Computing, ISTI-CNR, Italy
[3] Procedural, Inc., Switzerland [4] ESAT-PSI, K.U. Leuven, Belgium

Abstract

We present a state-of-the-art system for obtaining and exploring large scale three-dimensional models of urban landscapes. A multimodal approach to reconstruction fuses cadastral information, laser range data, and oblique imagery into building models, which are then refined by applying procedural rules for replacing textures with 3D elements, such as windows and doors, therefore enhancing the model quality and adding semantics to the model. For city scale exploration, these detailed models are uploaded to a web-based service, which automatically constructs an approximate scalable multiresolution representation. This representation can be interactively transmitted and visualized over the net to clients ranging from graphics PCs to web-enabled portable devices. The approach's characteristics and performance are illustrated using real-world city-scale data.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual reality F.4.2 [Mathematical Logic and Formal Languages]: Grammars and Other Rewriting Systems— I.3.5 [Computational Geometry and Object Modeling]: Modeling packages—

1. Introduction

A large part of Earth's population lives and works in dense urban areas, and much of our cultural heritage revolves around complex cityscapes. 3D urban models can potentially be used for a variety of means: as user-friendly interfaces to urban geographic information systems, for visualizing simulation results, for accessing associated metadata/paradata in scientific applications, as well as for communicating with the general public, e.g., for public awareness, instruction, gaming, or virtual tourism.

Pure manual modeling, or direct parametric modeling, of 3D urban environments have demonstrated the ability of creating life-like reconstructions. The labor-intensive approach associated to current 3D modeling and rendering environments is, however, not always applicable at a large scale, and can be accepted only in limited application domains (e.g., movies or video-games). Moreover, such models are typically too complex to be streamed and rendered in real-time, and should be suitably simplified for integration in interactive platforms. A variety of acquisition devices, among them terrestrial and airborne laser scanners and cameras, permit the acquisition of 3D data and color at a city scale. Typical reconstruction pipelines from such data produce measurable 3D models in an automatic manner, but with limited visual quality. Creating a pipeline combining direct reconstruction from acquired data with procedural modeling, and supporting interactive local and remote viewing of such models, would bridge the gap between the artistic re-creation approach of creative industries and the geometric reconstruction from acquired imagery approach of geo-information science.

Contribution. In this system/application paper, we focus on presenting state-of-the-art integrated enabling technology components for obtaining and exploring 3D urban landscapes. In our approach, we reconstruct textured mass models from image, range, and cadastral data. Selected buildings are then refined by applying procedural rules, which replace textures with 3D elements, such as windows and doors, therefore enhancing the model and adding semantics to it. For city scale exploration, mass- or detailed models are uploaded to a web-based service, which automatically constructs a scalable multiresolution representation starting from the detailed geometric representation.

Advantages. The pipeline is meant to be scalable and applicable to wide-area reconstructions with building-level construction details. It is designed to be integrated in larger systems for urban modeling and visualization. As demonstrated by our results on real-world city-scale datasets, the overall system makes it possible to reduce the time for producing detailed cityscapes starting from measurements, and, at the same time, makes those models available to a larger variety of end users through a scalable approach. The simplified multiresolution representation can be incrementally rebuilt during editing sessions, and is interactively transmitted and visualized over the network to clients ranging from graphics PCs to web-enabled portable devices.

Limitations. Modeling and visualization of 3D cities is a complex topic, and our approach has also obviously some limitations. First of all, its focus is on renderable representation of large building collections, and, thus, external components must be used to handle building interiors or other objects (e.g., terrain, roads, and vegetation). Second, multimodal reconstruction requires the availability of synchronized image and cadastral data, which limits its general applicability, and may raise accuracy issues in case of misalignment. We do not see this fact as a major limitation, since accurate combined datasets are increasingly available from imaging companies, and our method can be combined with techniques for footprint and elevation extraction from images. Finally, our streaming model is an extremely compact but lossy representation. Close-ups of landmark building containing important overhangs must thus be handled with other techniques (e.g., textured multiresolution meshes). On the other hand, thanks to its compactness and flexibility, our representation enables smooth large-scale city browsing even on low-bandwidth configurations.

Despite these limitations, the elaboration and combination of these solutions in a single unified pipeline is definitely non trivial and represents an enhancement to the state-of-the-art.

2. Related work

Our system extends and combines state-of-the-art results in a number of technological areas. In the following, we only discuss the approaches most closely related to ours.

Automatic 3D reconstruction of urban environments. (Semi-)automatic reconstruction of 3D buildings and cities is an extensively researched topic comprising a range of input data, approaches, and representations. Purely vision-driven approaches vary in availability of georeferencing information and strategy for creating dense models. Examples are reconstruction of piecewise planar geometries from video sequences [FZ98] and georeferenced vision-based systems for urban reconstruction based on high-resolution airborne optical camera systems [C311]. Multimodal approaches involving georeferenced LiDAR (Light Detection And Ranging) and imagery from different angles have been frequently explored, e.g., for reconstruction of facade meshes [FJZ05] or for generating textured models of individual buildings by first computing 3D geometry from the images and then matching that to the range data [SA02]. More relevant to our work, automated systems have been created for combining georeferenced imagery with aerial LIDAR [FSZ04, MKI09, KP10, PY09b, PY09a]. Here, we harness additional information provided by cadastral data in the form of building footprints.

Procedural techniques for creating detailed city models. Several tools for procedurally generating 3D building models have been described. Here we concentrate on methods based on shape grammars [SG72], which encode architecture in rules, whose derivation generates 3D models. *Split grammars* [WWSR03] and were originally introduced by encoding facades, and have been extended into a *computer-generated architecture (CGA)* grammar [MWH*06] for describing buildings, and, more recently, 3D objects with complex interconnected structures [KK11]. By controlling a few intuitive input variables, complex structures are generated, minimizing manual labor. Traditionally these approaches are used mainly in generative fields such as entertainment [WW08], simulation [MHY*07] and urban planning [HKS08]. Recently, manual [ARB07] and computer-vision-based [MZWG07] image analysis concepts have been merged with procedural modeling. This reverses the information flow: rules (and their driving parameters) are extracted from facade images. Rules permit to procedurally generate approximations of the facade and can be used to generate a more visually appealing 3D representation than the (potentially low-res) 2D images. Moreover, semantic information is obtained, for instance window/door positions or number of floors, allowing for quantitative interpretation. Many of the mentioned techniques have been refined further in the commercial software *CityEngine* [Pro08]. In contrast to image-based reconstruction, LIDAR point clouds of facades can interactively be transformed into polygonal models [NSZ*10]. In our work, we complement this approach, which takes as input 3D data, with the possibility of generating a rule-based representations from oblique airborne imagery of building facades. In addition, we support the insertion of ground-based images for refining the model.

Achieving scalability for streaming and rendering. Exploring the large detailed urban models generated by our pipeline, seamlessly going from high altitude flight views to street level views, handling both mass models and procedurally refined ones, is extremely challenging, since models have high texture and geometric details. The classic solution for managing large textured urban models is the texture-atlas tree [BD05], which, however, considers multi-resolution textures but low-resolution geometry, and does not take into account network streaming issues. Major problems arise when rendering clusters of distant buildings, hard to faithfully and compactly represent with simplified textured meshes. Direct

rendering from procedural models [HWA*10, MGHS11] is appealing, but has currently limited performance and does not handle non-procedurally generated contents. Impostor-like techniques, introduced over a decade ago (see [SCK07] for a survey) are enjoying a renewed interest, because of the evolution of graphics hardware, which is more and more programmable and oriented toward massively parallel rasterization. In the *omnidirectional relief impostors* [ABB*07, AB08] approach, the urban scene is approximated with a cloud of relief maps, and a GPU raycasting is done for each frame on the subset of such cloud which is optimal with respect to the point of view. In this work, we follow instead the BlockMap approach [CDG*07, DCG*09] of exploiting a small number of precomputed maps to efficiently encode and render urban models as a set of compactly encoded textured prisms. This representation is more similar to LODs than to impostors, since it encodes a discretization of the original geometry. We extend that work by introducing a reconstruction engine supporting incremental rebuilds, a web based interface to the reconstruction pipeline, and a refined rendering engine supporting selective exclusion.

3. Overview

Reconstruction of urban environments starts with data acquisition. The more data about a city is available the more information can be exploited during reconstruction. For the pipeline described in this paper, we require cadastral data (building footprints as georeferenced 2D polygons, potentially with metadata such as building height, typically Esri shapefiles), as well as georeferenced multi-angle oblique and nadir airborne imagery. These aligned datasets are typically available from geographic information companies, and are used in system such as BingMaps. In addition we can optionally exploit digital terrain models (DTM, georeferenced model of the ground surface, without any plants and buildings), digital elevation models (DEM, similar to the DTM, but all objects on the ground are included, a typical example being LIDAR data, i.e., georeferenced point clouds), as well as ground-based images of buildings (close-up details, not georeferenced). Other important data types are manually created CAD models (e.g. landmark buildings) and street networks. While such data can be handled by the system, it is not discussed further here.

In our system, the builder component has the purpose to let the user import all available data and combine it to generate 3D models of individual buildings. Starting from the raw data mentioned above, we first reconstruct a mass model of entire cities, i.e., terrain elevation plus photogrammetric urban models in the form of simple, real-world sized 3D building models with flat facades (see Sec. 4). This simplified model is then refined to add 3D procedural details to all or selected buildings (see Sec. 5). Reconstruction typically happens either during an interactive process or in a batch process. At the end of reconstruction, or once each building or group is finalized, the refined models are published to a geometry server, which incrementally optimizes them for streaming and rendering and stores the resulting encoding in an updatable multiresolution database (see Sec. 6). A client-server architecture then enables multiple clients to explore city models stored on the rendering server (Sec. 7)

4. Multimodal techniques for the automatic acquisition of city models

Automatic reconstruction can be applied in batch to all building footprints or initialized in the builder from a user-selected one. We first extrude an initial mesh from the footprint, along with building heights given in the metadata. Next, the mesh is refined using available DEM data to determine roof shapes. The most prevalent DEM data is aerially-acquired LIDAR range data, composed of densely-sampled 3D points. If these are available, we filter points within the footprint polygon, include any known points on the boundary, particularly measured roof corners, and perform constrained Delaunay triangulation to acquire a roof mesh. We then project textures onto the mesh from aerial imagery. Registration between the mesh and imagery is accomplished using measured ground points at the image corners. Oblique images are available along four cardinal directions, and we select textures maximizing image area within the facade.

Occlusion removal and feedback. Visibility constraints are critical to texturing facades with aerial imagery. Particularly in dense urban areas, many facade sections may be incomplete due to occlusion by vegetation or other buildings, so ground images or other assets can be manually or semi-automatically added to complete these areas, as described in Sec. 5. To aid the user, we detect and mark occluded regions on each facade. In cases where facades are hidden by other buildings, we find occlusions by projecting mesh extrusions into the camera and only selecting visible sections of the facade texture, similarly to the approach in [FSZ04]. Vegetation detection has been extensively studied, see e.g. [IBC08] for a recent example using a linear SVM classifier on color infrared images. We use a simple texture classifier on image patches, which we trained on a separate set of manually-segmented images. For each patch, we construct a feature vector containing the following components to encapsulate color and texture: mean RGB and HSV, a five bin hue histogram, and an edge orientation histogram containing orientation and number of modes. The feature is classified as vegetation or background using an efficient approximate nearest neighbor classifier [ML09]. Finally, morphological closing refines the detections into an occlusion mask(see Fig. 2). Besides highlighting missing regions, we also generate a quality indicator for the building based on the percentage of visible facade area, which can then be used for planning acquisition of ground-based images.

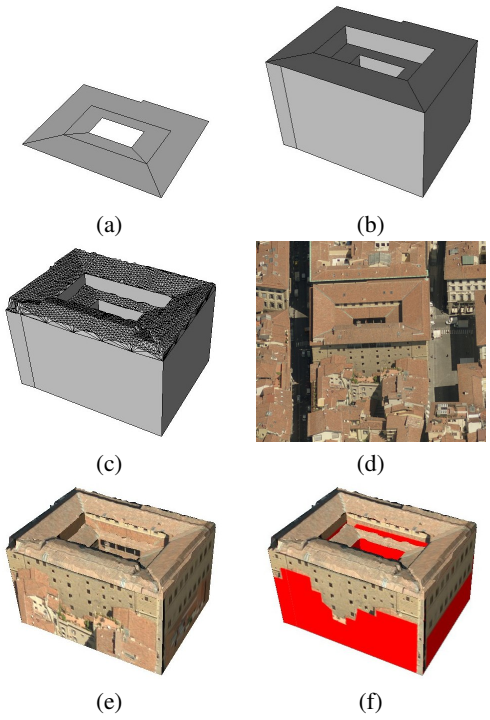


Figure 1: *Automatic Reconstruction of Palazzo Strozzi: (a) user selects footprints; (b) initial mesh extruded; (c) roofs refined from DEM; (d) relevant aerial images selected; (e) building textured; (f) occlusions marked.*

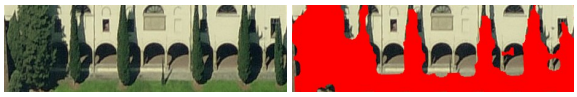


Figure 2: *Left: image region including vegetation. Right: detected vegetation marked in red.*

5. Adding Detail with Procedural Technology

The computer vision techniques described above permit the generation of photogrammetric urban models: simple, real-world sized 3D building models with flat facades. Each facade polygon has a 2D texture projected upon. This texture is a cutout from one of the airborne images. Possibilities to add 3D detail to the model are:

1. Manual modeling with an external tool (e.g., Maya or 3D Max): This is the traditional approach and not further discussed here.
2. Semi-automatic creation of complex CGA facade rule templates from an image.
3. Procedural modeling: If no facade image is available, other sources can be used to design a procedural description of the building. These can be written descriptions, sketches, paintings or archaeological evidence.

The second approach is described in more detail below.

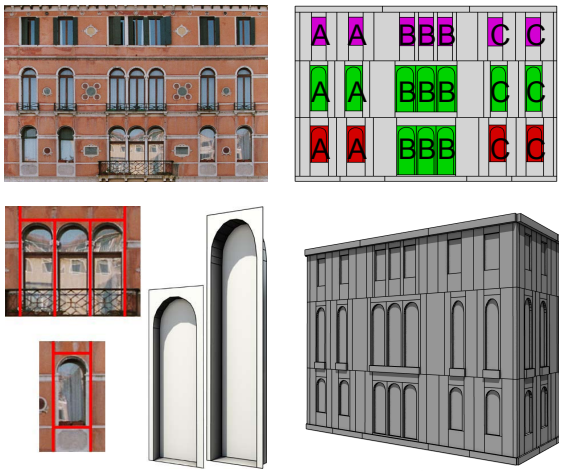


Figure 3: *Semi-automatic creation of a procedural facade from an image. Top left: Image of the facade. Top right: Subdivision scheme applied with identical geometries colored. Bottom left: 3D geometry assets. Bottom right: Final 3D model.*

Semi-automatic procedural reconstruction. Fig. 3 shows how (a ground-based) image can be used to turn a 2D facade into a 3D model. A top-down subdivision hierarchy of the facade [WWSR03, BA05, MWH*06, MZWG07] is interactively defined by the user with the aid of the photography. On top of the image, the user can interactively place split lines and identify tiles with identical geometry. From this information, a number of rules in CGA code are generated automatically, eliminating the need to manually write code. The rules apply the subdivision scheme to an arbitrary polygon and insert 3D assets. Such assets can either be modeled manually or, if accuracy is not the primary goal, downloaded from an asset library such as Google 3D warehouse† or Turbosquid‡.

In the example in Fig. 3, a facade image from Venice is interactively analyzed. The facade is subdivided vertically into two border regions on the bottom and the top, and three floors in-between. This results in following (simplified) CGA code, for more elaborate examples of CGA code see for example [MWH*06] or [MVW*06]:

```
Facade-->
   split(y) { 0.41: Bottom | 4.69: Floor1 |
              ~4.41: Floor2 | 3.89: Floor3 | 0.48: Top }
```

Each floor is then split horizontally; here, each Floor can be described with an A*B*C* scheme (with * denoting repetition).

```
Floor1 -->
   split(x) { 0.35: Wall | ~5.67: Floor1_A_rep |
              0.71: Wall | ~4.56: Floor1_B_rep |
              1.69: Wall | ~5.76: Floor1_C_rep | 0.25: Wall }
```

† http://sketchup.google.com/3Dwarehouse

‡ http://www.turbosquid.com

The A, B and C elements typically consist of a number of 2D wall elements (textured rectangles) and a central 3D geometry (window or door). The rule for A looks like this:

```
Floor1_A_rep-->
   split(x) { ~2.84: Floor1_A }*

Floor1_A-->
   split(x) { 0.74: Wall | ~1.50: Floor1_A_mid | 0.60: Wall }

Floor1_A_mid -->
   split(y) { 1.11: Wall | ~2.91: Window | 0.67: Wall }

Window -->
   t(0, 0, -0.15) s('1, '1, 0.15) i("smallArc.obj")
```

The rules for B look very similar, but a different 3D asset is used. Applying the rules, a 3D version of the facade can be generated.

6. Optimized representation for streaming and rendering

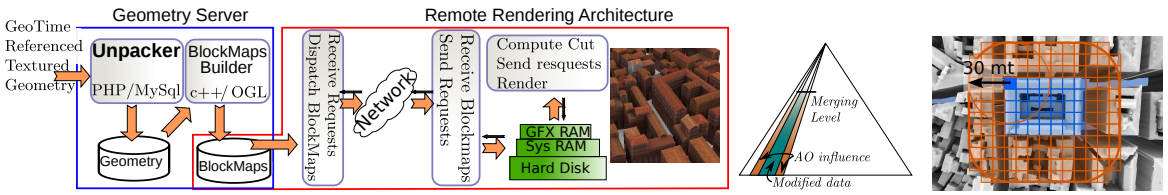
The result of the mass- or procedural reconstruction of a city is ultimately a textured polygonal model. However, a whole city does not have to be built at once as a single large model, but will generally result from several partial reconstruction actions, each one dedicated to a specific portion, which may be a city block, a street or even a single building. We refer a textured model of this kind as *Block*. Our system defines a component called *Geometry Server* (see Figure 4(a)) to which blocks are uploaded and stored. A Block is encoded as an archive file containing:

- one or more files encoding polygonal models;
- one ore more images for the texture of polygonal models;
- the WGS84 [NIM00] coordinates of the models;
- the Historic Date of the model

Specifying the geographical coordinates separately from the polygonal model enables users to work on a local reference frame for reconstruction and to specify the actual location of the model at upload time. This is especially useful to instance the same model in different locations (for example to model row houses). Since cities change over time, each model is also assigned with a *Historic Date*, therefore a dataset may contain several versions of the same building the refer to different moments in history. When a Block is uploaded to the Geometry Server, a simple server-side program (named Unpacker in Fig. 4(a)) decompresses the archive and adds an entry to a MySql database. The database is made of a single table and stores WGS84 position, Date, and a BLOB data for geometry and texture files.

Rendering Urban Environments. Real-time rendering of large detailed urban models is extremely challenging and the many techniques that work for dense meshes cannot be applied to this case. What makes urban models so peculiar is that their geometry is made of many small connected components, i.e., the buildings, which are often similar in shape, adjoining, rich in detail, and unevenly distributed. While multiresolution texturing and geometric levels of detail may provide acceptable solution for buildings near to the viewer and moderate degrees of simplification, major problems arise when rendering clusters of distant buildings, hard to faithfully and compactly represent with simplified textured meshes. In the framework of this project, we developed the BlockMaps [DCG*09], a GPU-friendly data structure for encoding coarse representations of both geometry and textured detail of a small set of buildings. BlockMaps store in a rectangular portion of texture memory all the data required to represent a group of textured vertical prisms defined and discretized on a square grid. The key idea of BlockMaps is that they encode the large scale features of a urban-like dataset, i.e. the vertical flat walls of the buildings with little memory footprint and they can be efficiently rendered with a simple GPU-based raycasting. A BlockMaps dataset is a quadtree where each node correspond to a BlockMap. The BlockMap at the root node corresponds to the whole domain, each of the four BlockMaps at the first level correspond to a quarter of the city and so on, until the deepest level of the hierarchy where each node/BlockMap cover few meters of the city. Building a BlockMap dataset is a bottom up process. The BlockMaps corresponding to the finer level, i.e. the leaves of the octree, are built with an ad hoc sampling of the original geometry, while BlockMaps of the generic level i are built using those at level $i+1$ (see [DCG*09] for details). The sampling algorithm for building a BlockMap requires multiple rendering passes and is not a fast process. For example for a model of 80K buildings may require over four hours on a PC [DCG*09]. While these times are acceptable for static datasets, in this project we wanted to make the process of building the city concurrent and incremental. Since we cannot afford to wait hours every time a building model is updated, we improved the original version of the BlockMaps by enabling partial reconstruction of the dataset.

Partial reconstruction. A partial reconstruction is executed every time a new Block is uploaded to the server, i.e. every time the model has been modified. We recall from the BlockMaps technique (see [DCG*09] for details) that only the leaves of the BlockMaps hierarchy are built from the textured geometry, while the upper levels are built sampling the BlockMaps themselves, therefore we only need reconstruct the leaves of the BlockMaps hierarchy that are affected by the change and propagate the reconstruction to the rest of the hierarchy. A leaf may be affected by a change of geometry *directly*, because it covers a portion of the domain where the geometry has been changed, or *indirectly*, because it is spatially near to a changed geometry. The second case happens because BlockMaps description also incorporate an ambient occlusion term which accounts for the average amount of light that reaches each point due to indirect illumination. In principle the ambient occlusion term at a point depends on the whole dataset, but in practice it may be computed considering a limited surrounding of the point, that in our experiments it is fixed to 30 meters. There-



(a) BlockMap subsystem. (b) Partial reconstruction.

Figure 4: *Left: Geometry server and remote rendering architecture. Right: An illustration of the partial reconstruction of the BlockMaps hierarchy after updating Palazzo Strozzi. The blue squares indicate the BlockMaps covering the domain for which textured geometry has been changed, the orange ones the BlockMaps which AO value may change as a consequence.*

fore, we also mark for reconstruction also the leaves within 50 meters from those whose geometry has been changed. Fig. 4(b) illustrates these steps in the hierarchy, rendering in blue the leaves with changed geometry and in orange those influenced because of the ambient occlusion computation. When the complete set of leaves to recompute is determined, in may reconstruct those levels and hence the portion of hierarchy from those leaves up to the root. Note that if several users make small modifications all over the dataset, we may experiment conflicts in trying to recompute the same internal nodes, i.e. the same BlockMaps, because of different modifications. Although we adopt a lazy update strategy and simply serialize the concurrent modifications to the same nodes, we must observe that, depending on how drastic is the modification and on the depth of the hierarchy, it is not always necessary to propagate them up to the root. For minor modifications (typically, a more detailed version of a building), after few levels, the regenerated BlockMaps are almost identical to the old ones. So, to reduce response times during concurrent editing, every time a BlockMap is recomputed we compute pixel-by-pixel difference with the old version and if the relative change is under a given threshold (we used 3% in our experiments) the bottom-up propagation is stopped (this is indicated as *merging level* in Fig. 4(b)). It should be noted that the errors propagate up, not down, and thus the possible inaccuracies are only at the coarser levels of the hierarchy.

7. Remote exploration architecture

We implemented a prototype client-server architecture enabling multiple clients to explore city models stored on a remote server (see Fig. 4(a)). The only data which is persistently resident on the client memory is the node hierarchy, which is transmitted upon connection, while all the BlockMaps initially reside only on the server side. At each frame, the client performs an error driven visit of the hierarchy until the best approximation to the user defined error threshold with the nodes *locally available* in GFX RAM is met. During the visit, the nodes which have not yet been downloaded, but would be needed by the current view to refine/coarsen downloaded nodes, are requested to the server. On its side, the server receives the requests and send back the BlockMaps. Landmark buildings are handled by selectively excluding BlockMap portions once a predefined LOD is reached, and replacing them with an external LOD representation (typically, multiresolution textured meshes) of the building geometry.

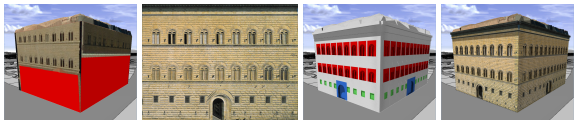
8. Results

We implemented the described system on Windows and Linux platforms using Java, C++, OpenGL, and GLSL shaders. A streamlined version of the renderer was also created for the web platform through WebGL.

We have extensively tested our system with a number of urban models. The quantitative and qualitative results discussed here are for the Florence urban environment, which is an examples of a large scale model created starting from the vision based approach. The source data (8GB) consists of 18.5K building footprints, 671 oblique and orthographic images at 4872x3248 resolution, a digital terrain model of 663K points, and a digital elevation model of 11M LIDAR points. We additionally present results for the ancient city of Rome, representative of a smaller scale but detailed model created by procedural means.

Automatic reconstruction. To detect vegetation, we created descriptors from 15×15 patches as described in Sec. 4. We evaluated our detector on five manually-annotated images (one chosen randomly from each camera direction) using leave-one-out evaluation and obtained an average pixelwise classification accuracy of 96.3%. As seen in Fig. 2, the most common facade artifacts occur in dark areas with excessive sensor noise. We tested our reconstruction system on a PC with a quad-core i7 920 processor @ 2.66GHz and 6 GB RAM. Building reconstruction is run in parallel, with each building reconstructed in a single thread. Averaged over 200 buildings, a single textured building without occlusion detection or roof reconstruction required 3.39s; building occlusion detection requires 1.58s, vegetation detection 0.48s, and roof reconstruction 0.19s, although large buildings may take significantly longer. Running four threads in parallel, reconstruction of the entire Florence dataset including building occlusion detection and roof reconstruction required approximately 345 minutes, including storing the generated geometries and textures to disk (which happens in a single thread).

Adding detail with procedural methods. Details are added to the mass model by procedural means. In this example, we illustrate the results through the procedural reconstruction of *Palazzo Strozzi* in Florence. Fig. 5a shows the textured mass model with occlusions marked in red. A ground-based picture of the front facade (Fig. 5b) was analyzed with our facade builder, resulting in a set of CGA rules describing the structure of the facade (Fig. 5c). The rules add semantic information such as positions of floors, windows, doors etc. to the model. Applying the rules to the 2D facades yields 3D models, because 3D assets such as windows, can be automatically inserted at the right places (Fig. 5d). The semiautomatic analysis, including asset modeling, took a skilled artist about 30 minutes. With access to an existing 3D asset library, the process could be streamlined to 5 to 10 minutes per facade. The actual model generation of the rules takes about 0.2*s*. Polycounts (without roof) are 4 in the mass model and 2721 in the refined model.



(a) (b) (c) (d)

Figure 5: *Procedural refinement of Palazzo Strozzi. a) Automatic reconstruction result; b) Ground-based image; c) Structural subdivision; d) Refined model with 3D facades.*

Multiresolution data generation. We have tested our multiresolution preprocessing and rendering system on a Intel Core i7 950 @3.7GHz, 12 GB RAM, NVidia 580 GTX (1.5GB VRAM). Table 1 shows input size, processing time and output size of for the two example datasets. The construction time depends both on the distribution of geometry and on the extension of the domain in meters. In these tests, we used 128^2 BlockMaps resolution and chose the depth of the tree so that each leaf node covers a square of $8m^2$, which means the we have height values at $6.25cm$ resolution. Since fully processing the models to generate a complete BlockMap hierarchy takes a relative long time, the partial reconstruction introduced in this work has clear advantages. For instance, regenerating the subsets of BlockMaps associated to a single building in Florence (Palazzo Strozzi as in Fig. 5) takes only 75 seconds to regenerate 150 BlockMaps of the hierarchy affected by the change. Note that, due to the sampling nature, the size of the BlockMaps may grow with respect to the original data, especially for procedural models with repeated texture patterns. However, the sampled BlockMap representation has clear advantages in terms of streaming performance (see below), since they approximately require constant per-rendered-pixel bandwidth.

Streaming and rendering. The resulting multiresolution models can be explored using networked graphics clients. As illustrated in the accompanying video, the BlockMaps technique provides a smooth rendering with no noticeable popping artifacts. On average, the frame rate is above $90fps$ at 1024×1024 resolution. The worst case happens when the view is almost horizontal but still over the roofs, because the ray casting algorithm must execute more steps to render each BlockMap and occlusion culling does not help. However, even in these conditions the frame rate never drops under $21fps$ for a full HD resolution (1920×1080). Representative snapshots are presented in Fig. 6 and 7. A drawback of the approach is that the representation is lossy and overhangs are not supported. For this reason, our library supports integration of landmark buildings using alternate techniques (e.g., multiresolution textured meshes) using a distance-based switch. See video for more results.

Model	tri (M)	tex (G)	input (GB)	time (hrs)	tree	out (GB)
Florence	5.96	2.13	9.22	12	10	11.5
Rome	1.74	2.13	3.5	20	11	7.4

Table 1: *Statistics for blockmap construction.*

Figure 6: *Two representative frames of an interactive navigation sequence over Rome.*

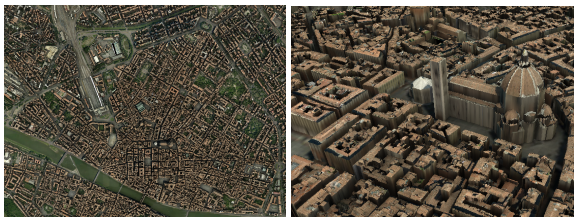


Figure 7: *Two representative frames of an interactive navigation sequence over Florence.*

9. Conclusions

We have presented an integrated pipeline for creating and exploring large-scale 3D models of urban landscapes by combining state-of-the-art solutions in multimodal reconstruction, procedural modeling, and multiresolution visualization. Our solution is meant for integration in larger systems for urban modeling and visualization.

There is much more to city modeling and rendering than is covered by our pipeline (e.g., vegetation and interiors). However, in the modeling/rendering context, our work tackles some of the most critical issues which distinguish

cities from other scenarios. While there are many competing solutions for accurate modeling of single shapes and rendering of interiors and/or large dense meshes, options are currently available for reconstruction and efficient remote rendering of large cityscapes. Future work will aim to improve and extend the pipeline, as well as to complete the integration with a city exploration system (see http://vcity.diginext.fr). In particular, we plan to further automate and speed-up facade analysis, the most time-consuming task of the procedural refinement process, by exploiting computer-vision-based algorithms to detect split lines automatically. Moreover, we plan to improve seamless blending of detailed models with BlockMap approximations by including solutions for remote rendering of detailed textured geometry.

Acknowledgments. This research is partially supported by the European project V-City (reference ICT-231199-V-CITY).

References

[AB08] ANDUJAR C., BRUNET P.: Relief impostor selection for large scale urban rendering. In *IEEE Virtual Reality Workshop on Virtual Cityscapes*. 2008. 3

[ABB*07] ANDUJAR C., BOO J., BRUNET P., FAIRÉN M., NAVAZO I., VÁZQUEZ P., A. VINACUA: Omni-directional relief impostors. *Computer Graphics Forum 26*, 3 (Sept. 2007), 553–560. 3

[ARB07] ALIAGA D. G., ROSEN P. A., BEKINS D. R.: Style grammars for interactive visualization of architecture. *IEEE Trans. Vis. Comput. Graph. 13*, 4 (2007), 786–797. 2

[BA05] BEKINS D. R., ALIAGA D. G.: Build-by-number: Rearranging the real world to visualize novel architectural spaces. In *IEEE Visualization* (2005), p. 19. 4

[BD05] BUCHHOLZ H., DÖLLNER J.: View-dependent rendering of multiresolution texture-atlases. In *IEEE Visualization* (2005), pp. 215–222. 2

[C311] C3: http://www.c3technologies.com, 2011. 2

[CDG*07] CIGNONI P., DI BENEDETTO M., GANOVELLI F., GOBBETTI E., MARTON F., SCOPIGNO R.: Ray-casted blockmaps for large urban visualization. *Computer Graphics Forum 26*, 3 (Sept. 2007). 3

[DCG*09] DI BENEDETTO M., CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., SCOPIGNO R.: Interactive remote exploration of massive cityscapes. In *Proc. VAST* (October 2009), pp. 9–16. 3, 5

[FJZ05] FRUEH C., JAIN S., ZAKHOR A.: Data processing algorithms for generating textured 3D building facade meshes from laser scans and camera images. *International Journal of Computer Vision 61*, 2 (feb 2005), 159–184. 2

[FSZ04] FRUEH C., SAMMON R., ZAKHOR A.: Automated texture mapping of 3D city models with oblique aerial imagery. In *Proc. 3DPVT* (2004), pp. 396–403. 2, 3

[FZ98] FITZGIBBON A. W., ZISSERMAN A.: Automatic 3D model acquisition and generation of new images from video sequences. In *Proc. ESPC* (1998), pp. 1261–1269. 2

[HKS08] HALATSCH J., KUNZE A., SCHMITT G.: Using shape grammars for master planning. In *Proc. Design Computing and Cognition* (2008), pp. 655–673. 2

[HWA*10] HAEGLER S., WONKA P., ARISONA S., VAN GOOL L., MÜLLER P.: Grammar-based encoding of facades. In *Computer Graphics Forum* (2010), vol. 29, pp. 1479–1487. 3

[IBC08] IOVAN C., BOLDO D., CORD M.: Detection, segmentation and characterization of vegetation in high-resolution aerial images for 3d city modeling. In *ISPRS Cong.* (2008), p. 247. 3

[KK11] KRECKLAU L., KOBBELT L.: Procedural modeling of interconnected structures. *Comput. Graph. Forum 30*, 2 (2011), 335–344. 2

[KP10] KARANTZALOS K., PARAGIOS N.: Large-scale building reconstruction through information fusion and 3-d priors. *IEEE Trans. Geoscience Rem. Sensing 48*, 5 (2010), 2283–2296. 2

[MGHS11] MARVIE J., GAUTRON P., HIRTZLIN P., SOURIMANT G.: Render-time procedural per-pixel geometry generation. In *Proc. Graphics Interface* (2011), pp. 167–174. 3

[MHY*07] MAÏM J., HAEGLER S., YERSIN B., MUELLER P., THALMANN D., GOOL L. V.: Populating ancient Pompeii with crowds of virtual romans. In *Proc. VAST* (2007). 2

[MKI09] MASTIN A., KEPNER J., III J. W. F.: Automatic registration of lidar and optical images of urban scenes. In *Proc. CVPR* (2009), pp. 2639–2646. 2

[ML09] MUJA M., LOWE D. G.: Fast approximate nearest neighbors with automatic algorithm configuration. In *Proc. VISAPP* (2009), pp. 331–340. 3

[MVW*06] MÜLLER P., VEREENOOGHE T., WONKA P., PAAP I., GOOL L. V.: Procedural 3D reconstruction of Puuc buildings in Xkipché. In *Proc. VAST* (2006), pp. 139–146. 4

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., VAN GOOL L.: Procedural modeling of buildings. *ACM Transactions on Graphics (TOG) 25*, 3 (2006), 614–623. 2, 4

[MZWG07] MÜLLER P., ZENG G., WONKA P., GOOL L. V.: Image-based procedural modeling of facades. *ACM Transactions on Graphics 26*, 3 (2007). 2, 4

[NIM00] NIMA: *Department of Defense World Geodetic System 1984 – Its Definition and Relationships with Local Geodetic Systems*. Tech. Rep. TR8350.2 - 3d Edition, NIMA, 2000. 5

[NSZ*10] NAN L., SHARF A., ZHANG H., COHEN-OR D., CHEN B.: Smartboxes for interactive urban reconstruction. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2010) 29*, 4 (2010), Article 93. 2

[Pro08] PROCEDURAL: http://www.procedural.com, 2008. 2

[PY09a] POULLIS C., YOU S.: Automatic reconstruction of cities from remote sensor data. In *Proc. CVPR* (2009), pp. 2775–2782. 2

[PY09b] POULLIS C., YOU S.: Photorealistic large-scale urban city model reconstruction. *IEEE Transactions on Visualization and Computer Graphics 15* (2009), 654–669. 2

[SA02] STAMOS I., ALLEN P. K.: Geometry and texture recovery of scenes of large scale. *Comput. Vis. Image Underst. 88* (2002), 94–118. 2

[SCK07] SHUM H., CHAN S., KANG S.: *Image-based rendering*. Springer-Verlag, 2007. 3

[SG72] STINY G., GIPS J.: Shape Grammars and the Generative Specification of Painting and Sculpture. In *Proc. Information Processing* (1972), pp. 1460–1465. 2

[WW08] WATSON B., WONKA P.: Procedural methods for urban modeling. *IEEE Comput. Graph. Appl. 28* (May 2008), 16–17. 2

[WWSR03] WONKA P., WIMMER M., SILLION F., RIBARSKY W.: Instant architecture. *ACM Transactions on Graphics 22*, 4 (2003), 669–677. 2, 4