Real-Time Urban Visualization: Virtual London Case Study

Anthony Steed[1]

Department of Computer Science, University College London

ABSTRACT

We present a case-study of building a real-time urban visualisation for the Greater London area. The model that we have created integrates a variety of high quality mapping resources, but emphasises real-time access to the whole model. The model is extensive (a roughly circular area of 60km diameter), and densely populated with polygon detail down to side-walks and aerial photography in limited areas at 10cm. The combined authoring & run-time system calculates image imposters, and dynamically caches these, and switches in geometry to maintain an interactive frame-rate.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism; I.3.2 [Computer Graphics]: Graphics Systems

Keywords: frontier sets, visibility partitioning, network scalability, networked virtual environments.

1 INTRODUCTION

Urban models have become more popular in the past couple of years[2]. Services such as Google Earth and Microsoft Virtual Earth have aggregated very large and diverse data sources and made it available to the general public. Of course, urban models have long had many professional uses, ranging from architectural visualisation, through to supporting analyses of mobile phone signal propagation. There is still a wide gap between the publicly available data, and that data that might be available to government or professional organisations: mapping data is expensive to create and maintain so high-quality information is necessarily licensed. There is also a wide gap between the software available: public data is selected for the level of visual quality that can be supplied to and rendered by the user's computer, whereas more professional applications require more strict precision and validation, and deal with larger and denser data sets.

In this paper we present an outline of the latest of a series of prototypes that attempt to marry the requirements of real-time and high-quality visual presentation with the high quality and dense data sources that are available. We discuss two classes of issue that we encounter that will be common to many similar models: automatic generation of procedural models using multiple layers of input data that might be partial in their coverage; and real-time rendering of such models which are far larger than will fit in machine memory.

The resulting demonstration, snapshots from which are shown in Figure 1, includes geometric model down to the pavement (side-walk), street and building level. The geometry is extruded with LIDAR height information where that is available and is draped with aerial photography at 10cm in limited areas. To support real-time rendering, the authoring system pre-computes texture imposters and optimised, memory-ready 3D models. We also developed an external spatial-referencing system that allows us to read 3D geometry from files on disk, in a geographic order (e.g. from north-east to south-west), rather than file order.

[1] A.Steed@cs.ucl.ac.uk

2 INPUT DATA

There are four main types of input data to the visualisation system:

- Ordnance Survey (OS) MasterMap® vector mapping data[6]. This is described in more detail in Section 2.2.
- Aerial photograph at a number of different resolutions up to a pixel equal to ~10cm
- Height data from aerial LIDAR (Light Detection and Ranging)
- Hand-crafted building models in 3DS format.

There are few constraints on what data is available, as long as it keyed to the UK's National Grid Reference system (hereafter reference to a "OS grid")[5]. The majority of the OS MasterMap data is supplied in files that cover 2km by 2km, but polygons are not clipped to files, each is stored in the file containing its centroid, meaning that a polygon might commonly overlap up to four tiles, and more than four for a few extremely large features (e.g. train lines or rivers).

Aerial photography can cover any ortho-rectangular area, and the modelling system deals overlapping imagery, taking the high-resolution available.

Height data is simply stored as files containing, X,Y position in the OS National Grid Reference system, with a height in metres.

The hand-crafted building models come from a number of different sources, including models generated from CAD packages such as MicroStation, and models from photogrammetric packages such as RealViz ImageModeller.

We describe the layout and naming in some detail, because we would emphasise that the modelling and real-time engines both deal with data that is not complete, in that, for example, the procedural geometry system can deal with buildings that are only partially covered by aerial photography. With several layers such issues can be quite complex.

2.1 Layout and Naming

As mentioned previously, most of the OS MasterMap data we have access to is in files covering roughly 2km by 2km. This is not exclusively the case though, and there are some smaller tiles. For convenience we arrange all available resources in a hierarchy based on the OS grid system. This is best explained by referring to the portion of the directory tree illustrated in Figure 2. The first level of the hierarchy refers to the media type. The second level "tq", "su" or "tl", refer to 100sqkm sections of the OS grid. The first letter designates a particular tile out of 25 in a 5 by 5 grid (thus 2500 km sq area) that covers (easily) the British Isles. The second letter designates a particular 100 km square out of the 25. In both levels the squares the letters proceed from north-west corner east and then south (the letter "i" is not used). The Greater London area overlaps three of these 100km tiles, with the letters

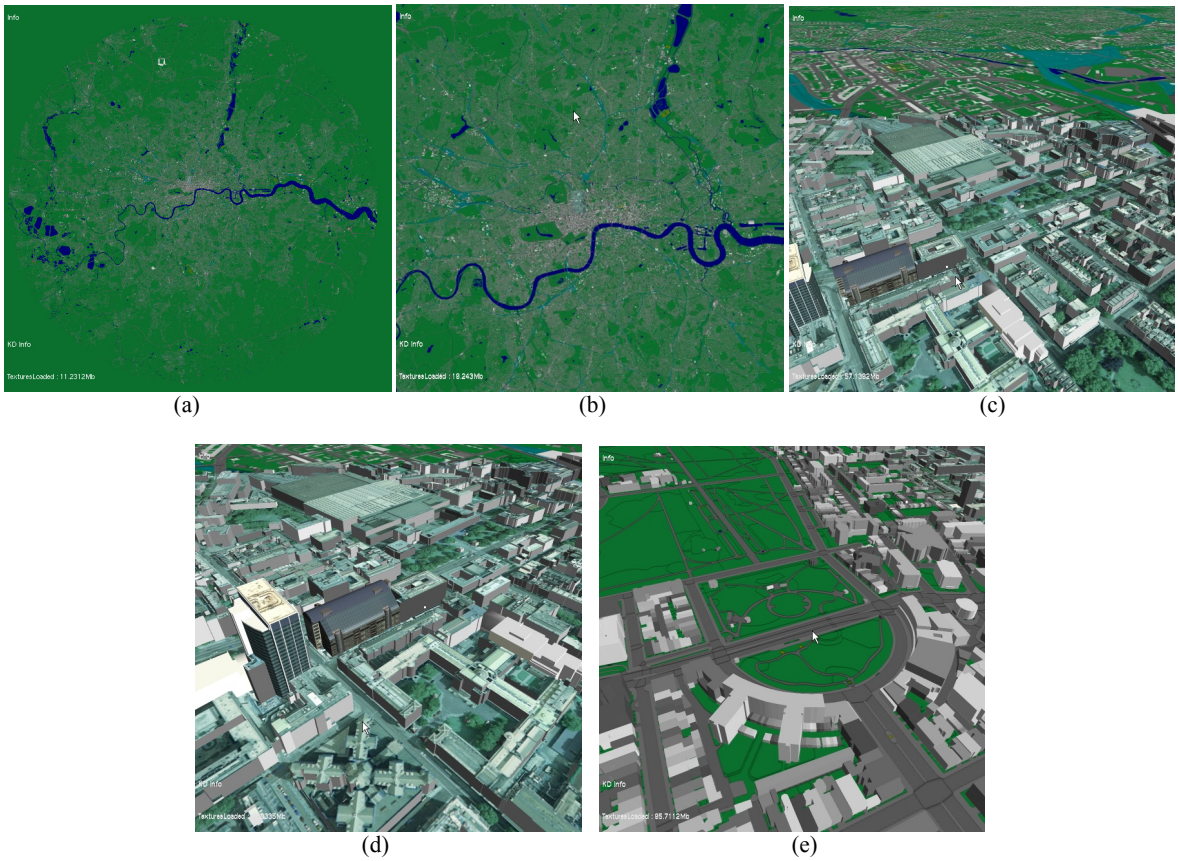

Figure 1. Images from the real-time data visualization system (a) Overview of whole model (b) Zooming in (c) Coming in to the UCL area with geometry for the near-distance and the generated imposters in the mid and far distance (d) Hand-crafted models of the UCL area amongst procedurally generated models with aerial photography (e) Typical geometry in the model where these is no aerial photography. Plain false colours are used these areas. Referring to (a) we can see that the majority of the model is actually un-surfaced (i.e. green)

TQ,SU and TL. The next level of the hierarchy, e.g. TQ2977, designates an individual 1km square, with the first two digits being the easting, and the second the northing.

The MasterMap data itself is stored in a hierarchy only as deep as the two letter identifier. We use the six character identifiers to order the other files, which are more numerous and from a variety of sources. We only use 2km divisions, matching the typical MasterMap file, thus each directory contains files for 2km by 2km. The coordinate used is the centre of the 2km square. Much of our data is centred on UCL, which is on squares TQ2977 and TQ2981. Occasionally a file in a directory will overlap adjacent 2km tiles, but the software will scan and load files from adjacent directories to accumulate all the resources required to create or render a model of any particular area.

Because of its size, aerial photography is stored in subdirectories, with the convention being that tiles are 1024 pixels square, but covering different scales (typically 125, 250, 500, 1000m). Within these directories, the individual files are labelled with an x and y position (e.g. 4_5.png). Not all the files need exist at any level.

2.2 OS MasterMap

Whilst a lot of effort is made on construction of new mapping sources, the OS MasterMap is a comprehensive set of mapping of Great Britain[6]. The Topography Layer has precise vector outlines of all features such as roads, pavements and buildings. It is stored in Geography Markup Language (GML)[4]. The data is normalized in the sense that all land is partitioned in to areas each classified as a single polygon. Each polygon is made of a chain of polylines, where polylines are shared between adjacent polygons. Vertices are also shared. Each polygon has a feature type stating what the object is (wall, building, pavement, lawn, etc.)

Making simple extruded models from this data is straightforward, as each building is already identified. Individual polygons, lines and points are identified by a TOpographic IDentifier (TOID) which can be used as a key to a wide variety of other datasets such as postcodes and addresses. Note that common places such as “University College London” would be mapped to many TOIDs.

3 Modeling System

There is a single piece of software that is responsible for modelling and rendering the data. An overview of its structure is given in Figure 3. The modelling system is responsible for creating all the derived assets. These are the binary 3D models, and the texture imposters. The run-time system then switches between displaying original or derived data, or combinations of the two. By default on start up, the software optionally builds all derived data where there is newer source data than the derived

```
Assets
  - GMLData
    - TQ
      - TQ2977.gml
      - TQ2981.gml
        …
    - TL
      …
- SU
      …
- AerialPhotography
  - TQ
    - TQ2977
      - 1000m
        - 1_1.png
        - 1_2.png
        …
      - 500m
        …
    - TQ2981
      …

Assets (continued)

- LIDAR
    - TQ
      - TQ2981
        - tq2981nw.txt
          …

- LandmarkModels
  - TQ
    - TQ2977
      - UCH.3ds
        …
```

Figure 2: Asset directory layout

data. This takes approximately 8 hours if all derived data is removed.

3.1 3D Modeling

Given LIDAR information and polygonal outlines of buildings, the procedural extrusion process looks straightforward. Although in previous work, we spent considerable time fixing ground plane interpolation to create a terrain[7], in the current model, we only have sparse LIDAR, and this left the question of how to interpolate heights outside the region for which LIDAR existed. Thus we mapped relative height between inside a building and the closest points on exterior ground plane objects (roads, parkland, etc.). The height of a building is simply calculated as the average of all LIDAR heights within the building outline minus an average of 3-4 points on the ground nearby. For areas without LIDAR we used an estimated height based on building footprint and a look up table of similarly sized buildings with known heights. The intention is in the near future to use a terrain model that comes from another source, which is complete for the British Isles, and use the offset models against that terrain.

The main complication with geometry modelling is the draping of texture maps. We support the draping of texture that only covers part of a building. If this is the case, we clip the 2D polygon to the border of the texture. This generates more polygons, but leaves polygons needing simpler shading models.

The secondary problem is that the texture maps can be very high-detailed. We are unable to create and manage multiple textures at 16K by 16K. We thus constrain all texture maps to be 1024 by 1024 pixels. Thus our internal representation of geometry nodes for 3D objects has pointers to a number of different possible texture maps. At run-time, we dynamically select the texture map to use, depending on which texture appropriate and is already cached in the AssetManager (see Section 4.1). For every geometry node we then dynamically change the texture coordinate matrix, so that we do not need to change the stored texture coordinate values on the geometry. This does mean that all geometry must again be clipped: two vertices in the same polygon might be in one texture map at one level of detail, but separate texture maps at another level. We need to clip the geometry against the texture maps: this time, we clip geometry down to all smallest texture maps that are stored in the asset repository. In practice, this doesn't lead to too many more polygons, as even the smallest texture maps cover about 100m by 100m.

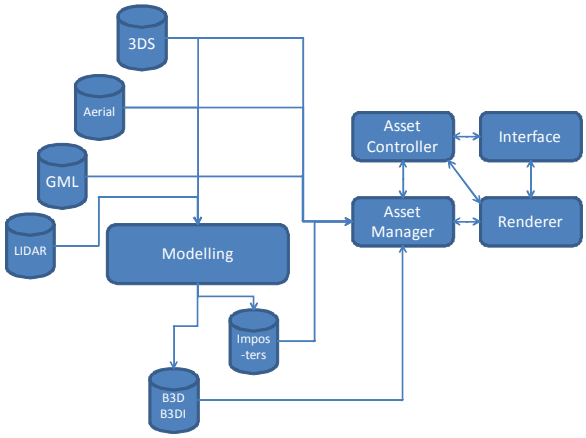

Figure 3: Modelling and Real-Time Components

If there is no aerial photography to drape, we do not clip the models. This means that the geometry files may have ragged edges, because the underlying GML data contains whole shape outlines unclipped to the tile boundaries.

Once all the geometry is created, is it stored on disk with abstract id numbers replacing pointers. This format can be read directly into memory and the pointers regenerated. This is the file format .b3d. In Section 4.2 we introduce.b3di files which are a separately stored index files.

3.2 Imposter Generation

Imposters are generated for vertical down aerial views. We generate these because the aerial photography doesn't cover the whole space, and we will, in any case be pasting hand-modelled geometry over the aerial photography. The imposters are generated the from the geometry as exported to the .b3d. We create imposters on a 2km by 2km grid. Creating these files, thus involves creating and loading all the assets for that tile. As noted, to create all the assets for a grid, we might need to load adjacent tiles because the GML have ragged edges, and hand-crafted building models might straddle two tiles. Imposters are rendered using an orthographic camera, using the standard real-time renderer (see next section), with all 3D detailing turned on.

4 Real-Time Engine

The real-time engine is responsible for generating visual imagery of the model. It provides a continuous real-time experience. Nevertheless, because of the sheer amount of data, popping of objects in to view is quite common.

4.1 Asset Management

In Figure 3, the Interface, Renderer and AssetController modules are in one process, and the AssetManager in another. Currently there are just these two processes, but we are working on making the AssetManager itself in to multiple processes.

The Renderer normally renders either the textured imposters or the 3D model assets for each tile in the model. The AssetController is responsible for scheduling switches between the two, and dealing with latency in loading. The decision is made on a proximity range to the closest corner of that model, with 3D models being cued to start loading at a distance of 3km away. The Interface constrains flying and moving, so that these ranges work well. Initially the user flies towards the model, and when they reach a height of roughly four times roof height, the Interface then

pitches the camera up, in to a more horizontal flying mode. This means that the 4 closest tiles that are loaded or being loaded are likely to be the ones visible when the user approaches the ground (c.f. an oblique approach, where the closest during tiles would "slip underneath" the camera before they were seen close to). The files are streamed, in a geographic order as described in the next subsection.

Texture imposters are treated in a similar way, in that pre-computed mip-mapped textures are loaded for all tiles at the start, and then as the tile covers more screen space, mip-map layers are inserted in to the texture for the tile, until the tile image reaches 1024 square pixels. We only apply larger images as textures to 3D models, with the workaround described previously that allows textures at resolutions of 1 pixel every ~10cm resolution.

The Interface additionally allows the user to select to display any of the original assets, including 2D rendering of the original GML data, rendering of the ground plane with the image imposter, even when zoomed beyond its LOD, etc. We can also load the GML data and use it to key the existing 3D model, so that, a 3D model can be selected, and the original TOIDs identified. Loading and parsing the GML data for a tile can take 20-40s depending on the tile complexity, so this is never done automatically, only when the user requests.

The AssetManager is responsible for reading files and preparing assets for rendering. It receives instructions from the AssetController and generates messages back when assets are fully prepared. The Renderer marks those assets it is actually using, so the AssetManager can implement a least-recently used caching policy when memory is constrained.

4.2 External File Indexing

The introduction of large 3D model files led to us developing an external index for the files. Because the geometry tiles are so large, they take several seconds to load. We schedule this loading in the background, but our experience is that because of the relatively unpredictable way that users move around the space, it is difficult to load geometry ahead of the users the majority of the time. This gets frustrating for the user, especially when there is no geometry immediately in front of them, so it is difficult to judge speed or direction.

The external index references individual objects inside a .b3d file. The external index (a ".b3di" file) is based on a kd-tree where references to geometry nodes are stored at nodes and leaves of the tree. The references are simply the start and end bytes of the object in the .b3d file. The asset manager first reads the kd-tree file, a very quick process taking a few ms, and then loads objects in any order. The asset manager marks the kd-tree with those objects so it doesn't load any objects more than once. Currently the asset manager will read the objects in increasing distance from the camera viewpoint, and the camera viewpoint can be moved whilst the file is still streaming in. Combined with multi-processing, this means that we immediately see objects coming in from the file as soon as it starts.

A requirement to make this work is that the .b3d files must be readable in any order. Many, if not most, model formats and their loaders have side-effects resulting from the way that the underlying graphics hardware works. The most common is that if a material isn't set on an object, it automatically "inherits" the material from the most recent object that did have a material. This is true for VRML97 for example. Another common problem with formats such as OBJ, is that the whole file must be read before any object can be parsed, as there is no constraint in the file format, on a face referencing a vertex that is after it in the file. This is not to say an OBJ file can not be suitable for out of order reading; indeed we originally prototyped this idea using OBJ. However, heavily hierarchical models such as VRML97 are difficult to re-order to satisfy the constraints we need.

The two constraints we thus impose in our in-memory and .b3d format are that materials are always set in the geometry node, and all indices must be "local", i.e. without reference to indexes outside the current geometry node. The run-time then manages OpenGL state itself, by marking redundant OpenGL state changes. It also sorts on state, to optimise rendering throughput.

5 Conclusions and Future Work

The current system supports an extremely large urban model that can be explored in a interactive real-time interface. It integrates the best available information for that area.

More extensive better quality models are becoming available at the current time[1]. Demonstrations of large models with façade textures can be seen in Microsoft's Virtual Earth. Such models can be generated fairly rapidly semi-automatically using oblique aerial photography and standard photogrammetric software. More recent work has looked at generation of 3D models using street-level scanning. Our focus has been on the real-time integration and presentation of models, and we could incorporate these types of models in our system. The main issues would be the resolution of geometry interference: such as hand-crafted models not quite fitting their footprint in the base mapping data.

We are also investigating much improved texture imposters that provide some vertical texture parallax.

Acknowledgements

This work was supported in part by the projects Advanced Grid Interfaces for Environmental e-science in the Lab and in the Field (EPSRC Grant GR/R81985/01) and the GeoVue: Geographic Virtual Urban Environments (ESRC). Many thanks to Chris Parker who developed the first version of the current software. Richard Milton and Salvatorre Spinello also contributed code that was utilised.

References

[1] Hu, J., You, S., Neumann, U. (2003) *Approaches to Large-Scale Urban Modeling*, IEEE Computer Graphics and Applications, Nov/Dec,2003

[2] Hudson-Smith, A., Milton, R., Dearden, J., Batty, M. (2007) Virtual Cities: Digital Mirrors into a Recursive World. CASA Working Paper 125, available at http://www.casa.ucl.ac.uk/publications/workingPaperDetail.asp?ID=125 (Verified 20th December 2007)

[3] Laycock, R.G. and Day, A.M. (2003). Automatically generating roof models from building footprints, In WSCG, 2003

[4] Open Geospatial Consortium (2007) Geography Markup Language http://www.opengeospatial.org/standards/gml (Verified 20th December 2007)

[5] Ordnance Survey (2007) Guide to National Grid http://www.ordnancesurvey.co.uk/oswebsite/gps/information/coordinatesystemsinfo/guidetonationalgrid/index.html (Verified 20th December 2007)

[6] Ordnance Survey (2007) OS MasterMap. http://www.ordnancesurvey.co.uk/oswebsite/products/osmastermap/ (Verified 20th December 2007)

[7] Steed,A., Spinello,S., Croxford,B., Milton,R. (2004). Data Visualisation within Urban Models. Theory and Practice of Computer Graphics, Jones, M. W. (ed.) IEEE Computer Society, 9-16.