Investigating approaches to improving rendering performance of 3D city models on mobile devices

Claire ELLUL* and Julia ALTENBUCHNER

Department of Civil, Environmental and Geomatic Engineering, University College London, London, UK

(Received 13 March 2013; final version received 23 August 2013)

Three-dimensional (3D) city models have uses including on-site validation of utility infrastructure, support for augmented reality, personalized tourist information, real estate sales, and 3D pedestrian navigation. Increasingly, such applications are deployed on mobile devices, whose use is becoming more prevalent. Tablet devices are used for more professional use requiring larger screens, mobile phones for more casual users. However, many 3D city models contain hundreds of buildings, which in turn results in performance issues when attempting to visualize such models on these devices. Two issues can be identified as contributory factors – the lower specification of the mobile device itself when compared with desktop machines and the lower bandwidth network between the device and the server (3G mobile or Wi-Fi). Both of these can be addressed by reducing the volume of data in the model. To achieve this, we generalize a 2D data-set (using aggregation and simplification) and then extrude the generalized 2D maps to 3D. This minimizes the number of buildings to be transmitted over the network and processed by the on-board graphics engine. To additionally address the bandwidth issue, we make use of topological data structuring to build and transmit a minimal description for each building. Combining these approaches, we compare the results obtained for generalized and un-generalized data-sets, on a tablet and mobile device. A performance increase of between 7 and 9 times is observed.

**Keywords:** 3D GIS; generalization; OpenGL ES; mobile devices; performance

1. Introduction

The mobile phone industry has been called the industry of the decade and has over 5.2 billion subscribers globally \(^{(1)}\), of which 1.08 billion are smartphones \(^{(2)}\); when compared with 1.2 billion personal computers (PCs), it seems that smartphones are rapidly becoming the most widely accessible computing platform \(^{(1)}\). As a result of this growth in device use, the mobile application industry is predicted to be worth an estimated 25 billion dollars by 2015 \(^{(3)}\). In particular, sales of tablets are forecast to challenge those of PCs by 2017 \(^{(4)}\). This has resulted in a surge in the development of innovative mobile applications to meet these demands.

A number of such applications make use of three-dimensional (3D) city models, which are becoming more prevalent and have applications including utility infrastructure validation (“call-before-you-dig”), planning \(^{(3–7)}\), augmented reality \(^{(8)}\), personalized tourist information \(^{(by\ Schulte\ and\ Coors\ as\ stated\ in\ Ref.\ (9))}\), real estate sales, and 3D navigation \(^{(10)}\). For each of these approaches, different 3D models (emphasizing different details) may be appropriate. A detailed 3D model is not necessarily useful for in-car navigation (and could prove a distraction). However, presenting an appropriate level of 3D information is useful to pedestrian end-users, some of whom may have difficulty in understanding 2D maps and who may wish to see details of key landmarks to aid navigation.

In particular, the process of extrusion (“growing” 2D topographic mapping data to a given height) is an efficient method of creating the 3D data-sets required for many such applications, in particular where coverage should be city wide and high level of detail is not required; e.g. roof structures. It also has the advantage of integrating 3D buildings with a 2D footprint \(^{(11)}\). However, the resulting 3D data is generally quite large in volume and complex in detail \(^{(12)}\) and thus potentially difficult to visualize in its entirety, in particular on a mobile device. Given that such devices operate over a mobile (3G) network or, at best, a Wi-Fi connection \(^{(13)}\) the size of the data-set also becomes an issue for data transmission.\(^{1}\)

This paper details an investigation into the impact of 2D generalization (aggregation and simplification, carried out prior to extrusion) and topological data structuring (to reduce duplicate information) on 3D city model rendering performance on mobile devices, making use of Open Graphics Library (OpenGL) ES (a commonly used platform for 3D rendering in mobile devices) to develop an App for testing.

2. Background

Considering mobile rendering from the point of view of system architecture, two potential bottlenecks can be identified when comparing standard desktop computers.
First, data must be transmitted to the device over a lower-bandwidth network (e.g. Wi-Fi), and second the data must be rendered on the device, which itself has a lower hardware specification. Thus, to improve performance when rendering 3D city models, it is important to understand the underlying data – how it is created and structured – in order to propose alternative solutions to minimizing data size for transmission and rendering. Understanding the process used for rendering will also help to address potential performance bottlenecks. This section provides a brief introduction to these concepts.

2.1. Generating 3D city models

3D city model data-sets have varying levels of detail (LoD) (14) – ranging from LoD 0 (a digital terrain model), through LoD1 (a block model without any roof structures), and moving up to LoD4, which includes roofs, and also interior structures. Such data can be generated from digital ortho-photos, 2.5D image draping, extrusion, computer aided design (CAD) models (5), LiDAR point clouds, applications, such as PhotoSynth (15), and terrestrial laser scanning. Single sources of data have been used (16); however, it is more common to combine multiple sources of data (17–19).

While laser scanning and CAD produce detailed models of individual buildings or city blocks, methods such as extrusion (combining 2D topographic mapping with height information derived from LiDAR data) provide a rapid mechanism for generating an entire City Model to LoD1. However, the process of extrusion, in particular when making use of a detailed 2D topographic data-set, results in a relatively large data-set in terms of data volume.

2.2. Reducing data volume – generalization

Longley et al. (20) note that:

> In the interests of compressing data, it is often necessary to remove detail, fitting them into a storage device of limited capacity, processing them faster, or creating less-confusing visualizations that emphasize general trends.

This process, known as generalization, derives a map or data-set with reduced complexity and contents from a detailed source, while retaining its major semantic and structural characteristics. Steps include: classification (grouping features according to their type), aggregation (replacement of several polygons by a single polygon), and simplification (removing points to create a simpler shape). In addition, to ensure that the output of the process is fit for a specific application, exaggeration (which highlights important features in a map (21) and symbolization (replacing features with point symbols) may be applied.

Generalization algorithms are generally mature in a 2D context (22–26) and are embedded in commercial software. Work has also been carried out on generalization of 3D data-sets (6, 11, 12, 27–29) although algorithms are not yet available commercially.

2.3. Reducing data volume – topological data structures

Geographical information systems (GIS) distinguish between two approaches to modeling lines, polygons or polyhedra. In the simple features approach (“spaghetti”), each object is stored individually and data is represented as a planar configuration of points, arcs, and areas with no explicit representation of the topological interrelationships of the configuration, such as the adjacency relationships between constituent areas or volumes (20, 30). Given that each feature is stored as a separate entity, this model requires duplication of information – for example, the shared wall between two adjacent buildings will be represented twice. This increases the amount of data that is held within the model.

In a topological model, features are represented by simple primitives (Nodes, Edges, and Faces) structured using topological rules (20). There is no redundancy as the shared boundaries between objects are stored once and then combined to make up the specific objects (31) and the model provides update consistency. As shared boundaries are only represented once, data structured topologically requires less storage than that structured as spaghetti.

Figure 1 shows a simplified topological data structure for 3D data, based on a Boundary-Representation (B-Rep) approach. B-Rep describes the “shell” of each object and the basic components of the model consist of a series of 0-dimension primitives (Nodes) – which in turn contain X, Y, and Z coordinates. These Nodes are joined together to construct 1-dimensional primitives (Edges) which in turn are ordered to form 2-dimensional primitives (Faces). The Faces are linked together to form 3D polyhedra, which represent the buildings in the City Model. Note that for simplicity the volumetric object, which would permit the identification of the components of a building and of which buildings share a Face, has been omitted as the work described here focuses on visualization and on reduction of Face primitives. In general, such an object should be added to have a complete 3D model.

The process of extrusion results in a spaghetti structure, where each Face between buildings is represented twice. Details of how this data can be transformed into a topological structure are given by Ellul (33).

2.4. Understanding potential rendering bottlenecks – introducing OpenGL ES

The cross-platform and cross-language graphics application programmers interface (API) OpenGL is a software interface to graphics hardware (34). OpenGL has several specifications for various purposes. Of relevance here is
OpenGL ES (OpenGL for Embedded Systems), which is commonly implemented for mobile devices. Key features were removed from OpenGL for this implementation, in particular OpenGL ES only allows triangle-based surface primitives and excludes quad, quadstrip, or polygon primitives. Triangles make hardware algorithms simpler and faster, as they are always convex and planar.

Figure 2 shows the path of a data-set through the rendering process in OpenGL ES. The “modeling” software should first generate a triangle mesh, which defines the shapes of the objects, and its attributes, colors, and textures and collates them into an OpenGL readable structure (arrays containing the triangle points and index arrays describing how these are used to describe the objects to be rendered). The data structure enables OpenGL to reuse the same vertex for several triangles, reducing memory requirements on the device.

Once the data is correctly structured, a process of translation and scaling is required to ensure the 3D City Model is visible. The data-set is also scaled to ensure that the ratio between the X and Y extents matches the width-to-height ratio of the device (i.e. that the resulting data is not distorted). Data is then projected into a 2D coordinate system for display on screen, with an orthographic projection avoiding foreshortening by disregarding the effect of vanishing points and applying a one to one correspondence between the real world units and the pixel positions on screen. This method is widely used for engineering applications.

Following projection, a “clipping” process is used to remove any objects that will not be visible to the user (this approach is standard in any computer graphics rendering process). For example, objects too close to the user obscure the view, and objects too far away will not be shown in enough detail to be useful. The penultimate step in rendering is transforming the resulting 2D data once again to move from scaled real world coordinates into the screen coordinate system. The data is then ready for the final stage in the process – rasterization, which converts the data into pixels on screen.

The rendering process thus involves a number of structuring and transformation processes. Minimizing the quantity of data input into this process should therefore improve performance.

2.5. Previous attempts to improve 3D rendering performance

Research that has been conducted into city modelling for mobile devices is perhaps limited due to the fact that devices have only recently become powerful enough to
render 3D graphics. Indeed, mobile computation still faces various limitations, including limited Central Processing Unit (CPU) and memory, the absence or limited performance of graphics accelerators, the absence or limited performance of Floating Point Units, energy consumption issues and lack of powerful development and debugging environments.

Initial research carried out in order to overcome these limitations focussed on hardware (40). For example, Woo et al. (41) produced small graphic accelerator chips to allow high performance combined with low power consumption. A second approach proposed remote rendering architectures solutions (40, 42). Data processing and rendering takes place on a powerful computer (with performance enhanced by clustering servers and using graphics accelerators (43) and only the results are sent to the mobile client.

When considering architectures where the rendering is carried out on the mobile client, the aim is to keep the graphics as simple as possible in order to achieve satisfactory performance. Quillet et al. (44) developed a system that extracts feature lines of building facades on the server side and streams data on demand to the client. The approach forces the point of view to street level.

More general approaches to reducing the volume of data to be rendered for a 3D City Model include data compression (17), mesh simplification (45), and the use of topological data structures to remove shared and hidden internal walls in a model generated by extrusion (46). Ellul and Joubran (47) tested a similar aggregation approach to that described here. In these cases the target for rendering was not a mobile device.

3. Preparing the data

The data-set used for this research is a detailed topographic map of a 1 km square area of central London, specifically “UKMap” data provided by the GeoInformation Group. Following a first “classification” stage in which all buildings were extracted from the data, generalization was carried out (48, 49). The data-set was first aggregated with a tolerance of 1 m, minimum buildings size of 25 m and internal minimum hole size of 25 m. The maximum height for each aggregated block was then assigned to the block. Each block was then simplified using 1, 5, and 10 m simplification tolerances. This process resulted in a total of five data-sets:

- The original buildings.
- The aggregated buildings (unsimplified).
- The aggregated buildings, simplified using a 1 m tolerance.
- The aggregated buildings, simplified using a 5 m tolerance.
- The aggregated buildings, simplified using a 10 m tolerance.

All data-sets were then converted to GML and loaded into a database (Oracle Spatial 11.2 g (50)) using GoLoader software (51), and the geometry updated to ensure the polygons were correctly oriented (counter clockwise) as required for extrusion. Oracle’s inbuilt functionality was then used to extrude the polygons to the given height, and the 3D topological data structure populated. To populate the structure, an algorithm was written in Java to firstly extract each Face from the Oracle SDO_GEOMETRY objet describing each building (a building is described with 6 or more Faces). Each Face was then decomposed into corresponding Nodes and Edges, and, using a tolerance of 0.5 m, common Nodes and Edges identified and marked as shared between the various Faces. Adjacent Faces (where adjacent buildings had been extruded) were also identified and modelled. In addition to the structure described in Figure 1, a direct NODE_FACE link table was created by examining the Face geometry and identifying an ordered list of corresponding Nodes. This will support optimization of the size of the transmitted data (see: Minimising Data Volume below).

3.1. The resulting data

Figure 3 shows screen shots in 2D of three of the datasets – the base data-set, showing all polygons, the aggregated data-set (with no simplification) and the aggregated data-set with 10 m simplification. As expected, there is some loss of detail resulting from the generalization process.

Table 1 shows the impact of generalization on the resulting 3D data-set. The number of building polygons is reduced significantly by aggregation and simplification. The process of aggregation reduces the area covered by the buildings by just over 3%. However, simplification, in particular using a 5-m tolerance, results in a significant increase in the total building area.

4. Implementing the visualization algorithm

Two Android devices were selected for testing – an Acer Iconia 10 inch tablet, having a screen size of 800 by 1280 pixels, and a Dual Core, 1 GHz processor, with 1 GB of RAM, running Android Version 3.2.1; and a lower-specification LG P500 mobile phone, running Android Version 2.3.3 – 320 × 480 pixels, 3.2 inches, CPU 600 MHz ARM 11, GPU Adreno 200.

Given the use of Android devices, Java was used to develop the client-side App along with the Google Android API. PHP code was developed to extract data from the database and serve it out to the App.

4.1. Summarizing the rendering process

The first part of the test code extracts the 3D data from the database in XML format for dispatch to the Android device. On the device, a process of triangulation is required to convert the data into a structure appropriate
for rendering. The data is then re-structured as required by Open GL ES and passed through the rendering pipeline. Time measurements are taken at various stages through the process and written to the device as a CSV file in order to permit performance comparisons.

Key stages of the rendering process, optimized to improve performance, are described in more detail in the following sections.

4.2. Step 1: transmitting data – minimizing data volume using topological data structures

As data transmission between server and client is likely to be via a mobile network (potentially a 3G network, having capability of 384 kbps for slowly moving devices6), the volume of data transmitted should be reduced in size. The data-set aggregation and simplification processes described above achieve this to a certain extent by reducing the number of faces to be transmitted from 26,915 to 2058, with a corresponding reduction in Nodes. However, investigating the data structure itself reveals that the information to be transmitted can be reduced still further. As the data has been generated by extrusion, all vertical faces can be described by their lower-left and upper-right coordinate pairs. Additionally, given that the data-set in this case is LOD 1, it can be assumed that both the polygon forming the ground level of the data-set and that forming the flat roof are identical in shape, with the only difference being their height. Thus, there is no need to transmit both, provided one assumes that the base polygon has height of 0.

Beyond this, additional reductions can also be made based on the following assumptions (which result from the topological data structure creation process):

(1) Looking at the individual wall Faces: the 3D buildings are created by extrusion. This means that all vertical faces are rectangular, and made up of a total of 5 coordinate points (the first and last points are repeated). Assuming that the points are always ordered from 1 to 5 as shown in Figure 4, each Face can be described solely using Nodes 2 and 3, and the lower edge determined by setting the Z (height) value to 0 for these Nodes.

(2) Examining at the Faces surrounding the whole building: Node 3 of each Face corresponds to Node 2 of the subsequent Face (Figure 5). Thus, it is only necessary to transmit Node 3 values for each Face to be able to reconstruct the entire building for display. This approach, of course, makes the assumption that each building is fully enclosed by vertical Faces.
Describing the above building using all coordinate values and individual Faces requires a total of 44 coordinate triplets (5 triplets for each of the 6 faces, and 7 triplets for the base and the roof). Describing the building as a whole, rather than on a Face by Face basis, reduces the amount of data to be transmitted to 6 coordinate triplets (Node 3 for each Face).

Taking the above strategies into account, PHP was used to query the database and extract three lists of information, which were then transmitted sequentially as XML.

- A full list of Node 3 coordinate values (X, Y, and Z) for all buildings (note that it is only the Node 3 values that are transmitted).
- A list detailing each building and the number of Nodes associated with it – to be used when parsing the list of “Node” coordinate values.
- The bounding volume of the data-set (for use in rendering transformations and clipping).

### 4.3. Step 2: Restructuring the roof faces – triangulating by Ear Clipping

Both the spaghetti and topological model for 3D data storage in a database make use of polygon Faces to describe each feature and as described above the third node of each Face is extracted for simplified data transmission. However, as OpenGL ES does not render polygons directly, each transmitted Face should be converted to a triangular mesh prior to rendering.

The “Ear Clipping” triangulation algorithm is a simple and robust algorithm which works well for geospatial polygons (36). It is underpinned by an assumption that the polygons which represent the building roofs in spatial data-sets are “simple polygons” (i.e. do not have internal holes or self-intersections). An “ear” of a polygon is defined as the edges between three consecutive nodes, \( n_i - 1, n_i, n_i + 1 \), where the tip of the ear \( n_i \) is convex,8 which is the case when the interior angle between those edges is less than 180°. Additionally, the algorithm also determines whether any other points from the face lie within the ear. It iterates sequentially through the Nodes describing each Face in sequence to identify ears which meet both stated criteria. If an ear is found, the three consecutive Nodes are stored as a new triangle and the tip of the ear, \( n_i \) is removed from the current Face. This process is continued until only three Nodes remain, which form the final triangle (36) (Figure 6). The theoretical worst-case runtime, the Ear Clipping algorithm, is \( O(n^3) \) (36).

### 4.4. Step 3: restructuring wall data – triangulating the “3rd Node” lists

OpenGL ES requires triangular primitives for rendering. However, each vertical wall has been generated by an extrusion process and is therefore represented by a rectangle. Given that the walls, which belong to the same building, all have the same height, they can be represented as a triangle strip (Figure 7). This approach takes advantage of the array structures used to reduce memory requirements for OpenGL ES.

### 4.5. Step 4: rendering the data

The projection used for the source UK Map data-set described above is British National Grid, which is an orthographic projection. To transform these coordinates into values appropriate for display, the initial screen coordinate system was set up with its origin (0, 0, 0) in the center of the viewport. The coordinate values of the imported buildings (in British National Grid) are far off to the northeast of the viewport. Transformation is carried out by finding the center point of each axis for the data-set (e.g. \( (x_{\text{max}} + x_{\text{min}})/2 \)) and then translating the data-set by the respective distance along the negative x and y. The z-axis is shifted upwards.

For the second and third steps of the rendering process, OpenGL ES uses the bounding coordinates of the data-set to project the 3D triangles into 2D space and to set clipping boundaries, which in the case of this project are set to the extents of the data-set. The near and far clipping planes were set to 1000 and \(-1000\) in order to avoid clipping when the user pans the objects in the z-direction. Projection is set to be orthographic.

### 5. Performance test results

In total, 10 rendering time measurements were carried out using each device for each of the five test data-sets. The results were then averaged for each data-set. Figure 8 shows the resulting data (full, aggregated, and aggregated and simplified using 10 m tolerance), with more detail in Figures 9 and 10.

Examining the images above (Figures 9 and 10) in more detail highlights a number of small problems. For example, there appear to be errors in the triangulation algorithm for the aggregated, 1 m simplified data-set.

---

8. In the context of this algorithm, an “ear” is defined as an edge where the interior angle is less than 180°.
This is perhaps due to the complexity of the resulting aggregated buildings (Figure 3). Indeed, these errors do not appear in the 10 m simplified data-set, where the roof triangulation matches the underlying building structure. In addition, a number of buildings are not shown in the aggregated and simplified data-sets. This is due to two factors – buildings with a footprint of smaller than 25 m² were eliminated by the aggregation process. The complexity of other aggregated buildings caused problems with the extrusion process, resulting in multi-part extrusions which were not well handled by the process used to create the Faces that underpin rendering (Figure 11).

Figure 6. The ear clipping algorithm.

Figure 7. Using meshes to render building walls.

Figure 8. The full extruded data-set (left), aggregated and no simplification (center), aggregated with 10 m simplification (right).

Figure 9. Two extracts from the 3D rendering of the complete data-set (no aggregation or simplification).
Three roof colors (red, blue, and yellow) can be observed. This is due to a problem encountered with the predefined array sizes in OpenGL ES—which limit the number of values that can be stored in one index array limited to 32,767 (array type “short”). The index array points to each x, y, and z value of each triangle, requiring a total of 130,000 required index values for the full data-set and 51,000 for the aggregated data-set. Thus, the software was developed to split the triangles into three separate arrays, coloring the roofs accordingly. This issue does not arise for the walls, which make use of triangle meshes rather than individual triangles, and hence take advantage of shared nodes between the triangles.

5.1. Rendering results – performance

Figure 12 shows the overall display time for each data-set (the method used to capture the results is described in Implementing the Visualization Algorithm above). As expected, this decreases as the complexity of the data-set decreases. In particular, a significant drop can be observed due to aggregation (without simplification) for both devices (2.74 times as fast for the LG device and 2.97 times as fast for the Acer). Performance for the aggregated, 10 m simplification data-set is 7.06 times as fast for the LG and 9.54 times as fast for the Acer. Moving from aggregated data to the 10 m simplification results in performance 2.36 times as fast for the LG and 3.2 times as fast for the Acer. Table 2 gives the average results (mean) for each test on each device, in milli-seconds.
For the LG phone, two elements of the process take up the most significant amount of time – XML parsing and overall mesh setup (including triangulation), with the XML parsing taking an average of 55% of the time and the mesh setup an average of 39%. This is significantly different for the Acer device, with its higher specification – where the XML parsing takes up an average of approximately 14% of the overall time, with the mesh setup taking 74% of the time. In both cases, the triangulation time is on average between 20 and 30% of the overall mesh setup time – 22% for the LG device and 29% for the Acer. This illustrates that although there are a relatively large number of Nodes, the actual triangulation is fairly limited in scope due to the fact that the system handles individual buildings rather than a large single mesh; for example, the average number of Nodes on a specific roof face is 6.74 for the Aggregated data-set and the maximum is 199 Nodes.

Figure 13 shows the overall display time vs. the number of roof triangles and overall display time vs. the number of coordinates describing the buildings.

For the LG phone, two elements of the process take up the most significant amount of time – XML parsing and overall mesh setup (including triangulation), with the XML parsing taking an average of 55% of the time and the mesh setup an average of 39%. This is significantly different for the Acer device, with its higher specification – where the XML parsing takes up an average of approximately 14% of the overall time, with the mesh setup taking 74% of the time. In both cases, the triangulation time is on average between 20 and 30% of the overall mesh setup time – 22% for the LG device and 29% for the Acer. This illustrates that although there are a relatively large number of Nodes, the actual triangulation is fairly limited in scope due to the fact that the system handles individual buildings rather than a large single mesh; for example, the average number of Nodes on a specific roof face is 6.74 for the Aggregated data-set and the maximum is 199 Nodes.

Figure 13 shows the overall display time vs. the number of roof triangles, with a best fit linear equation. In both cases, as expected, the variance of the result ($R^2$) is very high – approximately 0.99, indicating that performance degradation is linear as the number of triangles increases, providing useful predictive information.

6. Discussion and future work

Generalization of a 2D data-set prior to extrusion into 3D is not new. However, when coupled with the new methods presented here to model data topologically, transmit a minimum description of the data-set to a mobile device, and implement triangulation on that device to result in lossless 3D rendering, the performance gains to be made combining generalization with these approaches are clear and significant. Although, in the case of the Acer tablet, the full test data-set rendered in a fairly acceptable 30 s, this was reduced to close to 3 s (i.e. 1/10th of the time) through the generalization process. For the lower-specification LG phone, the initial rendering time for the full data-set was not acceptable at all, at 212 s. The aggregation and simplification process brought this down to a more acceptable, although not ideal, 32 s.

A number of issues were encountered during the data preparation process which meant that some buildings were not included in the resulting data-sets. Although it is important that these are resolved in order to generate a valid city model, it is not felt that the inclusion of the buildings would invalidate the overall trends demonstrated above.

Overall, therefore, these results highlight the importance of the aggregation and simplification process in order to obtain reasonably acceptable performance on lower specification devices, as well as suggesting that a far larger or more detailed data-set could potentially be rendered directly to the tablet if required. In particular, the transmission of a single Node (Node 3) for each Face achieves lossless results. On the small sample of tests executed, linear performance degradation was noted, which may provide useful insight into the potential rendering time for other 3D city models using the approach described. The importance of hardware specification was also highlighted, not only to improve performance for the overall task, but also for specific issues relating to parsing the XML data to extract the required coordinate information.

The adequacy of 3D detail, the visual impact of the resulting 3D data-set, the suitability of the response times- and the overall usability of the 3D model is perhaps more subjective and will depend on the specific application- for which the 3D City model is to be used – i.e. the context of use (52). For example, definitions such

| Data-set | Parse XML | Triangulation | Setup mesh | Draw mesh | Total time |
|----------|-----------|---------------|------------|-----------|------------|
| All Data – LG | 112,576.1 | 23,073.1 | 94,918.8 | 212.5 | 212,226.3 |
| Aggregated data – LG | 39,131.3 | 6923.0 | 34,135.7 | 171.6 | 77,465.2 |
| Agg. Simp1 m – LG | 26,258.9 | 4006.3 | 19,453.9 | 173.8 | 47,791.7 |
| UCL Agg Simp5 LG | 18,754.6 | 2715.1 | 11,652.9 | 179.2 | 32,832 |
| UCL Agg Simp10 LG | 17,824.4 | 2237.7 | 9406.2 | 183.7 | 30,103.3 |
| UCL All Acer | 4281.4 | 6819.8 | 22,930.7 | 2.5 | 31,060.1 |
| UCL Agg Acer | 1391.2 | 2285.5 | 8135.3 | 3.9 | 10,464.6 |
| UCL Agg Simp1 Acer | 749.9 | 1256.7 | 4316.4 | 2.7 | 5757 |
| UCL Agg Simp5 Acer | 538.4 | 801.7 | 2738.6 | 2.3 | 3803.7 |
| UCL Agg Simp10 Acer | 481.8 | 673.2 | 2270.1 | 2.6 | 3255.7 |

\(a^{\text{Includes triangulation time}}\)

\(b^{\text{End-to-end execution time, from App launch to end of the rendering process}}\)

\[
y = 3.777x + 9482 \\
R^2 = 0.9988
\]

\[
y = 2.103x + 90.081 \\
R^2 = 0.9929
\]
as “an application should respond within 2 s to provide users with a feeling of interactivity” cannot be applied universally, and in mobile applications 2 s is too long for an application that communicates with a driver (53). The scale at which the data is visualized is a second factor to consider. The source data-set, UKMap, contains a very high level of detail with buildings sub-divided into smaller elements where roof height varies. This detail may be useful, for example, in a planning context, for large-scale mapping. However, when a larger extent of the data-set is viewed (Figure 8) the detail perhaps appears to “clutter” the map and the more simple lines of the aggregated and simplified data-set could be said to be more visually appealing, without significant loss of detail or general shape of the overall building. This may, therefore, be suitable for visualizing data at a smaller scale – for example, 3D air quality distribution across a city. Generalization concepts, including exaggeration and symbolization, are also important in this context.

A number of directions for future work can be identified. In order to overcome the issue with the OpenGL index arrays, the triangulation process should be structured to take greater advantage of the shared nodes between the individual triangles. In addition, further consideration should be given to adding 2D topographic data for roads, parks, and pavements, as well as 3D building textures and/or adding images, to the resulting city model, to give a more realistic visualization, and hence rendering time. The bandwidth of the Wi-Fi network used for testing should be taken into account to observe the differences between the results obtained when using a 3G network, home Wi-Fi and on-street Wi-Fi.

Additionally, server-side data structuring could be investigated, given that data structuring (XML parsing and placing the data into the required OpenGL arrays) forms the majority of the performance overhead. It may also be possible to dynamically query the data using a spatial query to extract data for the area of interest as the user zooms and pans around the map rather than render the entire data-set. Aggregated or disaggregated data could be streamed depending on the extent of the requested data. It is also important to identify the impact of additional LoD on performance – for example, what additional overhead does adding detailed 3D roof create?

In conclusion, the work described in this paper illustrates the potential for rendering relatively extensive and/or detailed 3D city models on mobile devices, making use of two well-known GIS techniques – topological data structures and generalization. It also highlights the importance of further research into usability, context, and both 2D and 3D generalization, in order to provide Apps that are fit for purpose and meet the demands fuelled by the growth in the mobile market.

Acknowledgments
The authors would like to thank the reviewers of this paper for their very helpful feedback. In addition, we would like to thank the GeoInformation Group for the provision of the UKMap data-set. Thanks also go to the team at Snowflake Ltd. for the GoLoader software, and to Intergraph for Geomedia Professional.

Notes
1. While Wi-Fi connectivity is increasing in potential (with speeds up to 600 MB/s promised (13)), such technology is not currently available to general end-users or the public Wi-Fi portals available in situations where 3D Apps are likely to be deployed (Smith 2013 notes that most homes will be limited to 300 MB/s, contrasting with Ethernet of 1000 MB/s).
2. WebGL is the equivalent to OpenGL ES for creating 3D graphics on the web.
3. UKMap, from The GeoInformation Group, www.geoinformationgroup.co.uk/, which includes height information for each building polygon.
4. Note that the aggregation process executed here yields a similar result to the removal of internal “walls” as described in Ref. (46).
5. Note that for this research the bandwidth of the network used was not measured, although transmission time from query submission to data receipt on the device has been logged as part of the overall display time shown in Table 2.
6. http://www.silicon-press.com/briefs/brief.3g/ (accessed April 4, 2013).
7. Note that floor data has not been included in the experiments described here, as it is hidden from the user’s view when a City Model is rendered.
8. To check whether a potential ear is convex, the cross product between the two vectors along its outside is used.

Notes on contributors
Claire Ellul completed her PhD in 3D GIS at the University College London in 2007, and is now working as a lecturer. Her 3D research relates to improving visualization and query performance of very large 3D data-sets and City Models.

Julia Altenbuchner was awarded a Master of Science in GIS at the University College London in 2012. She is currently working on her PhD as a member of the Extreme Citizen Science research group (UCL) focusing on the development of GIS tools for non-literate communities.

References
(1) Ahonen, T. All the numbers, All the facts on mobile the trillion-dollar industry. Why is Google saying: Put your best people on mobile? (Communities Dominate Brands Blog). http://communities-dominate.blogs.com/brands/2011/02/all-the-numbers-all-the-facts-on-mobile-the-trillion-dollar-industry-why-is-google-saying-put-your-b.html (accessed Nov 7, 2012).
(2) Alexander, A. Smartphone usage statistics. http://ansonalex.com/infographics/smartphone-usage-statistics-2012-infographic/ (accessed Aug 4, 2012).
(3) Markets and Markets (M&M). World mobile applications market – Advanced technologies. http://www.marketsandmarkets.com/Market-Reports/mobile-applications-228.html (accessed Aug 12, 2012).
(4) Arthur, C. Tablets will challenge PC sales by 2017 as Android passes iPad, says IDC. http://www.guardian.co.uk/technology/2013/mar/13/tablets-challenge-pc-sales-2017-android (accessed April 13, 2013).
(43) Lamberti, F.; Zunino, C.; Sanna, A.; Fiume, A. An Accelerated Remote Graphics Architecture for PDAS; ACM Press: New York, 2003; pp 55–61.

(44) Quillet, J.-C.; Thomas, G.; Granier, X.; Guitton, P.; Marvie, J.-E. Using Expressive Rendering for Remote Visualization of Large City Models. Proceedings of the 11th International Conference on 3D Web Technology, Columbia, MD, 2006; pp 27–35.

(45) Sester, M. 3D Visualization and Generalization. In Photogrammetric Week 2007; Fritsch, D. Ed.; Wichmann: Stuttgart, 2007; pp 3–9.

(46) Ellul, C. Can Topological Pre-Culling of Faces Improve Rendering Performance of City Models in Google Earth? In Progress and Trends in 3D GeoInformation Sciences (Lecture Notes in Geoinformation and Cartography); Pouliot, J., Daniel, S., Hubert, F., Zamyadi, A., Eds., Vol. 47; Springer: Berlin, 2013; pp 133–154.

(47) Ellul, C.; Joubran, J. Preliminary Investigations into the Potential of Improving Rendering Performance of 3D Datasets Using 2D Generalisation. Usage, Usability, and Utility of 3D City Models, 2012; pp 03005. http://3u3d.edpsciences.org/articles/3u3d/pdf/2012/01/3u3d_03005.pdf (accessed Dec 7, 2013).

(48) ESRI. Aggregating polygons (data management). http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?topicname=Aggregate_Polygons_(Data_Management) (accessed Aug 15, 2012).

(49) ESRI. ArcGIS for desktop. http://www.esri.com/software/arcgis/arcgis-for-desktop (accessed Oct 15, 2012).

(50) Oracle. Oracle spatial SDO_UTIL.EXTRUDE. http://docs.oracle.com/cd/B28359_01/appdev.111/b28400/sdo_util.htm (accessed Oct 10, 2012).

(51) Snowflake. Go loader – Load GML into your database of choice. http://www.snowflakesoftware.com/products/goloader/features/ (accessed Aug 15, 2012).

(52) Kjeldskov, J.; Graham, C. A Review of Mobile HCI Research Methods. In Human-Computer Interaction with Mobile Devices and Services (Lecture Notes in Computer Science) Chittaro, L., Ed., Vol. 2795; Springer: Berlin, 2003; pp 317–335.

(53) Marsh, S.; Haklay, M. Valuation and Deployment. In Interacting with Geospatial Technologies: Haklay, M., Ed.; Wiley: Hoboken, NJ, 2010; pp 199–222.