Sixth Irish Workshop on Computer Graphics (2005)
Eurographics Irish Chapter

Geometry Reduction for Urban Simulation on Handheld Devices

A. Brosnan, J. Hamill and C. O'Sullivan

Image Synthesis Group, Trinity College Dublin, Ireland

Abstract

We present a real-time urban simulation on a Personal Digital Assistant (PDA). An existing desktop urban simulation is used to automatically generate potential visibility data, sub-divide the world into areas of similar potential visibility, and generate imposter images for complex and distant objects; all of which are used to achieve interactive frame-rates on a handheld device.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Display Algorithms; I.3.7 [Computer Graphics]: Virtual reality

1. Introduction

Simulating detailed urban environments in real-time requires that the geometric complexity of the scene be reduced before rendering. If all the geometry in the world is to be rendered, interactive frame-rates will not be achieved on a standard desktop PC. The need to cull geometry from the scene is even greater when the simulation is to be run on a handheld device, such as a Personal Digital Assistant (PDA) or mobile phone. These devices have considerably less computational power than desktop PCs, and many do not have dedicated graphics acceleration hardware. They also have limited memory for storage of geometric models and textures. For these reasons aggressive geometry culling is a necessity when displaying city walkthroughs on handheld platforms. Many culling techniques require considerable run-time processing. PDAs have limited computational power, besides their limited graphical capabilities; pre-processing techniques which cull geometry with minimal run-time computation are preferable.

In this paper we describe the generation of Potential Visibility (PV) data for an urban environment, tailored for use on a handheld device. The PV data defines the subset of all objects in the virtual world which are visible from particular areas in that world. We describe the use of this information to achieve interactive frame-rates in an urban simulation on a mobile platform. The PV data is used to sub-divide the world into *sections*. Associated with each section is a limited number of objects (models of buildings in a city), each of which is visible from some or all points within that section. For objects visible but distant, billboard imposters are generated. Each imposter is comprised of a simple quad and texture, used to replace a 3-D model and thus speed up rendering. The PV data, sections and imposters are generated at a pre-processing step, to limit the amount of computation needed at run-time. They are all generated automatically from an existing, PC-based urban simulation. An automated system to generate the data is far preferable to generating it manually, which would be laborious and inaccurate. Furthermore, the automated process allows easy re-building of the data when the virtual world is enlarged, or otherwise modified to keep pace with the real city on which it is modeled. Section 6 of this paper details the results of this endeavour, a real-time urban simulation on a PDA.

2. Background

We describe general geometry culling techniques; the desktop urban simulation used to generate data for the handheld system; earlier work on the PDA urban simulation, and techniques used in other handheld systems.

2.1. Potential Visibility & Geometry Culling

Much work has been done in the area of geometry culling for real-time walkthroughs, games and other applications. Removal of unnecessary geometry before rendering is essential in order to achieve interactive frame-rates in de-

tailed, 3-Dimensional virtual worlds. Run-time culling techniques include backface-culling (removal of polygons that face away from the camera), frustum-culling (removal of polygons outside the camera view frustum), and occlusion culling (removal of polygons obscured by other polygons). Another technique involves the use of Potentially Visible Sets (PVS). PVS, often pre-computed, describe those objects in the world which can be seen from a particular point or area within the world.

In [TS91], Teller et al. describe a system of visibility pre-processing for building walkthroughs. An axis-aligned architectural model is divided into rectangular *cells*, corresponding roughly to the rooms of the building. The boundaries of the cells coincide with the walls of the rooms. The walls act as occluders, obscuring polygons behind them. Thus only a small number of cells are visible at any given time — the cell in which the camera is positioned, and nearby cells visible through doorways and other *portals* in the world. Portals are non-opaque cell boundaries. They are used to produce an *adjacency graph* which describes the inter-connectivity of the cells. As a final pre-processing step, cell-to-cell visibility is computed by examining which pairs of cells have unobstructed lines of sight between them. Thus a data structure is created which sub-divides the world into cells, and describes which other cells are visible from a given cell. The geometry in the current and other visible cells constitutes the potentially visible set of polygons.

Only the appropriate cells are considered for rendering at run-time, providing significant geometry culling and hence improved frame-rate. A further reduction in geometry is achieved at run-time by excluding cells not in the camera's view frustum; for example a cell on the other side of a doorway the viewer has just entered through. The above-mentioned pre-processing and run-time culling techniques are used by Teller et al. for a walkthrough of a single architectural model, *i.e.* indoors. However, similar techniques can be employed in a cityscape where tall buildings surround the viewer, occluding other buildings in much the same way that walls within a building occlude other rooms in that building.

Airey et al. [ARB90] also pre-compute potential visibility data for a single, complex building. In addition, they strike a balance between visual quality and frame-rate by using low-level-of-detail polygonal meshes when the viewer is moving; then switching to higher-detail meshes when the viewer is stationary, and the scene static.

2.2. Desktop Urban Simulation

In [HO03], Hamill et al. describe Virtual Dublin, a desktop urban simulation in which the user navigates around a recreation of Dublin City Centre. Figure 1 is a screenshot from Virtual Dublin, showing the Front Square of Trinity College. Virtual Dublin is a detailed 3-D environment, running at 60 frames per second (fps) on a desktop PC. In [DHOO05], Dobbyn et al. enhance this virtual city by the addition of animated virtual humans. Virtual Dublin uses a variety of run-time culling techniques to reduce the number of buildings drawn at each frame and thus achieve interactive frame-rates.

Figure 1: *Virtual Dublin desktop urban simulation*

Firstly, buildings outside of the view frustum, *i.e.* those out-of-view behind or to the side of the camera, are culled. This subset of the city's buildings is further reduced by a method of occlusion culling. All buildings in the view frustum are drawn to an occlusion buffer. Each building is drawn to this buffer in a single unique colour. If that colour is found when the buffer is scanned, the building is not occluded by other buildings and should be drawn to the screen. The additional occlusion-draw is much faster than a full draw, as the resolution is reduced, and lighting and texturing are not used; therefore, the speed-up achieved by the resultant culling outweighs the time taken for the occlusion-draw. By these means a list of visible buildings is determined for the current camera position and orientation — these and only these buildings are rendered. In addition, OpenGL provides a simple run-time backface cull, removing those polygons facing away from the viewer.

Imposters are used to replace some of the visible geometric models, providing further increases in frame-rate. An imposter is a 2-Dimensional entity consisting of a single quad with a texture bound to it. The quad adjusts to face the camera — the imposter is like a rotating billboard. Different textures are used depending on where the camera is with respect to the building. From distance the loss of detail when the imposter replaces the geometric model is imperceptible, so imposters are often used for background objects, while 3-D models appear in the foreground. Some systems create textures from all angles around the object as a pre-processing step, and select the appropriate one at run-time; this requires a large amount of memory to store all the imposter textures. However, Virtual Dublin generates the imposter textures at run-time, and the speed-up is achieved by re-using the same texture for a number of frames. This method works

(a) (b) (c)

Figure 2: *Successive screenshots*

best when the camera is stationary, but can be used when the camera is moving, so long as a new texture is created when the old one becomes obsolete; this happens when the change in angle between camera and object reaches a certain value, and the inaccuracy of the old texture becomes apparent to the viewer (Hamill et al. examine the perceptibility thresholds in [HMDO05]).

Jeschke et al. [JWSP05] use visibility culling, geometric simplification (substitution of less-detailed polygonal meshes), and imposters to achieve interactive frame-rates in a desktop urban simulation. The amount of memory required to store their imposters is low, because they use the same imposter texture from a variety of positions, and create imposter textures for groups of adjacent objects rather than individual textures for each object.

2.3. Handheld Urban Simulation

Initial work on the mobile version of Virtual Dublin was done by Rossi et al. [RCO03]. At pre-processing, the world is sub-divided into small areas, and the appropriate buildings are assigned to each area. At run-time the designated buildings are displayed. This is a Potential Visibility (PV) technique similar to the one described in this paper (see Subsection 6.1). However, they generate the PV data and section boundaries manually. This technique cannot reasonably be applied to a large, ever-expanding virtual city. In addition to the work being tedious and time-consuming, the resulting PV data would most-likely be less accurate than if a rigorous automated technique were used. Furthermore, no attempt is made to optimise the sub-division. The research described in this paper is motivated by the need for automatic generation and re-generation of data, as well as the desire for a better, if not absolutely optimal, sub-division. Rossi et al. describe the distributed, multi-user aspect of the handheld system. The building models are downloaded to multiple client devices (handheld or otherwise). These devices communicate with each other via the server to coordinate their positions in the world (a dead reckoning algorithm is used to reduce the associated network traffic).

Other work on handheld (or other graphically limited) platforms use image-based rendering [CG02], point-based rendering [DD04] and other geometry reduction techniques, at the expense of reduced visual fidelity. In [Cum03], Cummins provides an overview of culling techniques and rendering speed-ups suitable for use on handheld devices. In this paper we attempt to *avoid* unnecessary geometry instead of reducing the geometric complexity, and visual quality, of each model. We calculate (with minimal run-time computation) which geometric models need to be rendered, and render most of them at full resolution. Low-resolution and imposter-based models do not look as good as high-resolution geometry, so we prefer to identify salient buildings (those in view and relatively close), and render them fully.

3. Generation of PV data

Virtual Dublin is a desktop PC urban simulation. Users of this program can navigate a 3-D simulation of Dublin City Centre in real-time. Virtual Dublin was used to generate Potential Visibility (PV) data for the handheld system, as follows.

The virtual world was built using a standard Cartesian coordinate system. A 2-D grid was laid over the world, dividing it into squares 2 virtual metres along each side. The squares run parallel to the x- (east-west) and z- (north-south) axes. For each intersection point on the grid, the PV was calculated. The PV calculation was achieved by placing the camera at the point in question, spinning the camera through 360 degrees, and recording those objects (buildings) displayed by Virtual Dublin during the spin. This provides a complete list of buildings visible from the given point, irrespective of the direction in which the camera was pointing. Given initial parameters defining the target area of the world (usually the entire city), the system automatically generates the PV data for every sample point in the target area. Figure 2 shows three screenshots (a), (b) and (c) from Virtual Dublin, all taken from the same position, with the camera rotated a fixed distance after each shot. A full 360 degree revolution

on the same point will display all buildings potentially visible from that point.

As mentioned in Subsection 2.2, Virtual Dublin uses frustum- and occlusion-culling at run-time. Within the Virtual Dublin *Draw()* function an array is created of buildings not culled by the above methods. Identification numbers for these buildings are put output to a data structure, and this data structure stores the PV data for each sample point in the world. As mentioned above, the distance between sample points (in both north-south and east-west directions) is equivalent to 2 metres in real-world terms. A smaller distance would naturally require a larger data structure. No errors are observed as a result of the sampling — *i.e.*, there are no buildings which should be visible from a position between two sample points, but which are not visible from the sample points themselves.

4. Generation of Sections

Given the Potential Visibility (PV) data, describing which buildings are visible from every point within the world, it is desirable to group these points into *sections*. A section of the virtual world is a 2-D area from which a known subset of the virtual city's buildings are visible. Only those buildings are rendered as the viewer moves within that section.

A user of the handheld urban simulation will move between sections as they navigate around the world. When (or preferably before) the user changes section, the newly required building models are downloaded to the mobile device. It is desirable to minimise the run-time network traffic on low-bandwidth, handheld devices. This would involve having geographically large sections, containing a large number of buildings. The viewer could explore large areas of the world without changing sections and downloading new models. From this point of view, the ideal section covers the whole virtual city, and after a large download at start-up no further bandwidth would be required. However the handheld devices are also constrained by memory limitations. They cannot store the entire world in memory at one time. From this perspective, geographically small sections — containing only the few buildings visible from the current location — are desirable. Also, as sections are increased in size, redundancy appears — buildings visible from some but not all points in the section, which will be drawn irrespective of where the viewer is in that section. From this point of view, the ideal section is a single point.

A balance must be found. Sections should neither be so large that the associated buildings cannot be stored in memory, or that excessive redundancy is introduced; nor so small that new buildings must continually be downloaded, and other overheads incurred. The algorithm used to generate the section boundaries is presented below.

For a given point p, with Potential Visibility PV_p, a set s is found of all other points with PV_p, or whose PV is a subset of

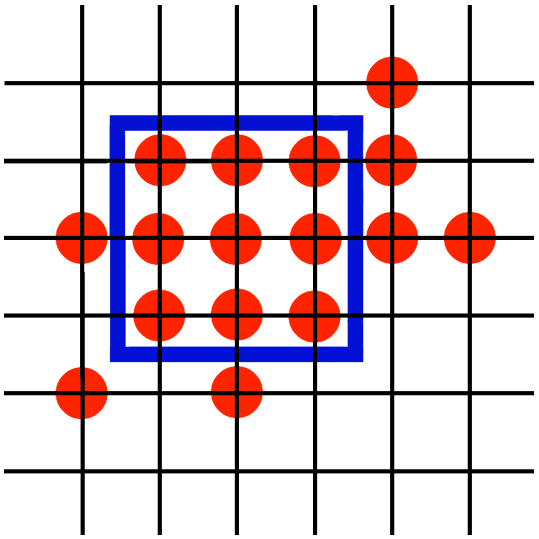

Figure 3: *Inner rectangle (blue) of points s (red)*

PV_p. From this irregularly shaped set s, the *inner rectangle* is extracted. The inner rectangle is a lattice-like rectangular group of sample points with no internal gaps. See Figure 3. The red points are the list s, and those in the blue box make up the inner rectangle — a contiguous, rectangular subset of s. Strictly rectangular sections are used because this simplifies the run-time calculation of which section the camera is in (of which more anon). The inner rectangle forms the basis for the section — it is subsequently expanded, retaining the rectangular shape, according to the following condition. If the section can be expanded on any side with the addition of no more than a threshold number of buildings to the Potential Visibility PV_s of the section, the expansion is allowed. This process is repeated until the section cannot be expanded any further. When multiple inner rectangles are available, the one that produces the largest section is chosen.

So long as the section achieves a minimum size, it is accepted, and all the points included in it are removed from the list of unassigned points. The section generation process is repeated for all points not already assigned to a section. If points remain which have not been assigned, the minimum size restriction for sections is relaxed and all points are assigned to a section. Sections can overlap, but will not do so unnecessarily; if a section expansion only adds points already assigned to other sections, the expansion is not performed.

Once the full extents of the final section are known, the boundaries of the sections (which will be axis-aligned) are computed, and the PV data aggregated for all points in the section. Thus the section is represented by its maximum and minimum x- and z- coordinates (describing a rectangle in the world) and an associated list of building models. It should be

noted that the geographic area of a section need not include the buildings assigned to it — merely points from which the buildings are visible. The final set of sections will vary depending on the order in which the section generation process is performed on the sample points — refinement of the algorithm to generate the best possible set of sections is left to future work. We take advantage of the fact that adjacent sections contain many of the same buildings, thus reducing the number of new buildings that must be downloaded when a new section is entered. If sufficient memory is available, buildings for multiple sections are stored. Thus thrashing does not occur as the viewer moves back and forth between two sections.

5. Generation of Imposters

The Potential Visibility (PV) data, generated as described in Section 3, details the set of building models visible from each sample point in the world. Some of these buildings will be very close to the viewer; others are far away, or mostly occluded, or both. If a building is barely visible, it is wasteful to perform a full rendering of the geometric model. We replace far-away buildings with imposters. Some buildings feature such a high polygon-count that they cause an unacceptable loss of frame-rate when displayed on a low-power, handheld device. These buildings are also replaced by imposters, irrespective of their distance from the camera. In all cases the imposters textures are pre-rendered. At run-time the appropriate images are chosen, based on the angle between the camera and the building; and the imposter quad is oriented toward the camera.

5.1. Distant-Building Imposters

If a building b, assigned to a section s, is at least a distance d from all points along the boundary of a section, building b is drawn from section s as an imposter. The Potential Visibility data is amended to reflect this. In many cases a single imposter texture will suffice from any point in s — *i.e.*, the angle between the camera position and b does not change sufficiently to invalidate the texture, unless the viewer moves to another section. However, if the section is sufficiently large that multiple imposter images are required, the appropriate image is selected at run-time. Note that it is not sufficient to check that b is at least distance d from the corners of section s — if s is wide, and b is roughly equidistant from the two extremes of s, b may be far from the corners but close to a point half-way along the width of the section boundary.

The imposter textures are generated as follows. Using the desktop Virtual Dublin system, the building is isolated against a black background and rendered, producing a 128×128-pixel texture. This is the maximum texture size allowed by the PocketGL API (PocketGl is a subset of OpenGL for PocketPC handheld devices). If necessary the building is rotated and the process repeated. The PV data determines what angle or angles the building needs to be rendered from, for a particular section. Multiple sections may require the same or very similar imposter images. If this is the case, only one texture need be stored on the PDA. The PV data can be manually tweaked so that sections requiring very similar imposters (textures rendered from almost identical angles) use the same imposter. In this way, the minimum of imposter textures are stored on the low-memory handheld device. At run-time, transparency in the imposter images is provided by the API, so the black background does not appear.

Figure 4: *Imposter of complex building*

5.2. Complex-Building Imposters

Imposters are generated in a similar way for buildings too polygonally complex to render in real-time on the mobile platform. An example is the model of Trinity College's Campanile, which weighs in at $12,000$ polygons. In this case imposters are needed from all angles around the building, and the imposter is always used in place of geometry. Ideally a higher-resolution texture would be used at close range, but as mentioned above the PocketGl API limits textures to 128×128-pixels. However the result is tolerable. See Figure 4 for the imposter image (at a higher resolution) and Figures 5 and 7 for the imposter in use in the handheld urban simulation.

6. Results

The PDA used is a HP iPaq hx4700, which features a 480×640 VGA display. The device does not have graphics acceleration hardware. The urban simulation runs at approximately 20 fps as the user navigates through the city, which covers a number of square kilometres of Dublin City Centre.

Figure 5: *Imposter (in foreground) in use on the PDA*

Figure 6: *Geometric model on PDA*

6.1. Sections

The sections, generated from the Potential Visibility data, are defined by their limits (maximum and minimum x- and z-coordinates) and a list of buildings. Complex and distant buildings are flagged to be drawn as imposters. For the complex buildings, and some of the distant buildings, multiple images are available — this fact is denoted by a flag. All this data is put out to a text file, and is then inserted into SQL database tables. The SQL database is used at run-time to calculate which models should be sent to the handheld device, depending on which section the viewer is in. The PDA determines if the user has switched sections by a simple test of the camera position against the limits of the section — the current section has changed if the camera position's x-coordinate is greater than the section's maximum x-coordinate (or less than the minimum coordinate, or similarly for the z-coordinate). The low computational complexity of this method is the advantage of axis-aligned, rectangular sections. When required, buildings for the new section are sent from a PC server, via WiFi wireless LAN, to the device. The usefulness of the section sub-division depends on the topology of the city — it is more useful in long narrow streets than in wide open spaces. However interactive frame-rates are achieved irrespective of the composition of the current section.

Figure 7: *Geometric models and imposter (in foreground) on PDA*

6.2. Imposters

The imposters used to replace distant buildings are indistinguishable from their geometric counterparts. The visual quality of the imposters used to replace complex buildings is less agreeable, but not wholly unacceptable. With a more advanced graphics API, larger textures could be used to good effect. The complex-building imposters can be seen in Figures 5 and 7.

7. Conclusions and Future Work

We conclude that pre-processing of Potential Visibility (PV) data, and fast use of this data at run-time, allied to the judicious use of imposters, enables interactive urban simulation on handheld devices.

Future work will involve optimising the sections. We wish to determine the ideal section size, at which the trade-off between section-switching overhead, and redundancy, is optimal. Non-rectangular, non-axis-aligned sections will

be implemented and their performance compared and contrasted with the rectangular ones. Although greater computation will be involved in determining the viewers current section, there will be less redundant buildings in the sections, and there may be an overall performance gain. Non-axis-aligned sections better reflect the streets of a city, like Dublin, which does not follow a regular grid system. The PV techniques should also be benchmarked against other culling techniques.

A technique to predict user movement between sections, and download the necessary building models in advance, will be implemented. Section-to-section adjacency data will be generated and used to this end. More advanced image-based rendering (*i.e.* imposter) techniques may be used in the future. At present the imposters are simple billboards; rotating quads with single textures attached. A more advanced graphics API would allow more detailed textures to be used, where necessary. Work has already been done on populating the city with animated crowds, and will continue. Ongoing advances in mobile computing technology, especially the introduction of graphics cards for handheld devices, will enable more ambitious crowd and urban simulations on the mobile platform.

8. Acknowledgements

This research was funded by Enterprise Ireland and the Irish Higher Education Authority.

References

[ARB90] AIREY J. M., ROHLF J. H., BROOKS JR F. P.: Towards image realism with interactive update rates in complex virtual building environments. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics* (New York, NY, USA, 1990), ACM Press, pp. 41–50.

[CG02] CHANG C.-F., GER S.-H.: Enhancing 3d graphics on mobile devices by image-based rendering. In *PCM '02: Proceedings of the Third IEEE Pacific Rim Conference on Multimedia* (London, UK, 2002), Springer-Verlag, pp. 1105–1111.

[Cum03] CUMMINS A.: *Real-time Display of 3D Graphics for Handheld Mobile Devices*. Master's thesis, Trinity College Dublin, 2003.

[DD04] DUGUET F., DRETTAKIS G.: Flexible point-based rendering on mobile devices. *IEEE Comput. Graph. Appl. 24*, 4 (2004), 57–63.

[DHOO05] DOBBYN S., HAMILL J., O'CONOR K., O'SULLIVAN C.: Geopostors: a real-time geometry / impostor crowd rendering system. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2005), ACM Press, pp. 95–102.

[HMDO05] HAMILL J., MCDONNELL R., DOBBYN S., O'SULLIVAN C.: Perceptual evaluation of impostor representations for virtual humans and buildings. *Computer Graphics Forum (Eurographics 2005), To Appear* (2005).

[HO03] HAMILL J., O'SULLIVAN C.: Virtual dublin - a framework for real-time urban simulation. In *Proceedings of WSCG* (2003), vol. 11, pp. 221–225.

[JWSP05] JESCHKE S., WIMMER M., SCHUMANN H., PURGATHOFER W.: Automatic impostor placement for guaranteed frame rates and low memory requirements. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2005), ACM Press, pp. 103–110.

[RCO03] ROSSI C., CUMMINS A., O'SULLIVAN C.: Distributed mobile multi-user urban simulation. In *GRAPH '03: Proceedings of the SIGGRAPH 2003 conference on Sketches & applications* (New York, NY, USA, 2003), ACM Press, pp. 1–1.

[TS91] TELLER S. J., SEQUIN C. H.: Visibility preprocessing for interactive walkthroughs. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1991), ACM Press, pp. 61–70.