# Chapter 01

## Modeling and Visualization of Complex Geometric Environments

- Introduction
- Modeling
- Visualization
- Summary
- References

# Modeling and Visualization of Complex Geometric Environments

***Türker Yılmaz, Uğur Güdükbay, and Varol Akman***
Department of Computer Engineering, Bilkent University, 06800 Bilkent, Ankara, TURKEY

*Visualization of large geometric environments has always been an exciting project for computer graphics practitioners. Modern graphics workstations allow rendering of millions of polygons per second. Although these systems are impressive, they cannot catch up with the quality demanded by graphics systems used for visualizing complex geometric environments. After all, in such systems the amount of data that need to be processed increases dramatically as well. No matter how much graphics hardware evolves, it looks like practitioners are going to crave for what is impracticable for such hardware to render at interactive frame rates. In this chapter, we present some modeling techniques to overcome the problem of graphics hardware bottleneck in a particular context, viz. visualization of terrains and urban environments.*

## 1 Introduction

In this chapter, we first present approaches towards modeling complex geometric environments comprising terrain height fields and urban scenery. Then, we discuss techniques to reduce the amount of workload in the graphics pipeline, thereby overcoming the graphics hardware bottleneck to some extent. These techniques include back-face culling, which eliminates the polygons that are back-facing from the viewer, and view-frustum culling, which eliminates the primitives outside the view-frustum with respect to a view position (so that they are not processed further in the graphics pipeline), as well as view-dependent refinement to selectively refine different parts of a scene using different simplification criteria. We also briefly mention the refinement criteria used in view-dependent visualizations of terrains. We study issues related with urban visualization and especially concentrate on the occlusion culling process by giving a taxonomy of methods in this area. Finally, we discuss techniques to speed-up stereoscopic visualization where the

second eye image is generated from the first eye image (in contrast to generating them separately). As we have already noted, our discussion takes place in the general context of terrain and urban visualization.

# 2 Modeling

The word *modeling* usually refers to the way data are represented in the computer memory and the way in which they are visualized. In the memory, data are kept in suitable data structures that are easy to access for the visualization algorithm. Here, we classify the types of structures into two, namely, *terrains* and *urban scenery*.

## 2.1 Terrain Representations

*Terrains* represent one of the most complex data sets in computer graphics because of their nonstandard nature. There is no simple mathematical characterization of terrains, and hence procedural methods for their representation cannot be applied. The data acquired can be stored and used as height fields, triangulated irregular network models, or quadtrees.

However, during the visualization of the terrains, it is cumbersome (and also unnecessary) to display triangles having all elevation points as their vertices. Therefore, the surface has to be approximated (while introducing some, preferably small, amount of error).

## 2.2 Height Fields

Terrain data are usually obtained by national imagery institutions or geo-science centers. One of the most common ways of acquiring terrain data is aerial photography and satellite imagery. Such data are usually in the form of grids collected at standard intervals, which we call the Digital Elevation Model (DEM). In DEM, the terrain data are not processed and the elevation values are acquired at regular intervals. The collection of elevation samples corresponds to *height fields*.

Height fields are elevation data sampled at regular intervals. The data acquired for terrains are stored in the DEM as height field representations. Naturally, this type of approach requires large amounts of storage because all elevation information are preserved regardless of the characteristics of the terrain surface. For example, the Digital Terrain Elevation Data (DTED) format, developed by NIMA (National Imagery and Mapping Agency of the U.S.), has two standard levels of data resolution (Fig. 1). One is DTED Level-1 in which there are three arc-seconds between two elevation points and the other is DTED Level-2 in which there is one arc-second between two elevation points (meaning higher resolution). Other types of data storage methods for terrain height fields such as gray-scale or vector format are also possible.

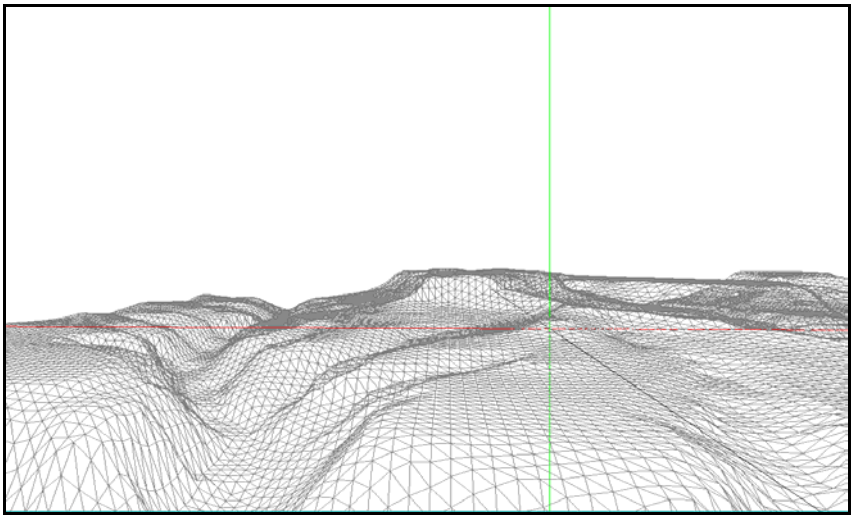

**Fig. 1.** *A sample view of height field information stored in the DEM, such as the DTED.*

### 2.2.1 Triangulated Irregular Networks

For representing the terrain, an efficient alternative to dense grids is the Triangulated Irregular Network (TIN). This stores a surface as a set of non-overlapping contiguous triangular facets of irregular size and shape [18]. The source of digital terrain data is dense raster models produced by automated orthophoto machines or direct sensors such as synthetic aperture radar.

A terrain surface can be characterized by a set of surface-specific points (peaks, pits, and passes), and a set of lines (ridges and channels) which connect them. The sample points in the TIN are chosen so that these features are contained as subgraphs of the model [18].

The surface is modeled as a set of contiguous non-overlapping triangles whose vertices are located adaptively on the terrain. The height field data are simplified using simplification algorithms and the resulting model is triangulated using the Delaunay criterion, finally yielding a TIN.

The TIN model is especially attractive because of its simplicity and memory efficiency. It is a significant alternative to the regular raster of the GRID model (Fig. 2). TINs can describe a surface at different resolutions.

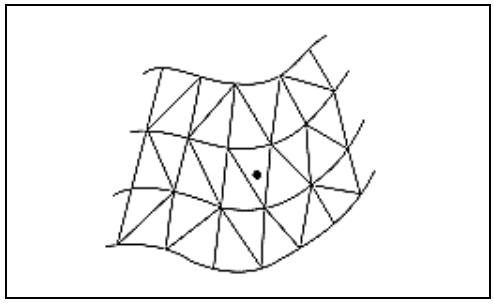

**Fig. 2.** *TIN generation.*

There are three ways of storing a triangulated network [34]:

- triangle-based structure (Table 1),
- point-based structure (Table 2), and
- edge-based structure (Table 3).

The first method is better for storing attributes (e.g., slope) for each triangle, but uses more storage space. The second one is better for generating contours and uses less storage, but attributes such as slope must be calculated and stored separately. The third one is an additional structure that must be maintained. Based on edge definitions, it provides neighboring information. (Here, it is necessary to store the previous two structures.) If an application needs this information, this structure is suitable.

**Table 1.** *Storage with triangle-based structure.*

| ID | Triangle Vertex Coordinates | Neighbors |
|---|---|---|
| 0 | $(x_{01}, y_{01}, z_{01})$, $(x_{02}, y_{02}, z_{02})$, $(x_{03}, y_{03}, z_{03})$ | 8, 9, 1 |
| 1 | $(x_{11}, y_{11}, z_{11})$, $(x_{12}, y_{12}, z_{12})$, $(x_{13}, y_{13}, z_{13})$ | 0, 12, 15 |
| ⋮ | ⋮ | ⋮ |

**Table 2.** *Storage with point-based structure.*

| ID | Vertex Coordinate List of All Terrain | Triangle-Based Structure IDs |
|---|---|---|
| 0 | $(x_0, y_0, z_0)$ | 8, 9, 1, 7, 12 |
| 1 | $(x_1, y_1, z_1)$ | 3, 4, 6, 13 |
| ⋮ | ⋮ | ⋮ |

**Table 3.** *Storage with edge-based structure.*

| ID | Point-Based Structure IDs | Triangle-Based Structure IDs |
|---|---|---|
| 0 | $(PB_{ID01}, PB_{ID02})$ | $Left_{TBID01}$, $Right_{TBID02}$ |
| 1 | $(PB_{ID11}, PB_{ID12})$ | $Left_{TBID11}$, $Right_{TBID12}$ |
| ⋮ | ⋮ | ⋮ |

Contour lines are one of the terrain features, representing the relief of the terrain with the same height. TINs can also be generated using elevation points along these contour lines and the interpolation is straightforward. However, TINs in island-like situations, as shown in Fig. 3, are special cases that warrant attention.

In the following section, we will describe another method for creating terrain surfaces. This method is also adaptive but at the expense of more storage. In return, it supports faster queries on the terrain structure.

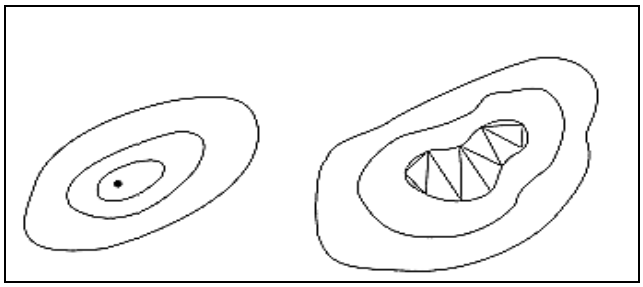

**Fig. 3.** *TINs in an island-like situations should be handled with care.*

### 2.2.2 Quadtree Representation

A quadtree is a rooted tree with internal nodes having four children. Each node is represented by at least four grid elevations, which correspond to squares. Constructing a quadtree for terrain data stored in a DTED file with grid elevations produces a dense representation of the terrain, i.e., all interval elevations are stored. The root node represents the whole terrain data with four corners. As we go down to the deeper levels in the quadtree hierarchy, the distance between the corners is halved and at the deepest level, there remains no other elevation point that is to be represented by the nodes of the quadtree (Fig. 4). The data structure obtained after the quadtree scheme applied is passed to a simplification algorithm. The model is simplified to eliminate the nodes having the same elevations within all children (Fig. 5). Further details of quadtrees and related hierarchical structures can be found in [39].

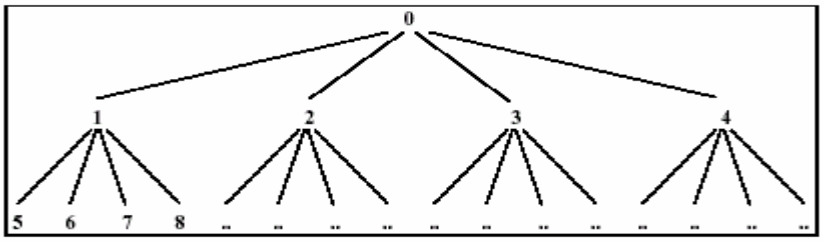


**Fig. 4.** *Quadtree structure.*

The quadtree structure can be represented as a 1D array (Fig. 4). In this tree, each level represents a different level-of-detail on the terrain. In order to traverse the nodes of the quadtree, represented as a 1D array, the following algorithms can be used:

```
parent(int child)
  return floor((child-1)/4);

child(int parent)
  if (level(parent)==MAXLEVEL)
       childnode=sibling(parent);
  else
       childnode=(4*parent)+1;
  return childnode;

sibling(int node)
  if (level(node)<level(node+1))
       return NIL;
  else
       if (parent(node)!=parent(node+1))
            return sibling(parent(node));
       else
            return (node+1);
```

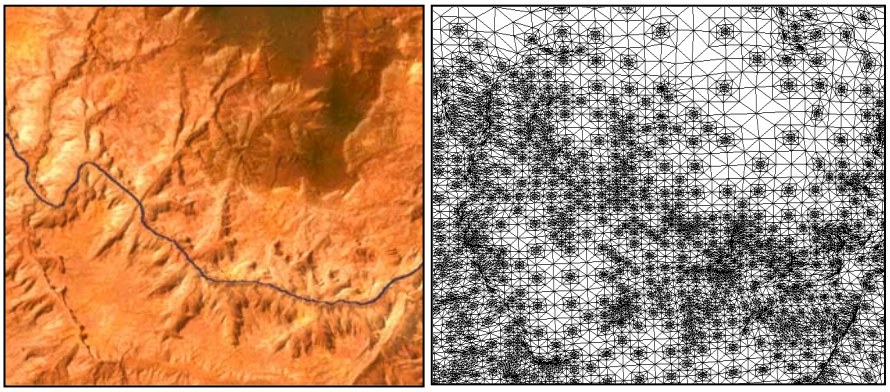

**Fig. 5.** *A simplifed quadtree representing a particular terrain (Grand Canyon data obtained from the United States Geological Survey (USGS). Processing by Chad McCabe of the Microsoft Geography Product Unit.).*

Quadtrees can be used to store indices, minimum and maximum elevations, and activation distances for the vertices, which are valuable for view-dependent refinement in terrain visualization. It is clear that an array-based representation can be used to eliminate the need for pointer manipulation. The numbering scheme used by the quadtree structure when it is stored in a one-dimensional array is

illustrated in Fig. 6. The root is labeled as 0 and the other nodes are numbered recursively in the counterclockwise direction.

<table>
<tr><td rowspan="8">4</td><td rowspan="8">3</td><td rowspan="4">20</td><td rowspan="4">19</td><td rowspan="4">16</td><td rowspan="4">15</td><td>84</td><td>83</td><td>80</td><td>79</td><td>68</td><td>67</td><td>64</td><td>63</td></tr>
<tr><td>81</td><td>82</td><td>77</td><td>78</td><td>65</td><td>66</td><td>61</td><td>62</td></tr>
<tr><td rowspan="4">17</td><td rowspan="4">18</td><td rowspan="4">13</td><td rowspan="4">14</td><td>72</td><td>71</td><td>76</td><td>75</td><td>56</td><td>55</td><td>60</td><td>59</td></tr>
<tr><td>69</td><td>70</td><td>73</td><td>74</td><td>53</td><td>54</td><td>57</td><td>58</td></tr>
<tr><td rowspan="8">1</td><td rowspan="8">2</td><td rowspan="4">8</td><td rowspan="4">7</td><td rowspan="4">12</td><td rowspan="4">11</td><td>36</td><td>35</td><td>32</td><td>31</td><td>52</td><td>51</td><td>48</td><td>47</td></tr>
<tr><td>33</td><td>34</td><td>29</td><td>30</td><td>49</td><td>50</td><td>45</td><td>46</td></tr>
<tr><td rowspan="4">5</td><td rowspan="4">6</td><td rowspan="4">9</td><td rowspan="4">10</td><td>24</td><td>23</td><td>28</td><td>27</td><td>40</td><td>39</td><td>44</td><td>43</td></tr>
<tr><td>21</td><td>22</td><td>25</td><td>26</td><td>37</td><td>38</td><td>41</td><td>42</td></tr>
</table>

**Fig. 6.** *Numbering scheme for quad blocks in a quadtree when it is stored in a 1D array with levels 2, 3, and 4.*

### 2.2.3 Multi-resolution Representation using Quadtrees

Multi-resolution representation of data refers to those data structures, which provide a way to visualize data in different resolutions, depending on some criterion. In this section, we describe multi-resolution representation using quadtrees. For other methods of multi-resolution representation such as progressive meshes, the reader is referred to [26].

In order to visualize complex scenes (e.g., terrain height fields) at interactive frame rates, efficient data structures need to be used. The quadtree representation perfectly fits into grid elevation data. Generally, triangles are used as modeling primitives for complex scenes. The triangulation must be adaptive in order to reduce the number of polygons that should be processed and make efficient use of the limited memory sources. This means that high frequency elevation changes should be triangulated with more triangles than low frequency regions. While doing this, artifacts that can emerge on the terrain should be minimized as much as possible.

For multi-resolution representations, under the assumption that the structure in Fig. 4 is used, each level represents a different resolution on the terrain. The algorithms developed for multi-resolution representation generally make use of *view-dependent*

*visualization*. This means the usage of some simplification criterion and traversing the nodes of the multi-resolution terrain representation hierarchy (Fig. 4) until the criterion is met.

## 2.3 Modeling of Urban Scenery

### 2.3.1 Data Acquisition and Modeling

As techniques for the acquisition of 3D data are developed, the need to improve current data representation and visualization methods becomes more pressing. These acquisition techniques and sources may involve:

- aerial (satellite) imagery of earth in high resolution,
- urban data automatically gathered by devices using laser range finding methods, and
- 3D views constructed with the help of radar systems.

Actually, data acquisition for terrains is less expensive than data acquisition for urban areas. For urban scenery creation, there are mainly three sources of information.

*1. Constructing models from building footprints:*

One of the main sources of information is building footprints, mostly available at official institutions of cartography. The buildings may easily be extruded from these footprints using the additional information stored, i.e., the number of floors or the construction type. However, the resultant geometry is a crude version, omitting many details such as balconies, pillars, etc. To make visualization more realistic, texturing may be applied. Moreover, there are techniques to automatically add selected features to the buildings, like balconies or windows [47]; see Fig. 7 for an illustration.

2. *Constructing building models individually, and later populating them in a virtual environment:*

In this technique, each building model is highly detailed and the resulting appearance is highly realistic. A sample application of this

method is shown in Fig 8, where we generated a virtual city composed of nearly 300 buildings with more than 400K polygons.

**Fig. 7.** *A scene from [47], where buildings are extruded and additional features are added to the building geometry (© Peter Wonka. Reprinted with permission.).*

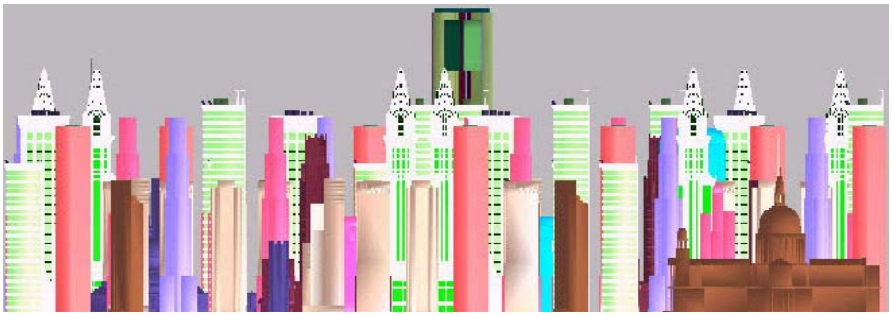

**Fig. 8.** *A virtual city that is a combination of individually modeled buildings.*

3. *(Semi)automatic reconstruction of buildings from aerial photographs:*

There are many ways of collecting urban information from aerial photographs. These images are procedurally corrected and methods are applied to obtain 3D data of an urban area. These data are later geo-corrected and other information is appended to them. To give an example [36], from various image maps given as input (e.g., land-

water boundaries and population density), a system generates a network of highways and streets, divides the land into lots, and creates the appropriate geometry for buildings on the respective allotments (Fig. 9).

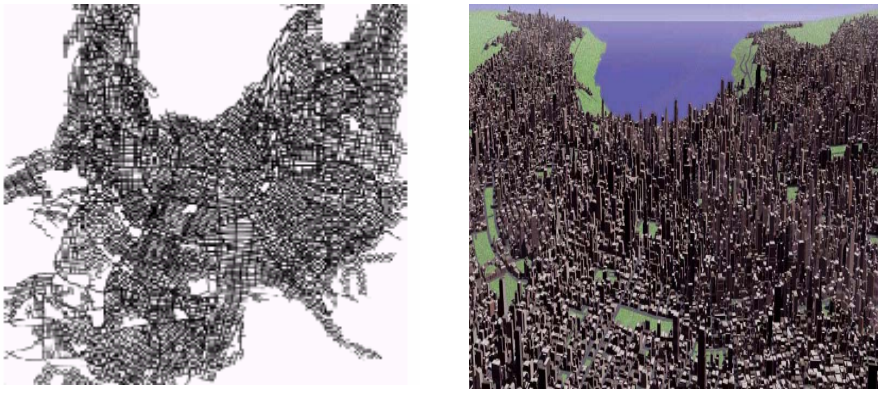

**Fig. 9.** *A (procedurally modeled) virtual city from [36]. (© Association for Computing Machinery. Reprinted with permission.)*

### 2.3.2 Building Representations

Naturally, buildings and terrain data are stored in spatial databases. The Database Management Systems (DBMSs) for spatial databases should manage access and retrieval of data to be sent to the graphics pipeline. There are mainly three types of building representations: *primitive-geometry*, *component-based*, and *space-based* [42].

*Primitive-geometry representation*: Most architectural drawings can be regarded as primitive-geometry representations. This kind of representation is based on geometric primitives, which give no explicit indication of what building entities they stand for. For this representation, storage space that can be used for the spatial database is crucial; the representation detail and the features that could be incorporated into the database require excessive amount of storage.

*Component-based representation*: We can cite CAD systems in this category. Most commercial design systems support explicit definitions for 3D building entities such as walls, windows, doors, floor slabs, and roofs. Representing buildings using this method allows designers to create and modify a single model rather than several (computationally unrelated) floor plans. In the representation

scheme, the spatial locations of the components are defined. More importantly, entities can be accessed based on their respective locations in the scene. In component-based systems, the buildings may be assigned additional information, such as floor count, occupied area, type of construction, energy type used for heating, purpose of use, etc. The components of the building are stored and relations among them are defined. When there is a need to retrieve a component, the relationships among the currently displayed components can be used.

*Space-based representation*: This is used to define the spaces used by the components of buildings. Instead of obtaining spaces used by a component by the spatial locations of other components, the space is enclosed by another polygon and this polygon is used for space determination. The designer can achieve this by defining the polygons that enclose a space. In the former case, when the polygon is closed, floor and ceiling elements can be automatically generated. Thus, a space is always ensured to be a polyhedron.

## 3 Visualization

Visualization is defined as the transformation of the symbolic data into a geometric form to enable researchers to observe their simulations and computations. It can be used both for interpreting image data fed into a computer (=image understanding) and for generating images from complex multidimensional data sets (=image synthesis) [46]. Here, we use the word “visualization” to mean the generation of images of complex 3D scenes for different camera positions while moving the camera interactively (i.e., rendering complex scenes). If interaction is required, then the frame rate of the drawn scene should be more than 17 frames per second. If the frame rates are below this, then it means the system has a bottleneck either in the graphics pipeline, or in the display process. If the graphics load is not adjusted, then the user might experience problems during visualization such as jaggy motion. Taking the graphics hardware as a constant, the only approach that can overcome this bottleneck and achieve interactive frame rates is to adjust the load of the graphics pipeline by using suitable algorithms (i.e., decreasing the number of triangles processed for each frame

using a predefined simplification criterion). These algorithms work according to the viewer position.

### 3.1 Culling Techniques

Before applying a view-dependent refinement process to reduce the number of triangles to be rendered, the portions of the scene that will not be displayed for a frame should be culled. In order to send only the related portions of the scene to the display processor, there are mainly three types of culling methods to get rid of the irrelevant portions of the geometry. The first one is *back-face culling*, discarding those polygons whose surface normals are facing away from the viewer. (This works only for convex objects.) The second one is *view-frustum culling*, discarding those objects that are out of the field of view. The last one is *occlusion culling*; this eliminates parts that are occluded by front objects and it is especially important for visualization of urban scenery where the buildings are occluding each other for different views. Back-face culling is explained below. View-frustum culling for terrain data and occlusion culling for urban scenery are discussed later in related subsections. In the case of *occlusion culling* for terrains, it is known that this culling method does not increase the performance significantly [28].

#### 3.1.1 Back-Face Culling

Back-face culling is the process of discarding back-facing polygons. It is not possible to see them because they are not facing the viewer. However, back-face culling works only if:

- there are no holes or transparent passages in the objects, and
- the objects are convex.

Back-face culling is performed by evaluating the equations of the planes that form the object surface, i.e., triangles with respect to the viewpoint and viewing direction. Back-facing polygons are eliminated if the dot product of the viewing direction and polygon normal is greater than zero. Back-face culling is implemented in hardware in many graphics boards. In [30], some improvements are proposed and are compared to the hardware implementation. A sub-linear algorithm for back-face computation is presented. The

polygonal model is partitioned into a hierarchy of clusters based on the similarity of orientation and physical proximity of polygons. The space is partitioned into back, front, and mixed regions with respect to each cluster. At run time, the algorithm uses the pre-computed cluster descriptions to locate the viewpoint in the corresponding region of each cluster.

## 3.2 Visualization of Terrain Data

Without decreasing the amount of geometry sent to the graphics pipeline, the quality and amount of the graphics primitives that can be viewed at interactive frame rates will be limited and insufficient. Therefore, the surface of a terrain has to be approximated up to a certain threshold, in order to decrease the number of triangles sent to the graphics pipeline without significant loss of image quality. While carrying out this process, the simplification part should cost significantly less than sending all graphics to the hardware and making the graphics pipeline do all the work.

### 3.2.1 View-Frustum Culling for Terrain Data

The view-frustum is a pyramid (generally chopped off using front and rear planes perpendicular to the viewing direction) and is based on the viewing parameters of the application. View-frustum culling process culls the parts of the view-frustum according to the six planes of the frustum.

A sample view-frustum over a terrain can be seen in Fig. 10. View-frustum culling for terrain data can be done as follows. The quadtree is traced from top to bottom and it is determined whether the nodes are viewable from the current viewing direction. Nodes of the quadtree are visited in preorder so that if a higher-level node is not in the frustum, then the children of that node are not further checked.

An efficient view-frustum culling (VFC) process is crucial to achieve interactive frame rates. The data structure is checked against the viewing frustum. To speed-up frustum culling process, frustum tests can be done using bounding spheres enclosing the nodes. In view-frustum culling, several optimizations are possible [24]:

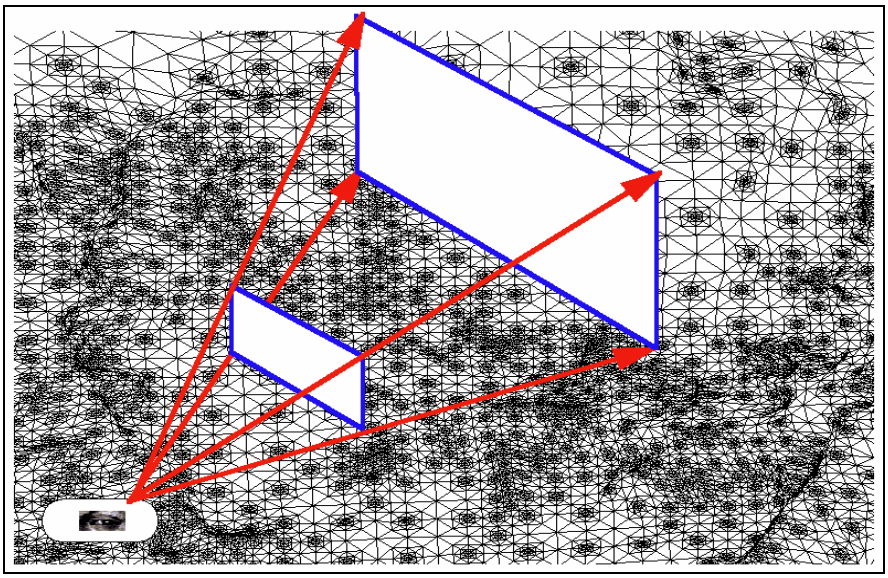

**Fig. 10.** *A sample view-frustum on terrain data.*

- One important optimization is to utilize the coherence between two frames when the user navigates through the terrain. If the user moves forward, then there is no need to cull the whole data again since the data have already been culled in the previous frame. So, previously culled blocks can be used for the current frame. This method is applicable if the frustum is not culled according to the far plane [2].

- Another method is *deferred VFC*, implying that VFC is not done for every frame but at predefined intervals. In this way, the overhead brought by the VFC step can be decreased. One problem with this approach is the navigation speed. If the user moves very fast involving rotation and backward motion, then the screen may not refresh on time. Accordingly, this approach is suitable for slow motion walkthroughs.

- As another approach, VFC depending on the deviation of the viewer location may be used. Deviation based culling is suitable for walkthroughs in which the viewer navigates very

fast. The VFC algorithm can be run only if the user moves a prespecified distance from the previously culled position.

If the data are large, then we have to test for the far plane too. In this case, an altitude-based scheme can be used for far plane distance determination. If the altitude of the viewer is at lower levels in the data, the far plane is brought closer to the viewer, proportional to the altitude of the viewer because it may not be possible to see farther distances. This approach establishes a balance between the frustum distance and the data resolution, depending on the type of the data. For view-frustum culling in urban sceneries, the quadtree node explained above could be replaced with objects in the urban scene, where the application of it is straightforward.

### 3.2.2 View-Dependent Visualization using Quadtrees

During the visualization process, there is no reason to send the parts of the scene that cannot be seen from the viewer's position and viewing direction. The fundamental idea in view-dependent rendering is to perform culling of these unnecessary data and reduce the workload of graphics hardware. However, whenever view-dependent rendering is mentioned, usually only multi-resolution representations come to mind. A multi-resolution representation deals with simplifying parts of the scene, when the detailed view is not needed. View-dependent rendering covers all visibility algorithms, which try to speed up the rendering performance, because all culling algorithms are based on the viewer's position and viewing direction.

View-dependent rendering is mostly performed on-the-fly, in order to reduce the cost of secondary storage and provide a more realistic view of the scene. An alternative to dynamic view dependent visualization, where the scene is simplified on-the-fly based on the current view, using precomputed Level-of-Detail (LOD) samples during a fly-through is also commonplace. Simplification algorithms are applied to obtain a hierarchy of successively coarser approximations for the objects. Such multi-resolution hierarchies have been used in LOD based rendering schemes to achieve better frame rates per second [20, 33]. These hierarchies usually have a number of distinct levels of detail, usually five to 10 for a given object or a part of terrain [49]. During the

visualization, if certain error criteria are satisfied, then one of the static detail levels is chosen and the object is displayed at that level.

The trend in surface simplification moved from statically defined levels of detail to dynamically created levels of detail after mid 1990's. Wavelet usage [16] and progressive meshes [26, 27, 28] have advanced simplification work a step further. These methods produce a continuous LOD through the entire scene, instead of a discrete number of levels of detail. Progressive meshes offer an elegant solution for continuous representation of polygonal meshes, which can be adapted to almost any kind of scene objects. In [49], the authors describe improvements on progressive meshes, by defining merge trees for performing edge collapses that permit adaptive refinement around any vertex. Progressive mesh usage is also adapted to regular grid approaches [32].

With quadtree representation, the grid structure of the elevation data is explored. The grid structure naturally lends itself to the quadtree representation. Each sequence of nine vertices comprehends to a quad block as in Fig. 11. These elevation points are selected in such a manner that no cracks occur during the simplification process and the elevation differences without the vertices do not disturb the quality of the resultant scene much – in any case, not more than a prespecified threshold value. The threshold values are defined in view of the simplification criterion employed by the simplification algorithm.

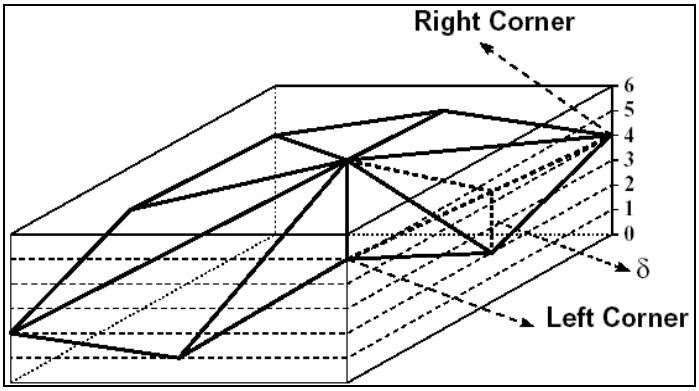


**Fig. 11.** *The distance δ between original and removed positions of the tested vertex.*

There are several methods and criteria to simplify the data. The criterion for the evaluation of removal depends on several factors. One of these methods is *screen-space error criterion*. In this method, elevation differences are taken into account to evaluate a vertex for removal (Fig. 12). The number of projected pixels for the vertex is calculated for this purpose (Fig. 11). If the projected size of an object is below a prespecified threshold (in terms of pixels), then the vertices and associated triangles within this object are removed. If this size is greater than the pre-specified pixel tolerance, then the vertex is kept.

The problem here is that if the viewer is looking at the terrain from above, then the number of projected pixels to the camera plane will be very small, causing the vertex to be considered as unimportant and yielding to the elimination of it. This especially becomes a problem for stereoscopic visualizations where the preservation of the depth information is crucial. The problem can be illustrated via an example (Fig. 12). Suppose we are looking at a tower from above and we use the screen-space error tolerance. Since the projection of the elevation difference will be very small with respect to the position of the eye, the tested vertices will be removed, although they are important to the viewer (i.e., they will make the viewer see the height of the tower when viewed in stereo). Thus, while the screen-space error metric is suitable for the monoscopic view [32], it degrades the stereo effect and may result in incorrect stereoscopic vision.

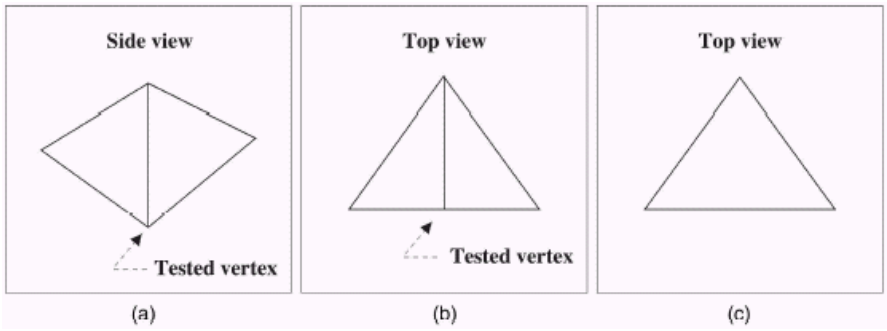


**Fig. 12.** *Screen-space error metric. (a) Side view of a quad block. (b) Top view of the same block. (c) Edge removal by screen-space error-based algorithm when the block is viewed from the top.*

Instead of using screen space error criterion, another one, called *distance-based angular error metric*, is defined [24]. This simplification criterion is suitable for both monoscopic and stereoscopic visualization, and yields faster algorithms for simplification. The elevation and distance of objects from the viewer are two important criteria that make us appreciate depth and differentiate between objects. Therefore, the threshold value must be specified adaptively so that it takes into account both of these parameters to reflect the correct depth information. Details of distance-based angular error threshold can be found in [24].

In order to prevent cracks over the terrain during simplification and provide a suitable heuristic for morphing to eliminate popping artifacts while switching between different resolutions of the terrain, a valid triangulation should be maintained. The vertex activation scheme using distance-based angular error threshold decides to activate or deactivate a vertex using the precomputed vertex activation values, and the distances between the viewer and the vertex location. Then, the area is triangulated accordingly. The vertex activation scheme is based on assigning correct activation distances to each vertex from the lowest level up to highest using a maximization operation and during visualization only measuring the distance between the viewer and the vertex. The details of vertex activation and triangulation can be found in [24].

### 3.3 Visualization of Urban Scenery

Urban visualization requires culling of unnecessary data, in order to navigate through the scene at interactive frame rates. Creating tight approximations of the data that need to be processed by the graphics hardware requires efficient occlusion culling algorithms. Besides, most of the current algorithms are either for architectural walkthroughs, in which there are large occluders hiding the most of the geometry behind them, or conservative methods for outdoor visualization.

For interactive walkthroughs of large building models or city-like scenes, a system must store in memory and render only a small portion of the model at each frame. The most important challenge is to identify the relevant portions of the model, swap them into

memory by using a robust spatial database access, and render at interactive frame rates as the user changes position and viewing direction.

### 3.3.1 Occlusion Culling with Preprocessing

Occlusion culling algorithms can be classified based on targeted environments. They are generally effective in densely occluded scenes and do not offer much in terrain like scenes [28]. On the other hand, LOD control and impostors contribute mostly in wide-open sparsely occluded situations. The first classification is based on the suitability of the data for occlusion culling. According to this classification, in the first category there are algorithms that are suitable for scenes where much of the geometry is hidden behind potential occluders. In the second category, there are algorithms developed for general scenes. These are so complex that use of some special hardware might be more suitable. Most current occlusion culling algorithms work with occluders much smaller than they could actually handle and often need human intervention for finding effective occluders (or send hidden geometry to graphics pipeline unnecessarily). Therefore, tightness of the estimation of occluded parts is an important issue to be considered.

Most of the visibility-culling algorithms have computationally intensive preprocessing stages [15, 20, 21, 25, 31]. Preprocessing generally includes the computation of some kind of hierarchical data structure to store the scene and finding visible objects and majority of occluders for previously determined view cells. The clustering schemes are carefully chosen so that the algorithms make use of the data structures created during this step. For walkthroughs of outdoor environments, controlled subdivision using data structures like BSP trees or quadtrees is suitable [5, 19, 25, 35]. A 3D spatial partitioning data structure such as a quadtree, octree or kD-tree can be used for general fly-through application [12]. The visibility octree is an adaptive data structure that stores potentially visible sets at its terminal nodes [40, 50]. Unlike uniform grid structures, its size depends on the nature of the scene. Different schemes for occlusion culling are applied after this clustering step has been performed.

The goal of an efficient visibility-culling algorithm is to calculate a conservative and fast elimination of those parts of the

scene that are definitely invisible. This means that if an object is visible it is certainly displayed. In the meantime, some unnecessary parts are sent to graphics pipeline. *Object space algorithms* are the ones that geometrically make computations on the scene and decide whether the objects are visible or not [8, 9, 10, 25, 29, 43]. The general approach of the previous work is to select some polygons to act as virtual occluders and check if they occlude any objects seen from the viewer. To reduce the cost of checking, occludees are usually approximated by bounding volumes. There is a conservative visibility preprocessing method for outdoor environments, which is described in [8]. A conservative superset of visible objects is computed for each cell by searching for a strong occluder for each object such that it cannot be seen from any point in the cell. The cellulization approach applied in this work is regular grid approach, which may not fit to real outdoor environments. The cellulization approach is best suited for architectural walkthroughs. In case of outdoor visualizations, a suitable cellulization is needed, which is not straightforward. Still, after suitable cellulization of the navigable area, most of the approaches used to visualize the indoor environments can also be used for outdoor environments.

Mostly, the target data for occlusion culling algorithms influence the way algorithms are designed. If an urban walkthrough restricts the viewer to move along the road paths, then cells should be associated with the roads only. For building interiors or ship-like scenes, most visibility algorithms decompose the model into cells [8, 20, 21, 43]. An occlusion region can be specified by an object space occlusion-culling algorithm using supporting planes [10]. These cells are connected by portals and the inter-cell visibility is computed. Since the walls of the buildings or doors of ships occlude a large amount of the geometry behind them, precomputing the potentially visible sets (PVSs) and later using this information to cull the invisible objects may be a promising approach [17, 20, 21, 43, 44]. This has the disadvantage of requiring large secondary storage for the PVS information. There are some algorithms to compress the data constituting the PVSs [3, 37, 38].

Conservative algorithms classify regions as invisible when they are completely occluded. Partially occluded objects are sent to the graphics pipeline as a whole. For urban environments that have less hidden geometry behind the objects, occlusion culling with a few

large occluders is a popular approach. The navigable area is again subdivided into cells in many approaches and for each frame, a small set of (about five to 30) occluders that are likely to occlude a large part of the model is selected. The reason why these algorithms select only a small set of occluders is that the amount of data needed to store the potentially visible set for each cell is large. The selection schemes differ among the algorithms with respect to errors introduced into the resultant image, accurateness of the selection, tightness of the conservativeness, and the data that are needed to be stored with this potentially visible set [1, 14, 29].

Under the aforementioned classification, object space methods can be regarded as output sensitive algorithms. Output sensitive algorithms have their runtime depend only on the size of the output, and not the input. In the case of *image based algorithms*, the main idea for achieving the goal is to perform visibility computations for each frame by scan conversion of some potential occluders and checking if the projections of the other objects fall inside the image area of the projection of the occluders. Examples include the hierarchical z-buffer algorithm [22], hierarchical tiling algorithm [23], hierarchical occlusion maps [51], and others [4, 6, 11, 15, 45, 48]. Some of these classify the scene into both scene data structure and image replaceable parts, namely near and far fields. This sort of occlusion culling is very similar to radiosity calculations [7]. In image-based simplification methods, the whole scene parts are replaced with an *impostor* – a generated image of the scene [13, 41]. Unfortunately, one impostor is usually valid for a few frames and must be updated frequently. Other approaches use textured depth meshes that incorporate depth information for efficient impostor update. One of the advantages of image space algorithms is that the target data can be very complex (for which object space algorithms are at a disadvantage) and the occluded objects are within a very tight estimation range. A common deficiency of image space algorithms is that they are mostly hardware dependent and the screen resolution is fixed.

### 3.4 Stereoscopic Visualization

In stereoscopic visualization (Fig. 13), the two views must be generated fast enough to achieve interactive frame rates. Apparently,

there will be limitations in terms of the features that could be incorporated to increase the realism (in comparison to monoscopic visualizations). Since the amount of data that can be processed decreases drastically, complex visualizations, such as visualization of urban scenery over a terrain, cannot be achieved easily.

There is some work done to decrease the time needed for generating the second eye image so that complex stereoscopic visualizations become possible. For this purpose, an algorithm has been proposed to speed up the generation of stereo pairs for stereoscopic view-dependent visualizations [24]. The algorithm, called Simultaneous Generation of Triangles (SGT), generates the triangles for the left and right eye images simultaneously using a single draw-list, thereby avoiding the need for performing the view frustum culling and the vertex activation operations needed for view-dependent refinement twice.

## 4 Summary

In this chapter, we presented some approaches toward terrain and urban scenery modeling. We discussed techniques to reduce the amount of workload in the graphics pipeline, thereby overcoming the graphics hardware bottleneck.

In the case of terrains, the most important problem is multi-resolution representation and simplification without significant loss of accuracy. In order to achieve this, an approach using quadtrees has been presented. Urban sceneries are somewhat different from terrains; they are more component based. An important problem is occlusion culling, because most geometry may be discarded without sending it to the graphics pipeline. In order to achieve these goals the algorithms must be adaptive: the algorithm should cost less compared to sending whole geometry to the graphics hardware and letting the machine do all the work and should keep accuracy of the geometry high as much as possible.

There may be other approaches to terrain and urban modeling, not mentioned in this chapter. However, we believe that the ones studied here constitute a useful, promising bunch.

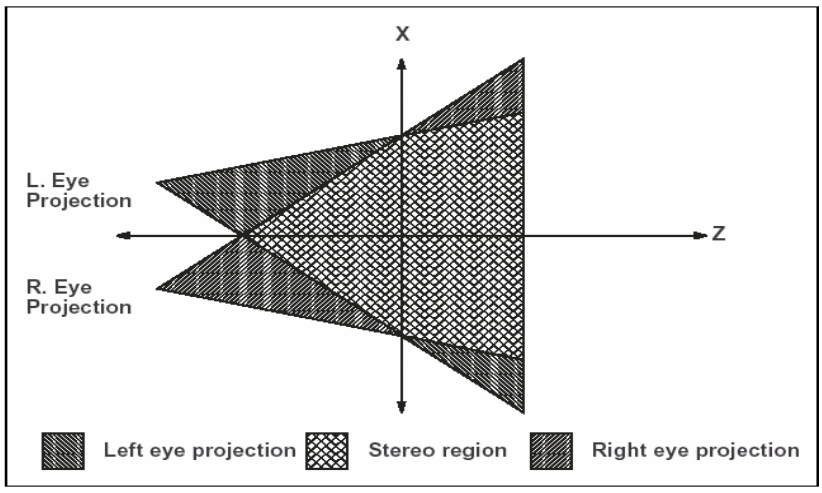


**Fig. 13.** *Stereoscopic projection with left and right eye frustums.*

## References

1. C. Andùjar, C. Saona-Vàzquez, I. Navazo, and P. Brunet. Integrating Occlusion Culling and Levels of Detail Through Hardly-Visible Sets. Computer Graphics Forum, 19(3):499–506, 2000.
2. U. Assarsson and T. Möller, Optimized View Frustum Culling Algorithms for Bounding Boxes, Journal of Graphics Tools, 5(1): 9-22, 2000.
3. C. L. Bajaj, V. Pascucci, and G. Zhuang. Progressive Compression and Transmission of Arbitrary Triangular Meshes. In Proceedings of IEEE Visualization, pages 307–316, 1999.
4. D. Bartz, M. Meißner, and T. Hüttner. OpenGL-Assisted Occlusion Culling for Large Polygonal Models. Computers & Graphics, 23(5):667–679, 1999.
5. J. Bittner, V. Havran, and P. Slavik. Hierarchical Visibility Culling with Occlusion Trees. In Proceedings of Computer Graphics International, pages 207–219, 1998.
6. B. Chen, J. E. Swan, E. Kuo, and A. E. Kaufman. LOD-Sprite Technique for Accelerated Terrain Rendering. In Proceedings of IEEE Visualization, pages 291–298, 1999.
7. Y.-Y. Chuang and M. Ouhyoung. Clustering and Visibility Preprocessing of Hierarchical Radiosity for Object-Based Environment. In Proceedings of Computer Graphics Workshop'95, pages 102–111, 1995.
8. D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative Visibility and Strong Occlusion for Viewspace Partitioning of Densely Occluded Scenes. Computer Graphics Forum, 17(3):243–254, 1998.

9. S. Coorg and S. Teller. Temporally Coherent Conservative Visibility. In Proceedings of ACM Symposium on Computational Geometry, pages 78–87, 1996.
10. S. Coorg and S. Teller. Real-Time Occlusion Culling for Models with Large Occluders. In Symposium on Interactive 3D Graphics, pages 83–90, 1997.
11. L. Darsa, B. Costa, and A. Varshney. Walkthroughs of Complex Environments Using Image-Based Simplification. Computers & Graphics, 22(1):55–69, 1998.
12. D. Davis, W. Ribarsky, T. Y. Jiang, N. Faust, and S. Ho. Real-Time Visualization of Scalably Large Collections of Heterogeneous Objects. In Proceedings of IEEE Visualization, pages 437–440, 1999.
13. X. Decoret, F. Sillion, G. Schaufler, and J. Dorsey. Multi-Layered Impostors for Accelerated Rendering. Computer Graphics Forum, 18(3):61–73, 1999.
14. L. Downs, T. Möller, and C. H. Sèquin. Occlusion Horizons for Driving Through Urban Scenes. In SIGGRAPH'01 Proceedings, pages 121–124, 2001.
15. F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative Visibility Preprocessing Using Extended Projections. In SIGGRAPH'00 Proceedings, pages 239–248, 2000.
16. M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution Analysis of Arbitrary Meshes. In SIGGRAPH'95 Proceedings, pages 173–182, 1995.
17. C. Erikson, D. Manocha, and W. V. Baxter. HLODs For Faster Display of Large Static and Dynamic Environments. In SIGGRAPH'01 Proceedings, pages 111–120, 2001.
18. R. J. Fowler and J. J. Little. Automatic Extraction of Irregular Network Digital Terrain Models. Computer Graphics, 1979.
19. H. Fuchs. On Visible Surface Generation by a Priori Tree Structures. In SIGGRAPH'80 Proceedings, pages 124–133, 1980.
20. T. A. Funkhouser, C. H. Sèquin, and S. J. Teller. Management of Large Amounts of Data in Interactive Building Walkthroughs. In Proceedings of Symposium on Interactive 3D Graphics, pages 11–20, 1992.
21. C. Gotsman, O. Sudarsky, and J. A. Fayman. Optimized Occlusion Culling Using Five-Dimensional Subdivision. Computers & Graphics, 23(5):645–654, 1999.
22. N. Greene. Hierarchical Z-buffer Visibility. In SIGGRAPH'93 Proceedings, pages 231–238, 1993.
23. N. Greene. Efficient Occlusion Culling for Z-Buffer Systems. In SIGGRAPH'99 Proceedings, pages 78–79, 1999.
24. U. Güdükbay and T. Yılmaz. Stereoscopic View-dependent Visualization of Terrain Height Fields. IEEE Transactions on Visualization and Computer Graphics, 8(4):330–345, 2002.
25. J. Heo, J. Kim, and K. Wohn. Conservative Visibility Preprocessing for Walkthroughs of Complex Urban Scenes. In Proceedings of ACM Symposium on Virtual Reality Software and Technology, pages 115–128, 2000.
26. H. Hoppe. Progressive Meshes. In SIGGRAPH'96 Proceedings, pages 99–108, 1996.

27. H. Hoppe. Efficient Implementation of Progressive Meshes. Computers & Graphics, 22(1):27–36, 1998.
28. H. Hoppe. Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering. In Proceedings of IEEE Visualization, pages 35–42, 1998.
29. J. T. Klosowski and C. T. Silva. Efficient Conservative Visibility Culling Using the Prioritized-Layered Projection Algorithm. IEEE Transactions on Visualization and Computer Graphics, 7(4):365–379, 2001.
30. S. Kumar, D. Manocha, W. Garrett, and M. Lin. Hierarchical Back-Face Computation. Computers & Graphics, 23(5):681–692, 1999.
31. F. A. Law and T. S. Tan. Preprocessing Occlusion for Real-Time Selective Refinement. In Proceedings of Symposium on Interactive 3D Graphics, pages 47–54, 1999.
32. P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. Turner. Real-Time Continuous Level of Detail Rendering of Height Fields. In SIGGRAPH'96 Proceedings, pages 109-118, 1996.
33. P. W. C. Maciel and P. Shirley. Visual Navigation of Large Environments Using Textured Clusters. In Proceedings of Symposium on Interactive 3D Graphics, pages 95–102, 1995.
34. S. Murai. GIS Workbook CD-ROM Version 1.0, Japan Association of Remote Sensing. 1999.
35. B. Naylor. Partitioning Tree Image Representation and Generation from 3D Geometric Models. In Proceedings of Graphics Interface, pages 201–212, 1992.
36. Y. I. H. Parish and P. Müller. Procedural Modeling of Cities. In SIGGRAPH'01 Proceedings, pages 301–308, 2001.
37. J. Popovic and H. Hoppe. Progressive Simplicial Complexes. In SIGGRAPH'97 Proceedings, pages 217–224, 1997.
38. J. Rossignac. Geometric Simplification and Compression in Multiresolution Surface Modeling. SIGGRAPH'97 Course Notes #25, 1997.
39. H. Samet. The Quadtree and Related Data Structures. ACM Computing Surveys, 16(2):187–260, 1984.
40. C. Saona-Vàzquez, I. Navazo, and P. Brunet. The Visibility Octree: A Data Structure for 3D Navigation. Computers & Graphics, 23(5):635–643, 1999.
41. F. Sillion, G. Drettakis, and B. Bodelet. Efficient Impostor Manipulation for Real-Time Visualization of Urban Scenery. In Proceedings of Eurographics'97, pages 207–218, 1997.
42. G. Suter and A. Mahdavi. Performance-Inspired Building Representations for Computational Design. In Proceedings of Building Simulation, Vol.3, pages 1203–1210, 1999.
43. S. J. Teller and C. H. Sequin. Visibility Preprocessing for Interactive Walkthroughs. In SIGGRAPH'91 Proceedings, pages 61–69, 1991.
44. S. Teller. Visibility Computations in Densely Occluded Environments. Ph.D. thesis, University of California, Berkeley, 1992.
45. M. Wand, M. Fischer, I. Peter, F. M. auf der Heide, and W. Straßer. The Randomized Z–Buffer Algorithm: Interactive Rendering of Highly Complex Scenes. In SIGGRAPH'01 Proceedings, pages 361–370, 2001.

46. A.Watt and M. Watt, Advanced Animation and Rendering Techniques: Theory and Practice, Addison-Wesley, 1997.
47. P. Wonka. Occlusion Culling for Real-Time Rendering of Urban Environments. Ph.D. Thesis, University of Vienna, 2001.
48. P. Wonka, M. Wimmer, and F. X. Sillion. Instant Visibility. In Proceedings of Eurographics, 20(3):411–421, 2001.
49. J. C. Xia, J. El-Sana, and A. Varshney. Adaptive Real-Time Level-of-Detail Based Rendering for Polygonal Models. IEEE Transactions on Visualization and Computer Graphics, 3(2):171–183, 1997.
50. K. Yamaguchi, T. L. Kunii, K. Fujimura, and H. Toriya. Octree-Related Data Structures and Algorithms. IEEE Computer Graphics and Applications, 4(1):53–59, 1984.
51. H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff III. Visibility Culling Using Hierarchical Occlusion Maps. In SIGGRAPH'97 Proceedings, pages 77–88, 1997.