Institut für Computergraphik
Technische Universität Wien

Karlsplatz 13/186/2
A-1040 Wien
AUSTRIA

Tel: +43 (1) 58801-18675
Fax: +43 (1) 5874932

Institute of Computer Graphics
Vienna University of Technology

email:
technical-report@cg.tuwien.ac.at

other services:
http://www.cg.tuwien.ac.at/
ftp://ftp.cg.tuwien.ac.at/

Lazy Occlusion Grid Culling

Heinrich Hey   Robert F. Tobler

TR-186-2-99-09
March 1999

Abstract

We present Lazy Occlusion Grid Culling, a new image-based occlusion culling technique for rendering of very large scenes which can be of general type. It is based on a low-resolution grid that is updated in a lazy manner and that allows fast decisions if an object is occluded or visible together with a hierarchical scene-representation to cull large parts of the scene at once. It is hardware-accelerateable and it works efficiently even on systems where pixel-based occlusion testing is implemented in software.

Lazy Occlusion Grid Culling

Heinrich Hey Robert F. Tobler

Vienna University of Technology
{hey,rft}@cg.tuwien.ac.at

Abstract. We present Lazy Occlusion Grid Culling, a new image-based occlusion culling technique for rendering of very large scenes which can be of general type. It is based on a low-resolution grid that is updated in a lazy manner and that allows fast decisions if an object is occluded or visible together with a hierarchical scene-representation to cull large parts of the scene at once. It is hardware-accelerateable and it works efficiently even on systems where pixel-based occlusion testing is implemented in software.

1 Introduction

Occlusion culling is very important for efficient rendering of very large scenes. Such scenes can consist of millions of polygons which is much more than available graphics hardware can render in useable time. Usually only a small part of the scene is visible. Therefore it is necessary to determine all those parts of the scene that are completely invisible due to occlusion by other parts so that these occluded objects do not have to be considered for drawing.

In this paper we present Lazy Occlusion Grid Culling, a new image-based occlusion culling technique that, in contrast to many existing techniques as explained in chapter 2, is capable of efficiently handling general scenes. It is hardware-accelerateable on systems that support simple pixel-based occlusion testing in hardware eg. as already proposed for OpenGL [3], [9]. It works efficiently even on systems where this pixel-based occlusion testing part is implemented in software due to missing hardware-support by the graphics system.

The image is subdivided into a low-resolution grid of cells with an occlusion-status for each cell. This low-resolution grid allows fast determination of occlusion or visibility of objects in large image-areas. The occlusion-status of a cell is updated in a lazy manner upon request of the cell-status if the image-area of the cell has been modified. This image-based occlusion testing is used together with a hierarchical scene-representation like an octree, a BSP-tree or any other kind of bounding volume hierarchy so that large occluded parts of the scene can be culled at once.

2 Related Work

Several different occlusion culling techniques already exist. They can be categorized if they operate on geometry or if they are image-based. An important feature is if and how much a technique allows hardware-acceleration, which applies to some of the image-based techniques, and if the required hardware is available. Many techniques require a preprocessing step and are limited to static environments which is the case with most potentially visible set-methods (see below). Also many techniques are only meant to work with scenes that are densely occluded or that have some special geometric characteristics.

Naylor [13] presented a method which realizes occlusion culling by generating a 2D BSP-tree of the image from a 3D BSP-tree of the scene. Occlusion culling methods that are specialized for 2½D scenes use the simplicity of the geometry of such scenes which is based on a ground plan [14], [19].

Potentially visible set (PVS)-based methods divide space into convex cells. Only the objects contained in a limited number of cells (the PVS) may be partially visible from the eyepoint or from somewhere in the cell containing the eyepoint and must therefore be drawn. PVS-based methods are suited for models like indoor-scenes, caves or pipes which consist of cell-structures so that the PVS are small. PVS are not suited eg. for outdoor-scenes unless they are very densely occluded as assumed in the method of Cohen-Or et al [4]. If a scene does not offer the required inherent cell-structure it would cause very large PVSs.

The PVS of the actual eyepoint can be computed by finding visible sequences of portals which are the connecting holes like doors or windows in the walls of the cells [12]. In static scenes the PVSs of all cells can be calulated in a preprocessing step. This can be done by determining the objects that are visible from a portal and which therefore belong to the PVS of the portal’s cell [2]. The PVS of a cell can also be determined by recursively searching for sequences of portals through which sightlines exists [17]. In 2D this is the case if the left and right vertices of the portals are linearly separable. The transfer of this method to 3D requires the replacement of the portals with their axisaligned bounding boxes. Paths of possible visibility are used to determine the PVSs in a method which is based on a regular grid in object-space and which is suited for cave-like structured scenes [20].

An example for the usage of PVSs on low-end hardware is the 3D game Quake [1] which generates the PVSs of its static environment as BSP-trees in a time-expensive preprocessing step and which then uses the BSP-tree of the actual cell for depth-sorting.

Some techniques only use a small set of heuristically choosen occluder-polygons for culling [5], [10], [21]. This is based on the assumption that these few polygons cover a large area of the image and therefore occlude large parts of the scene. This assumption is problematic because there is no occlusion culling possible in the areas that are not covered by these few occluders and often many small polygons together form a larger occluding area.

Image-based techniques are not limited to scenes with a distinctive cell-structure like in indoor scenes. They are usually also suited eg. for forest-scenes where large parts of the scene are occluded by a set of many small objects.

The hierarchical z-buffer [6] is an image-based method that uses a pyramid of z-values to cull objects in large already completely occluded parts of the image with only a few z-comparisons. The scene is subdivided into an octree to realize a hierarchy of bounding volumes which are tested against the hierarchical z-buffer. If a bounding volume is completely occluded then its sub-objects are also occluded and can be culled. This method can be extended to error-bounded antialiasing [7]. Today there is still no hierarchical z-buffer hardware available and the implementation of the hierarchical z-buffer on conventional z-buffer hardware or in pure software is not very efficient.

Sudarksy and Gotsman presented how to use dynamic scenes with temporal bounding volumes together with the hierarchical z-buffer and the image-BSP-tree method [15], [16].

Hierarchical coverage masks [8] use a pyramid which only stores if an image-area or (sub)pixel is occluded instead of a z-value. Also in contrast to the hierarchical z-buffer each entry in the pyramid has 8x8 instead of 2x2 subentries, therefore the pyramid has fewer levels. The major characteristic of this technique is that it uses fast table-lookups and bit-operations instead of traditional scanline-rasterization. Geometry must be traversed in exact front back order. Traditional graphics-hardware can be used for texturing and shading but the main hierarchical coverage masks algorithm which processes on the coverage pyramid has to be done in pure software.

Hierarchical occlusion maps [21] work with a pyramid of averaged occlusion values which is generated using mipmap-hardware. Therefore the occlusion-determination is only approximative (non-conservative) and objects which should be visible may be culled. The occlusion-pyramid is only build initially for a small set of heuristically choosen occluder-polygons, which is not suitable for many types of scenes.

A brute-force technique is to test for occlusion by simply rasterizing a bounding volume without modifying any buffer and querying whether any fragment passed the z-test [6], [9]. In a more hardware-oriented proposal [3] this can be done for several parts of the image in parallel. Another idea is to do only a sparse sampling of the pixels covered by the bounding volume to test for occlusion [11].

Occlusion culling in walkthrough-scenarios can be done by rendering the scene up to a given distance (near field) into a software-opacity buffer [18]. After that the pixels that are not occluded by the near field are filled with the rest of the scene (far field) using horizon estimation, image caching and ray casting.

3 Lazy Occlusion Grid Culling

The image is subdivided into a low-resolution grid of cells and each cell contains the occlusion-status of its image-area. The optimal grid-resolution or number of pixels per cell is system-dependent and can be determined heuristically by testing typical scenes of the desired application at different grid-resolutions. The grid is based on a z-buffer or an occlusion-buffer where a single bit per pixel shows if that pixel is free or already occluded. Both kinds of buffer are updated conventionally when drawing primitives. The z-buffer variant has the advantage that the objects in the scene can be processed in an approximative near to far order. The occlusion-buffer variant needs an exact near to far order but has the property that a once set pixel is definitively occluded. Therefore it has the advantage that the pixel-tests (see below) are less expensive.

3.1 Occlusion-Buffer Variant

In the occlusion-buffer variant each cell of the grid stores its status which can be

- completely free
- partially free
- completely occluded
- additionally it can be modified

A cell which is completely covered by a bounding volume is called a non-border cell of that bounding volume. A cell which is only partially intersected by a bounding volume is called a border cell of that bounding volume (see figure 1).

To summarize the major occlusion testing steps which are described in detail below, a bounding volume is visible if it intersects a

- completely free cell
- partially free non-border cell

A bounding volume is occluded in the area of a

- completely occluded cell

If these tests have not classified the bounding volume as visible or occluded, a pixel-based occlusion test is done for a

- partially free border cell
- modified not completely occluded cell

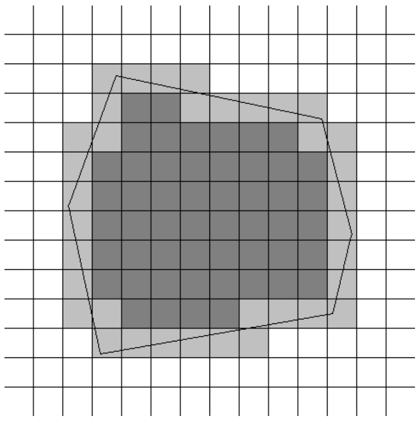

Fig. 1. Border (light grey) and non-border (dark grey) cells of a bounding volume

A bounding volume that intersects a cell which is completely free is visible. Initially all cells are completely free, therefore the first objects in their respective image-area can be classified as visible very fast. A bounding volume is also visible if it has a partially free non-border cell, because the bounding volume covers the whole cell and therefore at least one of the free pixels. If a cell is completely occluded it occludes the bounding volume in its area no matter if it is a border or a non-border cell. If a border cell is partially free the bounding volume may be occluded by the content of the cell. Therefore it is necessary in this case to test if there exists a free pixel that is covered by the bounding volume within the cell. This pixel-test is important because without it bounding volumes would have to be classified as visible unless the cell is completely occluded, eg. objects behind the horizon shall be occluded but the horizon does not completely occlude its intersected cells.

A bounding volume is classified as occluded if it is occluded in all cells it intersects. In this case the bounding volume and all its sub-objects can be culled. Otherwise the bounding volume is visible and its sub-bounding volumes are processed recursively. A visible bounding volume that contains primitives (usually a leaf-node in the scene-hierarchy) draws them conventionally and marks all of its intersected cells as modified. If later another bounding volume requests the status of a modified cell that is not already marked as completely occluded a pixel-test is applied to this cell to determine its status. After this the cell's modified flag is of course cleared.

This lazy evaluation of the cells' status has the advantage that several bounding volumes can draw their primitives and therefore increase the probability of occluding an area before the pixel-testing has to be done, therefore reducing the number of pixel-tests. In the case of a non-border cell the whole cell-area is tested if it contains a free pixel and accordingly to the result the cell is marked as partially or completely occluded. We do not test if it is completely free because this would require to test if all pixels in the cell-area are free. It is very probably that the intersected cells of a tight leaf-bounding volume are really modified by its primitives, therefore we can end the pixel-test and say that the cell is most likely partially occluded if we find the first free pixel. In the case of a modified border cell only the cell-area covered by the bounding volume is pixel-tested to determine if the bounding volume is occluded in the cell. This is done because often there exist many small bounding volumes and in this way at most only the image-area of such a bounding volume has to be pixel-tested but not the considerably larger area of its border cells.

All pixel-tests of a bounding volume are done at last after the faster status-based testing of all its cells has been done. Therefore pixel-tests have to be done only if the bounding volume has not already been classified as visible by means of the faster status-based testing in one of its cells.

3.2 Z-Buffer Variant

In the z-buffer variant each cell stores the farest z-value of its pixels (z_{far}) and status-flags if the cell is completely free or if the cell is modified. Initially all cells' z_{far} are set to z_{max} and their status is completely free, therefore an intersecting bounding volume is visible. A bounding volume is completely occluded in a cell's area if the nearest z-value (z_{near}) of the bounding volume is greater than z_{far} of the cell. Otherwise the cell is partially free relative to the bounding volume's z_{near}. In this case the bounding volume is visible if it is a non-border cell. Similar to the occlusion-buffer variant a pixel-test is necessary if the partially free cell is a border cell of the bounding volume or if the partially free cell is modified. In the case of a border cell the area of the bounding volume in the cell is tested for a free pixel. In the case of a modified non-border cell its actual z_{far} is determined according to the z-values of all its pixels. In contrast to that we can end the non-border cell pixel-test in the occlusion-buffer variant if we find the first free pixel.

3.3 Extensions

A possible extension to further enhance efficiency is to quickly rate a cell as free instead of pixel-testing if the probability that the cell is occluded is smaller than a given threshold. This probability can be approximated by adding up the size of the modified areas in the cell which for reason of speed can be done without considering if the areas are intersecting.

Another extension is to test if a bounding volume's border cell is completely occluded (which requires to test at most all pixels in the cell's area) if the area of intersection of the bounding volume and the cell is larger than a given threshold. In this case the overhead of testing in the whole cell-area instead of testing in the smaller intersection-area is not so big compared to the possibility to detect that the cell is completely occluded.

3.4 Hardware Acceleration

The most suited hardware-support for Lazy Occlusion Grid Culling would be given on systems that support pixel-based occlusion testing. This means that the result of the z-buffer or occlusion-buffer test can be requested after rasterization of a bounding volume, giving information if at least one pixel is visible or if all pixels are occluded. Writing to any buffer must of course be disabled during this rasterization step. Such functionality has been already proposed for OpenGL [3], [9], [11] and we hope that it will be soon available.

The next possibility to incorporate hardware is to access the hardware-z-buffer for reading and to do the pixel-testing in software. In OpenGL a part of the z-buffer can be read with glReadPixels. The speed of this function, which is able to do much more than a simple memcopy, varies much between different systems but on several workstations it is implemented quite fast.

On systems where the access to the hardware-z-buffer is not efficient enough a software-occlusion-buffer or -z-buffer is used for pixel-testing parallel to the hardware-z-buffer which is furthermore used to render the primitives in hardware. The occlusion-buffer has the advantage that it is faster than the z-buffer in software because the occlusion-buffer implementation needs to access only one byte per pixel and it does not have to calculate and compare the pixels' z-values.

3.5 Comparison to existing image-based techniques

In comparison to the Hierarchical Z-Buffer [6] which does not work efficiently if implemented on existing graphics-hardware or in pure software, Lazy Occlusion Grid Culling can be hardware-accelerated on systems that support a fast access to the z-buffer or that support a z-test query and it works efficiently even if this hardware-functionality is not available and therefore is implemented in software. A z-test query alone would not be enough for the Hierarchical Z-Buffer because the expensive hierarchical updates of the z-buffer-pyramid would still have to be done in software. Graphics hardware can be used together with Hierarchical Coverage Masks [8] only for shading and texuring because the handling of the Hierarchical Coverage Masks can only be done in software. In contrast to Lazy Occlusion Grid Culling, Hierarchical Occlusion Maps [21] only supports approximative occlusion culling because of the bilinear interpolation used to calculate the maps. Also other than Lazy Occlusion Grid Culling which uses all visible polygons as occluders, Hierarchical Occlusion Maps only uses a small number of occluders, therefore it is not suited for general scenes where many polygons together form a large occluding area or where the scene is too complex to accept unoccluded image-areas where no occlusion culling is done.

4 Implementation and Results

We have tested Lazy Occlusion Grid Culling on a 266 MHz PC with a Voodoo2 3D accelerator under OpenGL, a typical setup as it is used for today’s mainstream entertainment software. We have implemented both the basic occlusion-buffer and z-buffer variant. Reading the z-buffer with glReadPixels, which is needed for the z-buffer variant, is an extremely slow operation on the Voodoo2, therefore we made the testing with the occlusion-buffer variant that uses a software-occlusion buffer parallel to the usual harware-based rendering.

The test scene in figure 2 contains 32x32x32 (32.786) cubes (393.216 triangles) in the small version and 128x128x128 (2.097.152) cubes (25.165.824 triangles) in the large version. The test scene in figure 3 is a city model which consists of 156.518 triangles in the small version and 2.499.248 triangles in the large version. The large version of a scene contains the small version plus additional occluded scene parts to test how efficiently the culling in such very large scenes works. The scenes are build upon a hierarchy of axisaligned bounding boxes. Both images contain regions where already the foremost objects occlude everything behind them as well as regions which offer a far view where many smaller objects are visible that together form an occluding area. The images have been rendered with and without Lazy Occlusion Grid Culling. In both cases the hierarchy of bounding boxes was used to hierarchically cull the scene-parts outside the viewing frustum. The size of the images is 640x480 and the size of the cells has been set to 16x16 pixels per cell which has proven to be the best choice for the platform we have used.

		without LOG Culling		with LOG Culling		speedup
cubes small	393.216 tri	1,025 s	242.496 tri	0,195 s	18.816 tri	5,3
cubes large	25.165.824 tri	55,235 s	13.044.384 tri	0,207 s	18.816 tri	266,8
city small	156.518 tri	0,701 s	106.294 tri	0,034 s	2.133 tri	20,6
city large	2.499.248 tri	11,036 s	1.691.806 tri	0,034 s	2.133 tri	324,6

Table 1. Rendering times and triangles rendered with OpenGL with and without Lazy Occlusion Grid Culling

The rendering times in table 1 show that Lazy Occlusion Grid Culling achieves a high speedup for all four scenes, although the occlusion-buffer had to be implemented in software. The rendering times of the large versions of the scenes are only a little bit longer than the rendering times of the small versions (in the city they are practically the same) which shows that the very large additional scene parts are culled very efficiently. The current implementation which has been done only in C++ could of course be further improved by doing careful low-level optimizations of critical code-sections.

5 Conclusion

We have presented an image-based occlusion culling technique which is capable of handling very large scenes of general type. By using the lazy updated occlusion information in the low-resolution grid together with a hierarchical scene-representation large occluded scene-parts can be culled very fast. We showed that it can be used with different kinds of hardware-support and our results show that it works efficiently even on systems where software replaces missing hardware-support for pixel-based occlusion testing.

References

1. M. Abrash. Inside Quake: Visible Surface Determination. *Dr. Dobb's Sourcebook* January/February 1996 pp. 41-45
2. J. M. Airey, J. H. Rohlf and F. P. Brooks Jr. Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments. *Symposium on Interactive 3D Graphics* pp. 41-50, 1990
3. D. Bartz, M. Meißner and T. Hüttner. Extending Graphics Hardware For Occlusion Queries In OpenGL. *EUROGRAPHICS/SIGGRAPH workshop on graphics hardware 1998*
4. D. Cohen-Or, G. Fibich, D. Halperin and E. Zadicario. Conservative Visibility and Strong Occlusion for Viewspace Partitioning of Densely Occluded Scenes. *EUROGRAPHICS ´98 Proceedings*, pp. 243-253, 1998
5. S. Coorg and S. Teller. A Spatially and Temporally Coherent Object Space Visibility Algorithm. MIT LCS Technical Report 546, 1996
6. N. Greene, M. Kass and G. Miller. Hierarchical Z-Buffer Visibility. *Computer Graphics (SIGGRAPH ´93 Proceedings)* pp. 231-238, 1993

7. N. Greene and M. Kass. Error-Bounded Antialiased Rendering of Complex Environments. *Computer Graphics (SIGGRAPH ´94 Proceedings)* pp. 59-66, 1994
8. N. Greene. Hierarchical Polygon Tiling with Coverage Masks. *Computer Graphics (SIGGRAPH ´96 Proceedings)* pp. 65-74, 1996
9. Hewlett-Packard. GL_HP_occlusion_test - preliminary. www.opengl.org/ Documentation/Version1.2/HPspecs/occlusion_test.txt, 1997
10. T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff and H. Zhang. Accelerated Occlusion Culling using Shadow Frusta. *ACM Symposium on Computational Geometry (SCG ´97)*, 1997
11. T. Hüttner, M. Meißner and D. Bartz. OpenGL-assisted Visibility Queries of Large Polygonal Models. ftp.gris.uni-tuebingen.de/pub/thuettne/ Occlusion_Culling/TechReport/techReport.pdf, 1998
12. D. Luebke and C. Georges. Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets. *Symposium on Interactive 3D Graphics* pp. 105-106, 1995
13. B. Naylor. Partitioning tree image representation and generation from 3d geometric models. *Graphics Interface ´92* pp. 201-212, 1992
14. D. Schmalstieg and R. F. Tobler. Exploiting coherence in 2½D visibility computation. *Computers & Graphics Vol. 21 No. 1 p. 121,* 1997
15. O. Sudarsky and C. Gotsman. Output-Sensitive Visibility Algorithms for Dynamic Scenes with Applications to Virtual Reality. *EUROGRAPHICS ´96 Proceedings* pp. 249-258, 1996
16. O. Sudarsky and C. Gotsman. Output-Sensitive Rendering and Communication in Dynamic Virtual Environments. *ACM Symposium on Virtual Reality Software and Technology (VRST) ´97* pp. 217-223, 1997
17. S. J. Teller and C. H. Séquin. Visibility Preprocessing For Interactive Walkthroughs. *Computer Graphics (SIGGRAPH ´91 Proceedings)* pp. 61-69, 1991
18. M. Wimmer, M. Giegl and D. Schmalstieg. Fast Walkthroughs with Image Caches and Ray Casting. to appear in *Virtual Environments ´99 (Proceedings of the 5th EUROGRAPHICS Workshop on Virtual Environments)*, 1999
19. P. Wonka and D. Schmalstieg. Occluder Shadows for Fast Walkthroughs of Urban Environments. to appear in *EUROGRAPHICS ´99 Proceedings*, 1999
20. R. Yagel and W. Ray. Visibility Computation for Efficient Walkthrough of Complex Environments. *Presence Vol. 5 No. 1* pp. 45-60, 1996
21. H. Zhang, D. Manocha, T. Hudson and K. E. Hoff III. Visibility Culling using Hierarchical Occlusion Maps. *Computer Graphics (SIGGRAPH ´97 Proceedings)* pp. 77-88, 1997

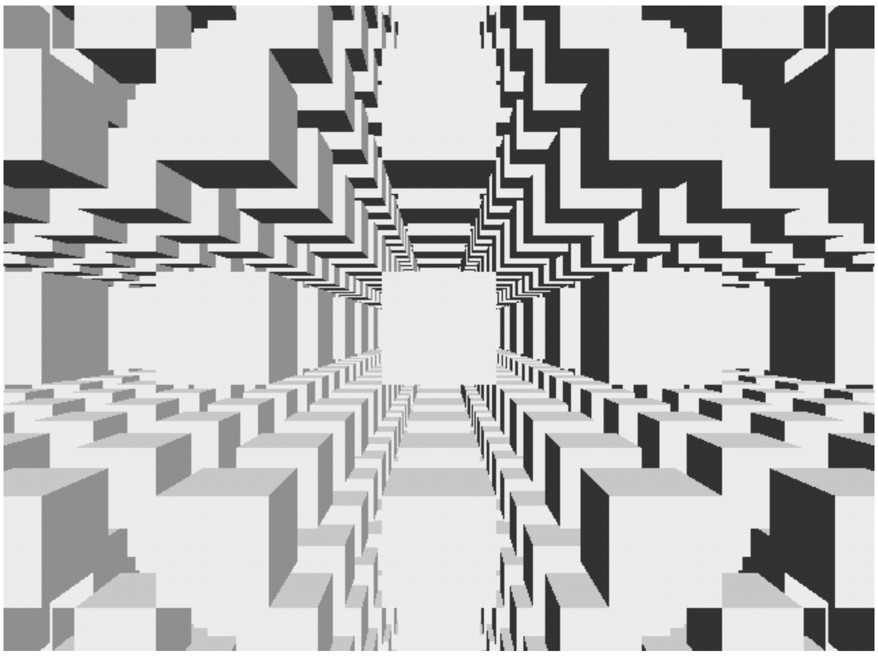

Fig. 2

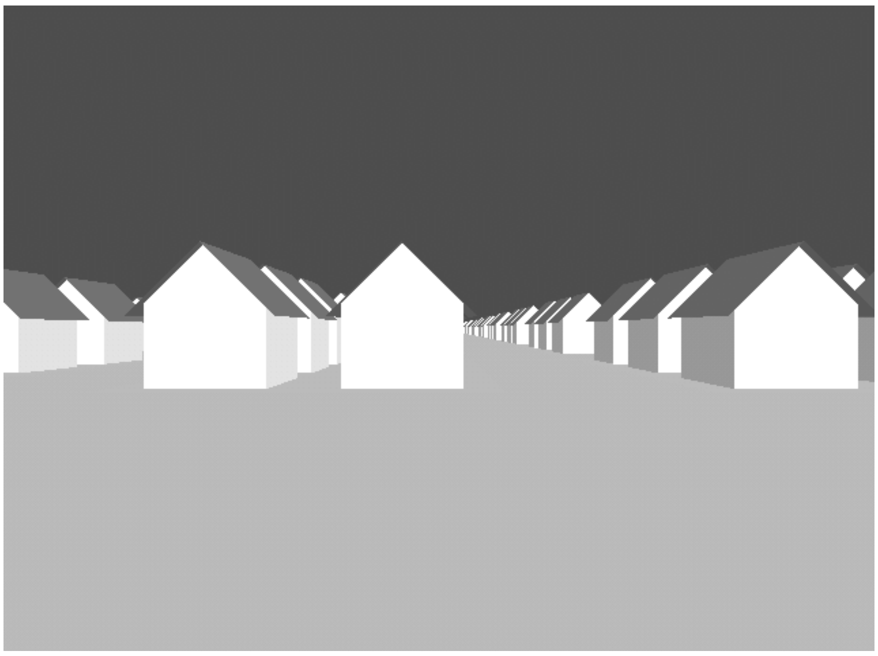

Fig. 3