Breaking the Walls: Scene Partitioning and Portal Creation

Alon Lerner
Tel-Aviv University
alan@post.tau.ac.il

Yiorgos Chrysanthou
University Of Cyprus
yiorgos@ucy.ac.cy

Daniel Cohen-Or
Tel-Aviv University
dcor@tau.ac.il

Abstract

In this paper we revisit the cells-and-portals visibility methods, originally developed for the special case of architectural interiors. We define an effectiveness measure for a cells-and-portals partition, and introduce a two-pass algorithm that computes a cells-and-portals partition. The algorithm uses a simple heuristic that creates short portals as a mean for generating an effective partition. The input to the algorithm is a set of half edges in 2D, that can be extracted from a complex polygonal model. The first pass of the algorithm creates an initial partition, which is then refined by the second pass. We show that our method creates a partition that is more effective than the common BSP partition, even when the latter is further refined with the application of our second pass. Our cells-and-portals algorithm is designed to deal with arbitrarily oriented walls. The algorithm also supports outdoor scenes, where the vertical walls of the buildings serve as occluders and portals are extended above the buildings. We show that the extended portals allow an output-sensitive rendering of large urban scenes. Finally, since our two-pass method is fully automatic and local, it supports incremental changes of the model by locally recomputing and updating the partition. We call our method "Breaking the Walls" (BW) since it breaks out of indoor scenes to outdoor scenes, and allows walls to be broken interactively, with an instant updating of the partition.

1 Introduction

In this paper we revisit visibility methods developed for the special case of architectural interiors. These methods were designed to exploit the prominent characteristics of architectural indoor scenes, namely, the natural partition of the scene into cells, typically rooms, and that visibility occurs through openings such as doors or windows, which are called portals, in this context.

In architectural interior models, visibility is limited to the immediate surroundings, while only a small fraction of remote geometry can be seen, if at all. The partitioning of the scene into *cells-and-portals* is today the state of the art in speeding up visibility calculations in the game industry.

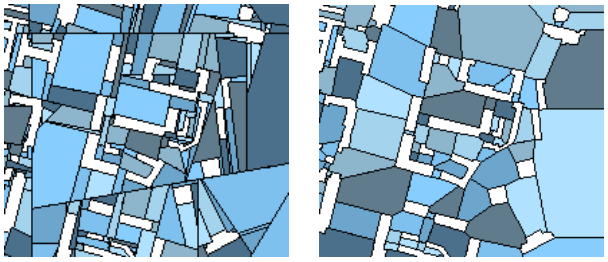


Figure 1. Bsp partition (left) vs. BW partition (right). The buildings are in white.

One of the limitations of the cells-and-portals concept is its automatic creation. To our knowledge, there is no automatic method for creating an effective partition, even for axis aligned models. A common practice is to let the artist define it manually. This is, of course, tedious work based solely on user intuition. In the game industry, manual partitioning may require relatively little effort in the level-design process, however, efficient automatic partitioning not only saves design work, but also optimizes runtime rendering speed, that saves precious computational resources for other tasks. Moreover, a local automatic partitioning algorithm allows to change the model dynamically and to recalculate the partition only for the area surrounding the change, therefore allowing more complex games, for example.

In a cells-and-portals partition, only the geometry within (partially) visible cells is sent to the graphics pipeline for rendering. The traversal of the visible cells incurs only a small overhead of testing the visibility of their portals. This test is extremely fast and consists of only a few simple geometric operations [10]. Nevertheless, it remains a challenge to compute an effective cells-and-portals partition.

For outdoor scenes, hierarchical space partitioning datastructures are commonly used. Their traversal necessarily imposes the overhead of testing the visibility of the upper levels of the hierarchy before reaching the visible cells at the leaves. These visibility tests are significantly more ex-

pensive than portal visibility tests, even when realized in hardware. Furthermore, as the size of the model grows, the path to the visible cells in the hierarchy also grows (even if logarithmically). As far as we know, cells-and-portals have not been used for outdoor scenes, and as we will show the method can be used for an outdoor scene provided that an effective partitioning can be found for the model.

Automatic cells-and-portals partitions are usually based on a Binary Space Partitioning (BSP) tree. As we will see later, this tends to create an excess of inefficient cells. Modifying a BSP partition can affect an extended area, especially if the modified edges come from high up in the tree [3].

The novel partitioning technique we describe in this paper is fully automatic, and generates an effective partition. It makes only local computations, therefore it supports online changes in the model. It is based on the heuristic that a good partition has short portals. The input for the algorithm is a set of half edges in 2D. The half edges can be obtained from a model by following the vertical faces that are at the base of the walls. After we have the partition, we define the portals as rectangles whose base is on the floor and their top is on the ceiling. Then, each face of the model is placed in the appropriate cell.

One of our contributions is an analysis and a quantitative measure for the effectiveness of a partition. Based on this measure we compare our method with the BSP based approach (see Figure 1). Our algorithm is designed to deal with arbitrarily oriented walls and it supports outdoor scenes, where the vertical walls of the buildings are used as occluders. To make this applicable, we extend the portals above the buildings to some predefined expected height. The extended portals allow efficient rendering of large urban scenes, in the sense that the geometry sent to the graphics pipeline is output sensitive.

We call our cells-and-portals algorithm "Breaking the Walls", denoted by BW, since conceptually it breaks the walls of the indoor scenes and moves on to outdoor scenes. Moreover, it also allows to break walls interactively in a walkthrough and quickly update the partition.

In the next section we review related work. In Section 3 we describe our partitioning algorithm, followed by a discussion on metrics for an effective partition in Section 4. The extension of the method to outdoor urban scenes is shown in Section 5. We conclude with some results for various indoor and outdoor scenes and a comparison to the BSP based partitioning in Section 6.

2 Related work

Indoor scenes saw some of the first applications of occlusion culling [1, 7, 13, 14]. In such scenes walls and other large elements serve as natural occluders while smaller elements are considered as geometric details. Non-opaque *portals*, such as doors and windows, connect adjacent *cells*, and together they define an *adjacency graph* where the nodes are associated with the cells, and the portals are the edges linking the adjacent connected nodes.

Cells might be able to see through other cells. Any two cells which have a sightline connecting them, are considered mutually visible. The geometry of all the visible cells from a given cell forms its potentially visible set (PVS). The PVS of each cell is stored and is readily available for rendering during the interactive walkthrough.

Luebke and Georges [10] propose a from-point cells-and-portals technique. Instead of precomputing the PVS's, the visible cells are computed on-the-fly during a recursive depth first traversal of the adjacency graph. Here the visibility is computed from a point, and a cell is considered visible if any of its portals is seen. The visibility tests for the portals are performed by projecting them into screen-space and testing the visibility of their image-space bounding boxes. The advantage of computing the visibility on-the-fly is that it applies a recursive view frustum culling through the sequence of portals, and thus the PVS is smaller than that of a from-cell one. Note that their algorithm does not require the cells to be convex.

Cells-and-portals techniques assume that the geometry is hidden and by testing the visibility of the portals, the visible geometry is detected. Alternatively, generic visibility techniques assume that the geometry is visible and cull regions which are detected as hidden [4]. Visibility culling techniques usually use a spatial hierarchy to represent the scene. The hierarchy is traversed top-down aiming at culling large portions of the scene early on [2, 5, 6, 8, 9, 17]. The traversal of the hierarchy necessarily incurs a logarithmic factor, which is avoided by the "flat" traversal of the adjacency graph in cells-and-portals techniques. However, it is far easier to define a spatial hierarchy than a cells-and-portals partition, in the general case. Even for indoor models, where the constrained architectural structure leads to a natural partition into cells, it is still not easy to compute an efficient partition. Typically, the partition is based on a 2D floor plan, taking advantage of the fact that the vertical walls are effective occluders. In this paper we show that also for outdoor urban models, it is possible to create an effective automatic cells-and-portals partition, assuming that the heights of the buildings are quite regular (see Section 5).

As mentioned in the introduction, it is desirable for the cells-and-portals definition to be automatic and scalable to arbitrary large and complex scenes. However, this is not the case with current known techniques.

Teller and Sequin [14] generate a partition of the model into convex cells using a BSP tree. They use as splitting planes of the tree only planes defined by the walls of the model. These are chosen in decreasing order of coverage, so for example if we have a long wall or a collection of

coplanar walls with few gaps between them, then the plane they define is more likely to be selected early. The leaves of the final BSP tree are the required convex cells. This, however, works well only if the walls are axis aligned or very regularly placed.

Meneveaux et al. [11] define a variation on the BSP cells. The vertical polygons of the scene are mapped onto points in a dual space. The points are then separated into clusters and for each cluster one representative point is chosen. The chosen point is then mapped back to primal space and is used as a splitting plane for the BSP. The splitting planes along with a priori construction rules are used to construct the cells. The construction rules define the shape of the cells. For example, most rooms are rectangular, therefore when applying this construction rule the splitting planes are be chosen in such a way that they create rectangular cells.

Partitioning with a BSP tree is mainly considered applicable for static scenes. The construction of the tree is an expensive operation which needs to be done at preprocessing. When one of the splitting planes is moved the tree is invalidated. There have been solutions proposed in the literature for allowing dynamic scenes [3, 12, 15], however they are often too complex to implement [12] and will usually perform well for a limited range of changes [3, 15].

Another related work is the unique work by Van de Panne and Stewart [16]. They introduce a method to compress the PVS of a partition by merging cells with similar PVS's. They deemed an effective partition as one where cells do not have similar PVS's and therefore do not compress.

3 The BW Algorithm

3.1 Preliminaries

Given a complex polygonal model, we select the vertical polygons, and extract from them the set of half edges that are given as input to the algorithm. The occlusion of non-vertical polygons is ignored. For each wall we define a half edge. The half edges are single-sided oriented edges, therefore, if both sides of a wall appear in the model, we define two half edges with opposite normals. Portals are transparent polygons that connect gaps between walls.

In the rest of the paper, we use the term *walls* to refer to the half edges that are given as input to the algorithm. A *portal* is a half edge that is created during the execution of the algorithm, it is defined as the shortest distance between two walls. Whenever a portal is created, both its half edges are created. By *edges* we refer to half edges which are either walls or portals. We say that two edges are *adjacent* if they share a vertex.

We assume that the walls do not cross each other and that there are no T-junctions. If such degeneracies exist, they can be resolved at preprocessing. During the execution of the algorithm walls might be split into several parts due to the creation of portals. Portals are not allowed to cross other edges.

A *cell* is polygonal region whose boundary is formed by a sequence of walls and valid portals. A cell is considered *valid* if the polygonal region defined by it does not contain any walls. In our partition, a cell is considered *optimal*, if it is a valid cell that cannot be split into several valid cells. The criterion for the validity of portals is defined in 3.4.

After the partition is created, the polygons of the model are associated with the appropriate cells, and the portals are redefined as rectangles that start from the floor and end at the ceiling.

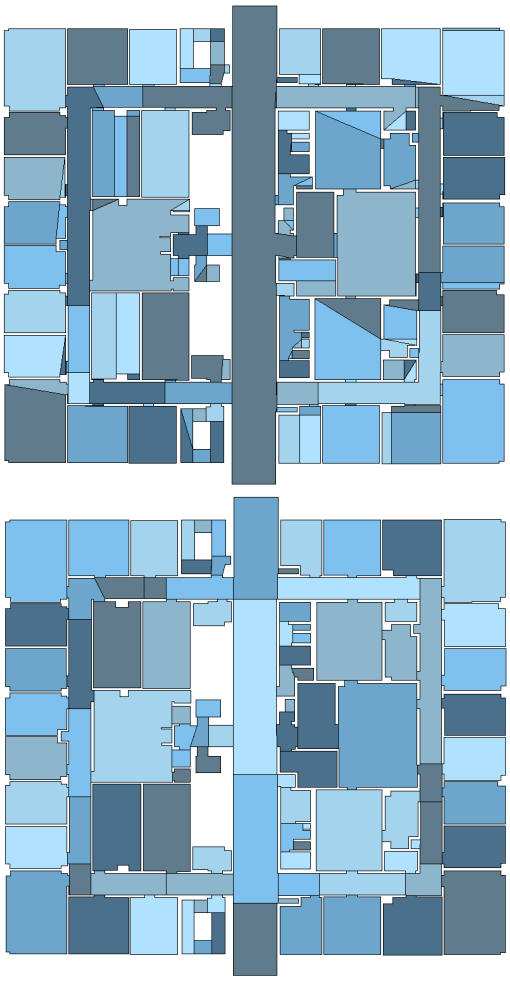


Figure 2. Soda model. (Top) The result of the first pass of the BW algorithm. (Bottom) The result of the second pass.

3.2 Overview

Given a set of half edges in the plane, we would like to add new edges, portals, and decompose the plane into connected cells. The rationale of our algorithm is to keep the cells large and avoid over-splitting. We strive to create cells with a small number of short portals. This is achieved through a two-pass locally greedy algorithm.

In the first pass we create an initial set of cells. The cells are created by traversing the walls, choosing the shortest step possible, until all the walls are classified to cells. At each step we either move to an adjacent wall, or to a close wall that creates a short portal. At this stage, we don't have any knowledge of the cells shape and size, therefore we can only base our decisions according to the length of the walls. Thus, we always create short portals relative to the adjacent walls.

In the second pass, we go through the cells and refine them. At this stage we know the size and shape of the cells, so we can split or merge them, striving to create cells where the portals are short relative to the cell's boundary length.

In the following we give a more detailed description of the two passes, and a description of how to update a partition when a change occurs in the model.

3.3 The First Pass - Initial Partition

The purpose of this pass is to link together walls and portals to create an initial set of cells. Most of the cells created in this pass are valid but not necessarily optimal (Figure 2 top).

The BW algorithm starts from an arbitrary wall and constructs a path by traversing the walls, in a counter-clockwise direction, adding at each step an adjacent edge (wall or portal) or creating the shortest possible portal by moving to the closest disjoint wall. Whenever the path closes up on itself, a cell is created, where the edges of the loop define its boundary. We continue the traversal from the end of the remaining path. If the remaining path is empty then we select a new seed. The above procedure is repeated until all walls have been used and classified to cells.

The core of the algorithm is the selection of the next wall in the traversal. Let us denote by Γ the current wall on which the algorithm is operating. First we examine the edges which are adjacent to Γ on its right endpoint. From these we select the edge that has the smallest internal angle with Γ (any other would lead to an invalid cell) and call it Γ_{adj}. Note that Γ_{adj} could be either a wall or a portal which was defined when a neighboring cell was created.

Γ_{adj} is not necessarily our best option at this stage. It may be possible to define a new portal, Δ, between Γ and the closest disjoint wall, Γ_{cls}, such that Δ is shorter than Γ_{adj} and it is located in the sub-space defined by the supporting lines of Γ and Γ_{adj}. Note that if Δ lies outside the aforementioned sub-space, then the cell, that will eventually be created by this path, will not be a valid one.

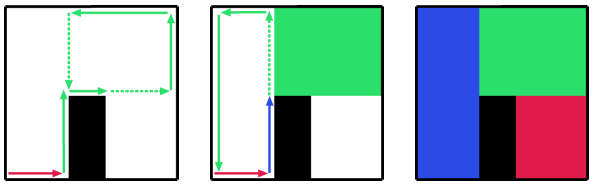


Figure 3. (a) Starting from the red edge, we follow the green path until it closes upon itself and creates the green cell in (b). Continuing from the blue edge, we follow the green path to create the blue cell in (c). Finally the red cell is created.

If Γ_{adj} is shorter than Δ, it is chosen as the next edge, otherwise Γ_{cls} is chosen and Δ is created. Note that if Γ_{adj} is chosen and it is a portal, we move to its adjacent wall on its right endpoint. An example of a first pass traversal is shown in Figure 3.

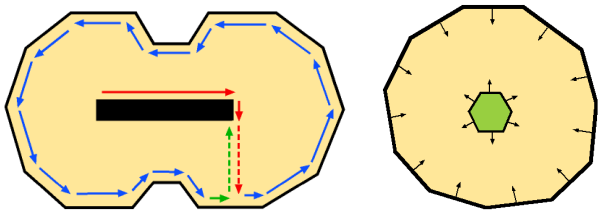
Occasionally, during the above process, a cell that encloses other geometry within it might be created, which of course is not allowed. Most likely, this is resolved when we traverse the enclosed geometry. As we follow the internal walls a portal may link them to the boundary of the surrounding cell. We then break up that cell, add its boundary to the current path, and continue with the traversal (Figure 4(a)). However, we may have a case such as that of Figure 4(b), where the enclosed geometry closes the loop on itself and fails to connect with its surroundings. We deal with these cases separately at the end of the first pass, by connecting them to their surrounding cells using the shortest portal between the two.

3.4 The Second Pass - Refinement

During the first part of the algorithm we created an initial partition of the model. Not all of the portals that were created are valid portals. We define a portal as *valid* if the ratio between its length and the length of the boundary of its cell is less than a predefined value α.

A cell is defined as underestimated if it contains an invalid portal (see Figure 5(a)). Underestimated cells are merged with the cell on the opposite side of the invalid portal.

A cell is defined as overestimated if it can be split, with a valid portal, into two valid cells (see Figure 5(a)). A split can occur only at a right turn, that is, where two consecutive edges have an internal angle greater than 180^o. We traverse



(a) (b)

Figure 4. Enclosed geometry, (a) Connects to the surrounding cell through a portal (dashed red). Traversal continues with the green path. (b) Forms an isolated cell. After the first pass, we connect it to the surrounding cell using the shortest portal that can be created between them.

the walls of the cell counter-clockwise and for each right turn we define the shortest valid portal emanating from one of the edges defining the turn. This defines a set of potential splitting portals for the given cell. We pick the shortest portal in the set, and split the cell accordingly. The two new cells may be split further. Note that an optimal cell has an empty potential splitting portal set, while an overestimated cell has at least one potential splitting portal (see Figure 5(b)). To detect the overestimated cells, we visit all the cells, and for each one, test whether a new valid portal can split it.

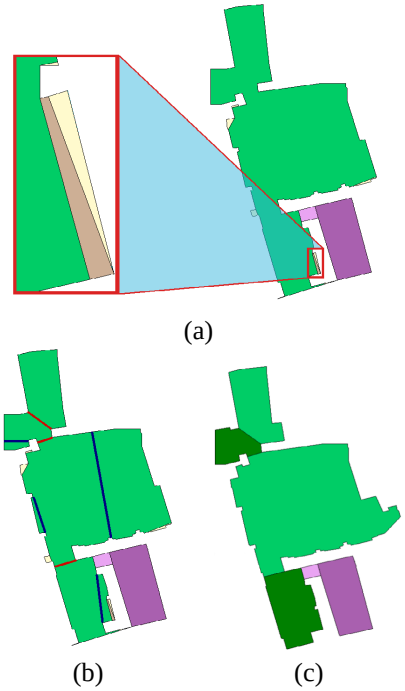
The second pass starts by splitting overestimated cells and then merging underestimated cells. The resulting partition consists of optimal cells requiring no further splits or merges (see Figure 2 bottom and Figure 5(c)).

3.5 Breaking walls

Sometimes we would like to change parts of the model without recomputing the entire partition. Since the algorithm is local, we can make local changes and get an updated valid partition, see Figure 6.

Adding or removing walls requires that we remove the cells surrounding these walls from the current partition and add their edges to a list of unclassified edges. Portals which connect two removed cells are deleted. If new walls are inserted into the scene, they are added to the list of unclassified edges. If walls are removed, their edges are removed from the list. Now, the BW algorithm creates a new partition for the unclassified edge list as before, and the new cells are inserted into the partition.

All the cells in the current partition, including the newly created ones, adhere to the definition of optimal cells in the BW algorithm. Therefore, the entire partition is a valid partition.



(a) (b) (c)

Figure 5. A small region of the London model. (a) The green cell is an overestimated cell. Brown cells, shown in closeup, are underestimated cells. Purple cells are optimal cells. (b) Red portals are valid splitting portals, while blue are invalid. (c) The final partition.

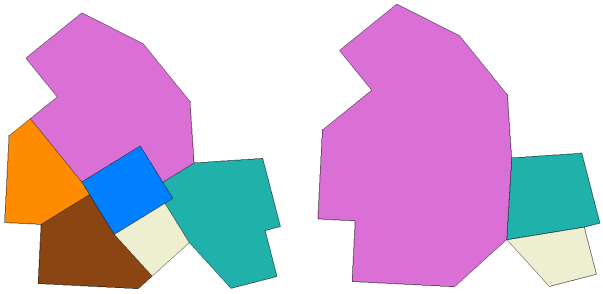


Figure 6. Removing the blue building. (a) The building and its immediate neighboring cells. (b) The partition after the buildings removal. This operation took less than 1 msec.

4 Effective Partition

A central question is: what is an effective partition, and in particular, what is an effective cells-and-portals partition? There are two aspects to be considered: time and space. Before we analyze the effectiveness, we should distinguish

between two applications of cells-and-portals. In the first, the PVS of each cell is precomputed and stored, so that it is readily available during the walkthrough [14]. In the second, the PVS is determined on-the-fly during the walkthrough [10]. The criteria are different in these two cases. The first is storage intensive [16], while the latter is computation intensive. Here we are interested only in the latter case, where the storage requirement is merely the adjacency graph. However, for this to be effective, the run-time computation needs to be optimized.

The runtime per-frame cost consists of two interdependent parts: the rendering time, which is directly dependent on the size of the PVS, and the time required to compute the current PVS. There should be a balance between the effort needed to compute a tighter PVS versus the time saved by not rendering hidden geometry. For example, if a partition consists of many tiny cells, the PVS will most likely be smaller than that of a partition which consists of a few large cells, but then it will require more time to classify the visible portals for a given viewpoint.

The following equation gives us a concrete measure for the effectiveness of a partition. Assuming that objects are distributed evenly in the scene, we can compute the runtime cost of using the partition for rendering from a point p as follows:

$$C_p = \sum_i (W_1 * A_i + W_2 * P_i * K_i),$$

with i ranging over all the visited cells, A_i the area of cell i, P_i the number of portals in cell i and K_i the number of times the cell was encountered during the rendering from the current viewpoint. The weights W_1 and W_2 are two positive constants that formulate the relative cost of rendering the cells and testing the visibility of the portals respectively. The weight W_1 depends on the performance of the graphics card, and W_2 depends on the computational power of the CPU.

Basically, we want our partition to minimize the average cost of rendering from a point including the portal visibility tests. Since this measure cannot be evaluated analytically, we approximate it by sampling the visibility from a large number of viewpoints within the cells of a given partition. In each sample we count the running time and the number of portals tested. Based on many samples we can approximate the relative values of W_1 and W_2.

5 Going Outdoors

Cells-and-portals are extended to outdoor scenes by creating *extended portals* as depicted in Figure 7. We define a height, H_c, which we consider to be the ceiling. H_c is set such that the highest point of most buildings is below H_c. Buildings that end above H_c are clipped to H_c. The parts above H_c are considered to be non-occluding. Short buildings are also considered to be non-occluding.

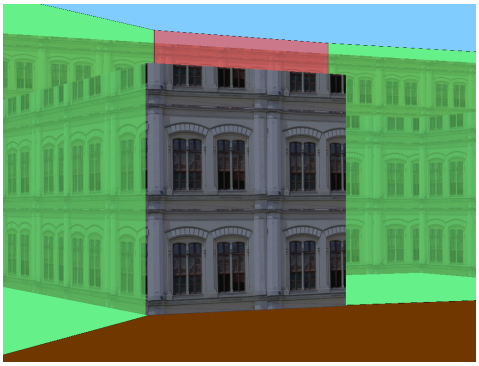


Figure 7. Part of the Vienna model. The portals are extended up to H_c. The green portals start from the ground, while the red portals start from the roofs of the buildings.

We create two separate partitions that are then merged and used as one. The first partition is a *ground-level* partition. This is a cells-and-portals partition of the empty space between the buildings. In this partition each portal is a vertical rectangle emanating from the bottom of the building up to the H_c. The second partition is a *roof-top* partition. This means that the roofs of the buildings are partitioned into cell-and-portals (see Figure 13). In this partition for each edge, wall or portal, we create an extended portal from the top of the building to H_c (see Figure 7). We connect the two partitions by adding the extended portals created on top of the buildings to the appropriate cells of the ground partition.

The geometry above H_c defines a separate geometry scene which is rendered separately. However, this geometry is usually occluded as it is mostly above the elevated view frustum defined above H_c.

We extract the half edges given to the algorithm, for the creation of the cells-and-portals partitions, from the vertical walls of the buildings. For each wall we extract two half edges, one facing the street, which has the same normal as the wall, the other facing the into the building, which has the normal in the opposite direction. The half-edges facing the street are used to create the ground-level partition, while the half-edges facing the buildings are used to create the roof-top partition.

The effectiveness of a cells-and-portals technique for outdoor scenes depends on the characteristics of the scene. As discussed above, to be effective, portals must be small. The portals emanating from the ground to H_c were already in the partition in the 2D case. The portals that emanate from the tops of buildings are the extra cost of going out-

doors. If the height of most of the buildings is close to H_c then the cost is fairly small: the extra portals defined above the buildings are rather small, and there is little geometry above H_c. On the other hand, if the scene exhibits a large variety of buildings, then large portals are defined above short buildings, and high buildings add a lot of geometry above H_c. We need to define the height H_c so that unusually high buildings do not extend the portals too much and thus make the portals ineffective. Note that in typical large cities, like London or Vienna, most of the buildings have about the same height (see Figure 8). If there are regions in the model that have higher buildings than others, then one can define H_c regionally.

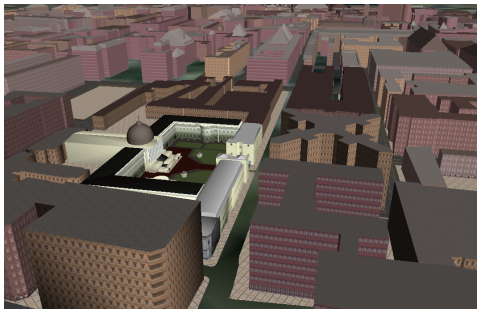
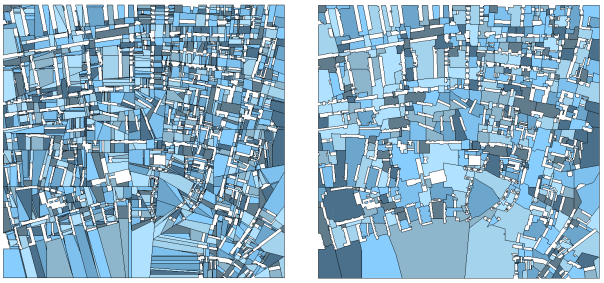


Figure 8. Part of the London model. Typically, in large cities buildings have about the same height.

To combine indoor and outdoor models, one needs to create a partition for the indoor model and another partition for the outdoor model and then combine the two resulting adjacency graphs into one.

6 Results and Discussion

We implemented the BW algorithm and integrated it into a cells-and-portals culling mechanism for an interactive walkthrough system. To test and evaluate the performance of our algorithm, we used four different models: A region of a model of London, the indoor Soda model, the sparse outdoor Sava model, and the Vienna 2000 model. The layouts and cell-and-portals partitions of the London model and the Vienna model can be seen in Figures 12 and 13. A straightforward BSP partition yields a very poor partition (Figure 9(a)), therefore we applied the second pass of our algorithm on the initial BSP partition (Figure 9(b)). Following the discussion in Section 4 the effectiveness of the partition for rendering is evaluated by measuring the average number of portals tested, and the average area rendered from an arbitrary point. Table 1 shows the effectiveness of the partition in comparison with a post-merge BSP partition for three of the models. For the London model, the BW partition clearly outperforms the post-merge BSP partition as it has an average of about six time less portals, and it renders less area (72.551 vs. 107.986). Interestingly enough, the post-merge BSP has less cells than the BW partition. This emphasizes the fact that the effectiveness of the partition is not a function of the number of cells. As shown in Table 1, the experiment with the Sava model yields similar results. The Soda model has different characteristics as it is an indoor model with axis-aligned walls. The results show that the BSP partition does not gain much from the simplicity of the model and the BW algorithm still yields a better partition. As can be seen in Table 1, the average number of visible cells in the merged BSP partition, is about the same as in the BW partition, while the number of tested portals is several times higher. The reason is that at creation, the BSP splitting planes split the portals as well as the cells. Resulting in cells that have on the boundary a sequence of portal segments. While keeping these portals is clearly redundant, it is not clear how it can be avoided.



(a) (b)

Figure 9. A region of the London model. (a) BSP partition. (b) BW second pass applied to the BSP cells.

The running time of the BW algorithm depends on the size of the model. On a P4-2Ghz computer it takes the algorithm a fraction of a second to calculate the partition (including the adjacency graph) for the Soda and Sava models. The London model is partitioned in less than thirty seconds while the Vienna model is partitioned in about ten minutes. As discussed in Section 3.5, small modifications to the model require only a local computation. Modifying and updating the partition in the Sava model, for example, takes less than a millisecond. Similar results are expected for the other models since the complexity depends only on the local neighborhood.

Our method has the free parameter α that defines the ratio between a valid portal and the circumference of its surrounding cell (see Section 3.4). Table 2 shows the average number of portals tested, cells visible and area rendered, as a function of α. As α increases, the number of visible cells

Model	Method	avg. num visible cells	avg. num portals tested	avg. area rendered
Soda	BW	8	25	106.247
	Merged BSP	12	85	116.739
London	BW	25	83	72.551
	Merged BSP	28	609	107.986
Sava	BW	18	62	956.454
	Merged BSP	23	372	1959.15

Table 1. The average number of visible cells, portals tested and area rendered by the BW and the post-merge BSP partitions.

α	portals	area	cells
10	181	228.082	11
20	91	111.874	16
30	81	76.167	22
40	100	69.480	34
50	145	72.330	54

Table 2. The average number of visible cells, portals tested and area rendered by BW as a function of α, the ratio defined for a valid portal. These results are computed for the London model.

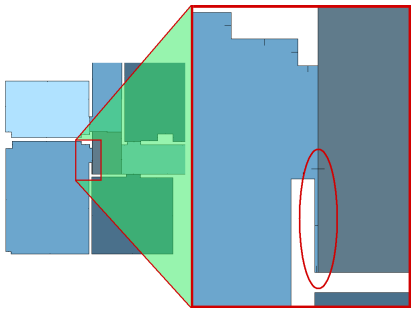


Figure 10. Upon close inspection a seemingly "bad" cell appears.

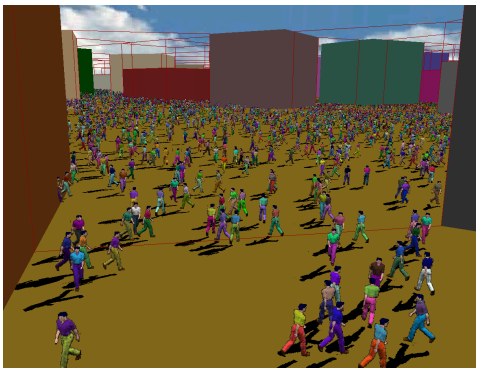
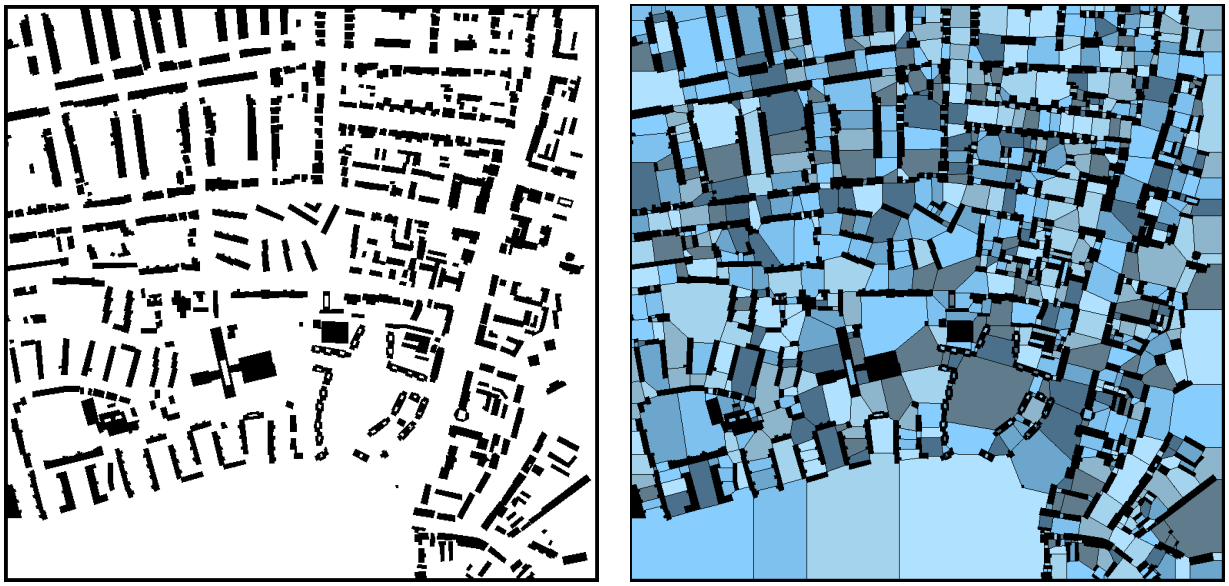


Figure 11. The Sava model walkthrough. The model is populated with 50,000 people.

increases and the average area rendered decreases. However, the number of portals gets a minimum around a ratio of 33%. The effectiveness expression discussed in Section 4 is a function of the average area rendered and the average number of portals tested. Thus, by varying α one can adjust the tradeoff between the rendering of the scene and the portal tests. If the scene is overly loaded with geometry it pays off to test more portals and save on the area. On the other hand, if the graphics card is strong enough, one can save portal tests and render a large area. Thus, our BW partition is adjustable to the complexity of the scene and the graphics capability available.

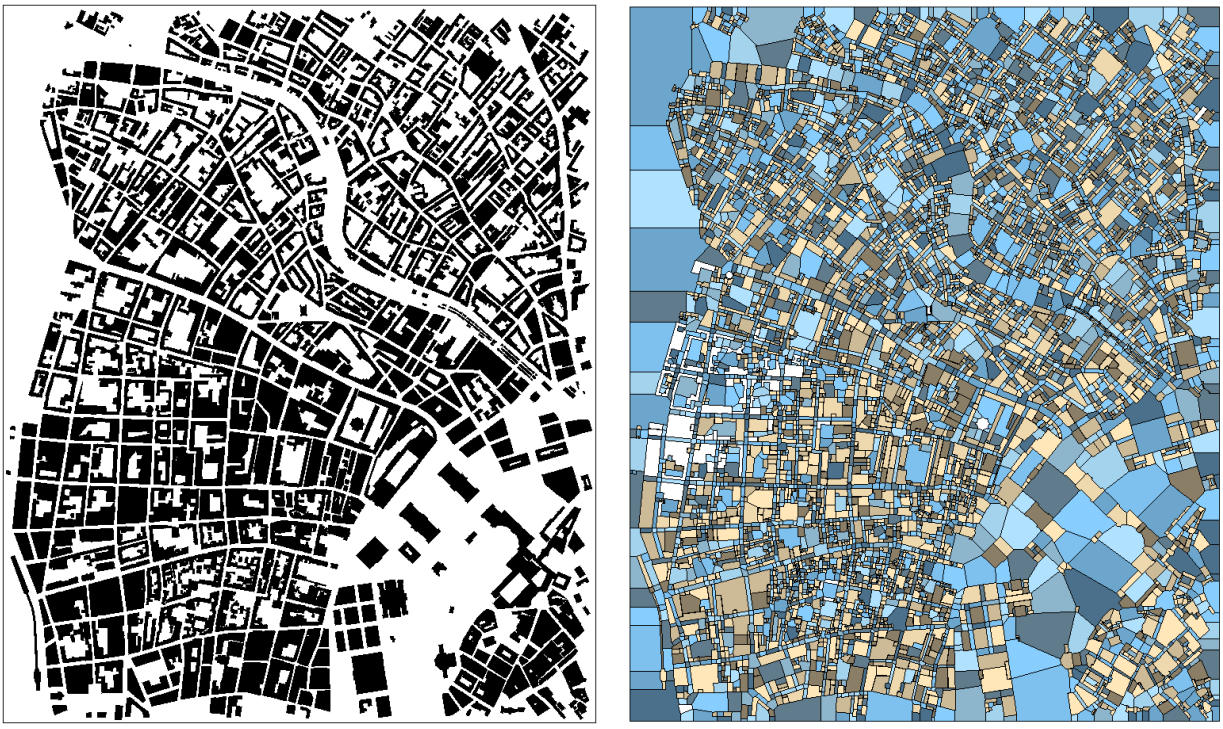
Although the resulting partition is an effective partition and all the cells adhere to the definition of BW optimal cells, some cells may seem less than perfect. In the Soda model, upon close inspection, a single "bad" looking cell appears (see Figure 10). This cell does not seem like a good cell and under no circumstances would anyone create such a cell in a manual partition. This cell is created as a result of misalignments of nearby walls. Nevertheless, the existence of a small number of such cells doesn't cause a significant decrease in the effectiveness of the partition since they do not increase the number of portals.

Note that above we did not report on the average rendering time, but rather on the average area rendered. The area is a good measure of the potential rendering time assuming the geometry load is evenly distributed. The actual time required to render empty cells surrounded by walls is usually too small to show the significance of an output-sensitive algorithm. Thus, in one of our experiments, we have loaded the streets of the Sava model with a large number of humans (see Figure 11). Here, efficient visibility culling is necessary. In our walkthrough system, we use a field-of-view of 45 degree, which means that inherently the output-sensitive PVS is significantly smaller than the one generated by an "all-around" pre-computed from-cell PVS. The associated cost of portal tests is linear in n, the number of visible cells (or portals). This is in contrast to the $\log(n)$ factor that is associated with any hierarchical culling method. Moreover, the portal test itself is simple, inexpensive and realized in software, in contrast to hardware-accelerated visibility culling methods, where the hardware-based visibility tests have to be performed sequentially on the nodes, and thus constantly stalling the graphics pipeline.



(a) (b)

Figure 12. (a) A region of the London model. (b) The result of the BW partition. The average visible area and portals is 72.551 and and 83, respectively.



(a) (b)

Figure 13. (a) The Vienna model. (b) The result of the BW partition. White buildings are buildings above the a 95% H_c threshold. In brown is the roof-top partition.

7 Conclusions

In this paper we presented an algorithm to create an effective cells-and-portals partition. We applied the technique on outdoor scenes using extended portals. This work shows that cells-and-portals is an efficient technique in the sense that is it output sensitive. It avoids the log factor associated with hierarchical visibility techniques and the portal tests do not incur a costly overhead. However, cells-and-portals cannot be applied on arbitrary scenes. It is applicable to scenes which can be subdivided into cells that are defined by simple occluders. As we showed here, for architectural models, whether indoors or outdoors, this is applicable. Extending the technique to general scenes is a worthwhile challenge, since it will benefit from the inherent advantages, mentioned above, of the cells-and-portals technique.

Acknowledgements

This research is supported in part by the EU funded project CREATE IST-2001-34231), and by the Israel Science Foundation founded by the Israel Academy of Sciences and Humanities, and by the Israeli Ministry of Science, and by a grant from the German Israel Foundation (GIF). We would like to thank Anthony Steed for the London Model and FT CL for the crowd rendering. We would also like to thank Peter Wonka and Michael Wimmer for the Vienna 2000 model.

References

[1] J. M. Airey, J. H. Rohlf, and F. P. Brooks, Jr. Towards image realism with interactive update rates in complex virtual building environments. *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, 24(2):41–50, Mar. 1990.

[2] J. Bittner, V. Havran, and P. Slavik. Hierarchical visibility culling with occlusion trees. In *Proceedings of Computer Graphics International '98*, pages 207–219, June 1998.

[3] Y. Chrysanthou. *Shadow Computation for 3D Interaction and Animation*. PhD thesis, Queen Mary and Westfield College, University of London, Feb. 1996.

[4] D. Cohen-Or, Y. Chrysanthou, C. Silva, and F. Durand. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, in press.

[5] S. Coorg and S. Teller. Temporally coherent conservative visibility. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 78–87, 1996.

[6] F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. *Proceedings of SIGGRAPH 2000*, pages 239–248, July 2000.

[7] T. A. Funkhouser, C. H. Séquin, and S. J. Teller. Management of large amounts of data in interactive building walkthroughs. *1992 Symposium on Interactive 3D Graphics*, 25(2):11–20, March 1992.

[8] N. Greene, M. Kass, and G. Miller. Hierarchical z-buffer visibility. *Proceedings of SIGGRAPH 93*, pages 231–240, 1993.

[9] V. Koltun, Y. Chrysanthou, and D. Cohen-Or. Hardware-accelerated from-region visibility using a dual ray space. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, pages 205–216. Eurographics, June 2001.

[10] D. Luebke and C. Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. In P. Hanrahan and J. Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 105–106. ACM SIGGRAPH, Apr. 1995.

[11] D. Meneveaux, K. Bouatouch, E. Maisel, and R. Delmont. A new partioning method for architectural environments. *The Journal of Visualization and Computer Animation*, 9(4):195–213, 1998.

[12] B. F. Naylor. Interactive solid geometry via partioning trees. In *Proc. of the Graphics Interface '92*, pages 11–18, Vancouver, Canada, 1992.

[13] S. Teller. *Visibility Computations in Densely Occluded Environments*. PhD thesis, University of California, Berkeley, 1992.

[14] S. J. Teller and C. H. Sequin. Visibility preprocessing for interactive walkthroughs. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4):61–69, July 1991.

[15] E. Torres. Optimization of the binary space partition algorithm (BSP) for visualization of dynamic scenes. 9(3):507–518, 1990. C.E. Vandoni and D.A. Duce (eds.), Elsevier Science Publishers B.V. North-Holland.

[16] M. van de Panne and A. J. Stewart. Effective compression techniques for precomputed visibility. In *Eurographics Workshop on Rendering*, pages 305–316, June 1999.

[17] H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff III. Visibility culling using hierarchical occlusion maps. In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 77–88. ACM SIGGRAPH, Addison Wesley, Aug. 1997.