A New Partitioning Method for Architectural Environments

Daniel Meneveaux, Kadi Bouatouch, Eric Maisel and Romuald Delmont
IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France

Abstract

Computing global illumination for complex environments in moderate time and walking through them is one of the challenges in computer graphics. To meet this goal, preprocessing is necessary. This preprocessing consists in partitioning the environment into cells and determining visibility between these cells. Most of the existing partitioning methods rely on the Binary Space Partitioning technique (BSP) which can be easily applied to axial environments. But for non axial scenes the BSP has a high complexity of $O(n^3)$ in time to construct a tree of size at worst $O(n^2)$, n beeing the total number of input polygons. Moreover this technique entails a large number of cells that do not necessarily fit with the topology of the environment. We propose in this paper a partitioning method which can be applied to non axial buildings with several floors. It consists of two steps. In the first step, each floor is extracted by applying a BSP technique using the most occlusive horizontal polygons for splitting. In the second step, each floor is in turn partitioned with a model-based method operating in a dual 2D space. The result is a low number of cells fitting at best with the environment topology.

1 Introduction

Even though the throughput of the graphics hardware has improved drastically over the past decade, for complex environments such as buildings containing millions of polygons, real time rendering (for walkthrough) still remains very difficult [1, 2, 3] and global illumination (with the radiosity method) still is a very demanding process in terms of computation and memory resources [4]. To overcome this problem some preprocessing for limiting the number of polygons is needed. For example, when walking through an environment only the polygons (or geometric primitives) visible to the viewpoint are rendered at each frame. As for global illumination with the radiosity method, only a subset of polygons contributes to the illumination of a polygon within the scene.

This preprocessing relies on the subdivision of the scene into cells. A viewpoint or a polygon in a cell C sees only the polygons lying in C as well as those visible through the holes (also termed *portals* by several authors) in its boundary, like windows and doors for bulding interiors.

The subdivision process is often based on a binary space partitioning (BSP). Indeed, the scene is recursively split with the help of planes containing a polygon. The result is a set of 3D cells. The number of cells may be too high and in general these cells do not fit with the topology of the scene. For example, a room may be subdivided into several pieces. Note that the BSP method can be easily applied to axial complex environments (all the scene's polygons are perpendicular to one of the coordinate system axis) [1, 5]. As noticed by Teller [5], for non axial scenes the BSP has an important complexity of $O(n^3)$ in time to construct a tree of size at worst $O(n^2)$, n beeing the total number of input polygons. In [5], Teller et. al. propose a heuristic for partitioning a very simple non axial scene. This heuristic consists in determining the splitting plane supporting the most occlusive polygon. In our opinion, as the splitting planes may be arbitrarily oriented, the resulting cells may have arbitrary shapes: Too thin, too long, too small, etc. Moreover, if the scene contains several walls slightly out of line but having nearly the same orientation, the BSP technique will choose each wall as a splitting polygon, creating then cells having no sense. Rather, it is preferable to associate with these nearly aligned walls only one splitting plane.

The main contribution of our paper is a novel partitioning method which can be applied to any kinds of building (axial, non axial, with several floors, etc.). Compared to the BSP technique, our subdivision method results in a smaller number of cells that fit well with the scene topology.

In this paper previous related works are first presented followed by an overview of our partitioning method and by details on its implementation. Next, results and comparison with Airey's and Teller's BSP method are given to demonstrate the efficiency of our partitioning method.

2 Previous Works

The most important works dealing with visibility calculation in complex environments (buildings) are due to Airey [1] and Teller [5, 6]. Both works rely on a BSP subdivision of the environment into 3D cells. For each polygon P (or cluster O, i.e. object made up of polygons) within a cell C, the polygons visible to P (or to O) are determined and forms the PVS (potentially visible set). The subdivision into cells together with the resulting PVS's allow to walk through a complex building and to make possible the computation of radiosity solutions.

When a BSP scheme is used to partition the scene, the choice of the best splitting plane is important. Note that a splitting plane contains at least one polygon. Airey devised a heuristic function to choose the splitting planes. This function combines three criteria quantified by:

- The *balance* factor of the split: It shows how evenly the plane separates the scene;
- The *occlusion* factor: It provides an information on how well the plane hides the two sides from each other;
- The *split* factor: It shows how little the plane splits the scene polygons.

The heuristic function proposed by Airey is a linear combination of these three factors:

$$partition\_function = 0.5 * occlusion + 0.3 * balance + 0.2 * split.$$

For axial scenes the BSP technique is easy to implement as shown in [1, 5]. However it results in a high number of cells which may not fit with the topology of the scenes. Indeed, a room or a corridor may be split into several pieces and many non-useful empty cells like wall interiors are created. Regarding non axial scenes, the complexity of the BSP technique is far higher as noticed by Teller [5].

For the above reasons, we propose in this paper a new partitioning method that may handle non-axial complex buildings and fits better with the topology of the scene. Our method generates PVS's larger than those that would be obtained with Teller's method. On the other hand, our method reduces the number of cells and drastically simplify visibility calculations. The obtained cells fit well with the topology of the scene, which makes easier walkthrough and Computer Supported Cooperative Works (CSCW).

3 Overview

The objective of our method is to partition all kinds of building, even those for which no a priori information is provided for distinguishing the furnitures from the walls. The scene is modeled with planar convex polygons and not necessarily axial. To partition the scene into 3D cells, our method utilizes only the vertical polygons, say the ones normal to the floor. In a preprocessing step, the set of polygons is divided into two lists: one containing the vertical polygons (like walls) and the other the rest of the polygons. Our partitioning method operates in a 2D space corresponding to the horizontal plane of the WCS (world coordinate system). By convention, the horizontal plane contains the x and y axes and will be denoted Oxy from now on, while z represents height. The vertical polygons are projected onto the Oxy plane. This projection results in a set of segments that are represented by points (θ, ρ) in a dual space, where ρ is the orthogonal distance of the WCS origin to this segment,

and θ the angle formed by this segment and the x axis (see figure 1). The points (θ_i, ρ_i)'s within the dual space corresponding to the projected vertical polygons are used for determining the plane which will help to split the scene into cells. From now on, these particular planes will be called *splitting planes* and their projection onto the horizontal plane *Oxy splitting lines*. Neighbouring points (θ_i, ρ_i)'s are clustered and each cluster is represented by one point R in the dual space. The vertical plane corresponding to R is chosen as a splitting plane. Once all the splitting planes have been found, the cells are determined with the help of construction rules, for example a bedroom is rectangular and its width has a value ranging from $Width_{min}$ meters to $Width_{max}$ meters. These rules guide the construction of the cells by means of the spliting planes.

The motivations of making our method operate in a 2D dual space rather than in a 3D space is related to the fact that:

- Partitioning is made easier in a lower dimension space.
- For a reason of efficiency, it is preferable to avoid associating a splitting plane with each polygon within the scene. Rather, polygons slightly out of line and having nearly the same orientation might be clustered in order to assign to them a single splitting plane (figure 6). This would avoid the extraction of useless cells (like the interior of walls) and would reduce the complexity of the partitioning process.

The next step of our method is the determination of visibility relationships between the resulting cells. To this end, an adjacency graph and a visibility graph are determined. A cell views another cell through portals like windows and doors. We have devised a simple and efficient method to determine these portals. The result of the proposed partitioning method is a reduced number of cells that fit well with the topology of the scene.

However, for complex scenes, some problems can arise:

- Some of the obtained cells could not to fit with the topology of the scene. For example, as certain particular cells may obey at once several construction rules defined by the user, the program splits them into several cells. In this case, it would be judicious to give the user the possibility to merge several cells into one or to interactively select the appropriate splitting planes associated with these particular cells.
- The furnitures within the scene may be partly composed of vertical polygons. These latter are taken into account in the splitting process, implying then the creation of too many cells not fitting with the topology of the scene. Once more, it would be interesting to give the user the opportunity to notify the splitting process to ignore the furnitures.

To bring a solution to these problems, we decided to give the user the possibility of intervening in the partitioning process. To this end, our software makes use of a user interface combining the *Motif* and *OpenGL* libraries. This interface displays two windows. In the first one is displayed a top-view wireframe image of the whole scene (or a zoomed portion) in which the user selects a portion of the scene (with the mouse) destinated to be partitioned into 3D cells. It is possible to walk through the cells resulting from this partitioning in the second window with the help of *OpenGl*. The partitioning process consists in successively selecting portions of the scene and performing their partitioning till the scene be completely partitioned into 3D cells. In addition, our software allows the user to select several cells and merge them into one. Also, the user can, at any time of the partitioning process, ask for the calculation of the visibility graph regarding the cells already extracted.

The next sections give more details on the determination of the splitting planes, the associated cells and of the adjacency and visibility graphs. The following sections deal with the implementation of our method and show some results.

4 Determining the splitting planes

4.1 Dual space

As said before, only the vertical polygons contribute to the partitioning of the scene into 3D cells. Each vertical polygon is projected onto the Oxy horizontal plane. The resulting segment belongs to the line defined by the equation :

$$N \bullet P + \rho = 0,$$

where $P(x, y)$ is a point lying on the line, $N = (N_x, N_y)$ the normal to the polygon ($N_z = 0$) and ρ the orthogonal distance of the WCS origin to the line (or to the polygon). Each segment is represented by a point in the dual space (θ, ρ), where $\theta = \arccos(N_x) * sign(N_y)$, N_x and N_y being the x and y coordinates of the normal N (see figure 1).

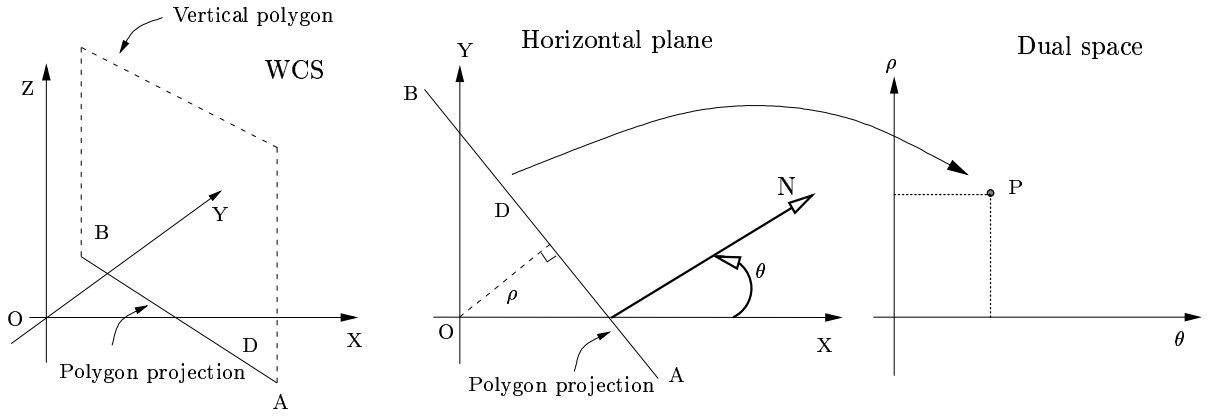

Figure 1: Dual space

The data structure associated with a vertical polygon is shown in figure 2.

```
typedef struct POLYGON {
   Color *color;
   int nb_vertices;                /* number of vertices */
   Point points[MAXPOINTS];        /* List of vertices */
   Vector normal;
   double area;
   double theta;                   /* orientation angle */
   double rho;                     /* distance from origin */
   double abscissa[2];             /* Coordinates in the 1D LCS of the */
                                   /* projected vertical polygon */
} *Polygon;
```

Figure 2: Data structure for a vertical polygon

This data structure contains the coordinates (θ, ρ) in the dual space of the point corresponding to the segment resulting from the projection of the polygon onto the Oxy horizontal plane. To this point is assigned the area of this polygon. Note that several vertical polygons may project onto the same point in the dual space, this means that these polygons are coplanar and similarly oriented. This particular case is handled by the clustering process described later on.

A 1D local coordinate system (called LCS from now on) is associated with the line supporting this segment and has an origin O_L which is the orthogonal projection of the WCS origin O onto this line as shown in figure 3. The direction vector of this coordinate system is defined by the angle $\theta = \theta_N - \pi/2$,

where θ_N is the angle formed by the normal to the polygon and the x axis. The coordinates of the two segment endpoints are given by the fields *abscissa* as pointed out in the above data structure.

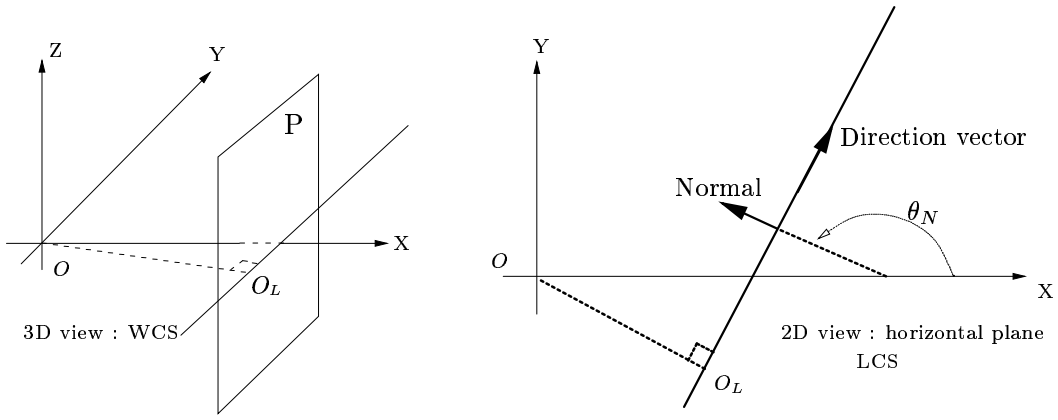

Figure 3: 1D local coordinate system (LCS)

4.2 Clustering

Once all the vertical polygons have been projected onto the Oxy horizontal plane, their projections (segments) are transformed into points in the dual space (θ, ρ). Within this space, two points close to each other correspond to vertical planes which almost have the same orientation (nearly the same normal). The aim now is to cluster neighboring points. The result is a certain number of clusters (see figure 4). From each cluster we determine a point (θ_s, ρ_s) which corresponds to the vertical splitting plane P_s associated with this cluster. Note that a splitting plane do not contain necessarily a polygon of the scene. Once clustering has been completed, the resulting splitting planes are used to partition the scene into cells.

Let us see now how the clustering is performed. First of all, the vertical polygons are sorted according to their θ value in an increasing order and saved in a list L_θ. Then, we search for the greatest distance between the θ values associated with two successive polygons in L_θ:

$$\max_{i \in [1,N]} (\theta_{i+1} - \theta_i),$$

N being the number of elements in L_θ. Let I be the index such that $(\theta_{I+1} - \theta_I) = \max_{i \in [1,N]}(\theta_{i+1} - \theta_i)$. We choose a value θ_p as the middle of the interval $[\theta_I, \theta_{I+1}]$, i.e. $\theta_p = \theta_I + 0.5 * (\theta_{I+1} - \theta_I)$, to split the list L_θ into two other lists L^1_θ and L^2_θ. The polygons in L^1_θ have a θ value smaller than θ_p while those of L^2_θ have a θ value greater than θ_p. This splitting process is recursively repeated for L^1_θ and L^2_θ. The recursion stops when the maximum distance $max(\theta_{i+1} - \theta_i)$ is below a certain threshold. This recursion results in a binary tree B_θ for which each leaf is a list of vertical polygons whose $\theta's$ are close to each other. Each leaf of B_θ is now sorted according to the ρ value and recursively split into two parts according to the method used for splitting L_θ. At each step of the splitting process, the splitting value ρ_p is equal to $\rho_I + 0.5 * \max_i(\rho_{I+1} - \rho_I)$, i being the index such that $(\rho_{I+1} - \rho_I) = \max_i(\rho_{i+1} - \rho_i)$ The recursion stops when $max(\rho_{i+1} - \rho_i)$ is below a certain threshold. At this step, each leaf i of B_θ is a root node of a binary tree B^i_ρ resulting from the splitting according to the ρ values. Finally, our clustering method results in an extended binary tree B whose leaves are clusters.

4.3 Splitting planes

For each cluster we determine a splitting plane whose equation is $N_s \bullet P + \rho_s = 0$. The associated θ_s value is chosen as close as possible to those of the polygons in the cluster while favouring the polygon of greatest area. More precisely, θ_s is equal to the mean of all the θ_i's weighted by the surface areas $area_i$ of the polygons within the cluster:

$$\theta_s = \frac{\sum_{i \in cluster} area_i * \theta_i}{\sum_{i \in cluster} area_i}.$$

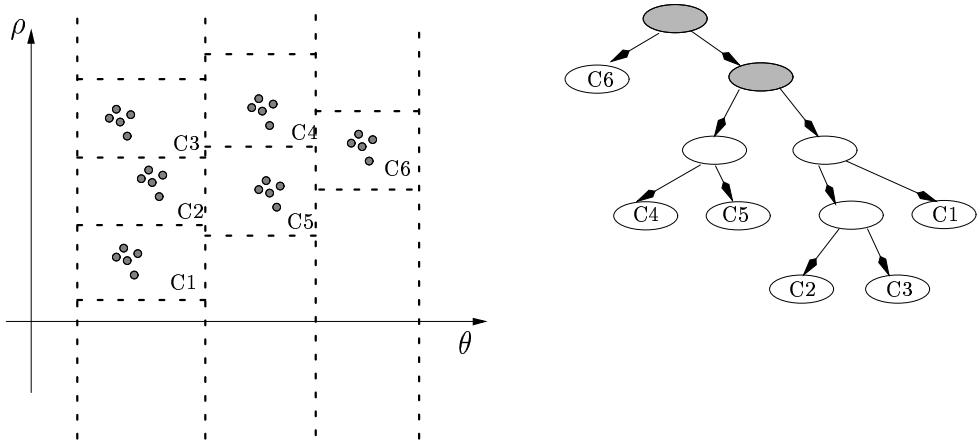

Figure 4: Clustering : Recursive subdivision resulting in a binary tree whose leaves C_i's are clusters.

Let us see now how ρ_s is determined. We choose one splitting plane so that the half-space defined by $N_s \bullet P + \rho_s > 0$ contains all the polygons belonging to the cluster. This condition can be expressed as $\rho_s > \max_i(-N_s \bullet P_i)$, where the $P_i's$ are the vertices of all the polygons within the cluster. Finally, we choose $\rho_s = \max_i(-N_s \bullet P_i) + \epsilon$, ϵ being a small value chosen by the user so that none of the vertical polygons lie on the splitting plane (5cm in our implementation). Figure 5 gives the data structure defining a splitting plane.

```
typedef struct SPLITTING_PLANE {
   double theta_s;
   double rho_s;              /* Distance to origin */
   double area;               /* Sum of the areas of the polygons in
                                    the associated cluster*/
   vector Normal_s;
   double origin[2];          /* Origin of the LCS */
   double abscissa[2];        /* minimum and maximum coordinates
                                    of the polygons in the LCS*/
} *Splitting_plane;
```

Figure 5: Data structure for a splitting plane

LCS is the 1D local coordinate system associated with the splitting plane. The segments resulting from the projection of the vertical planes (within the same cluster represented by the splitting plane) onto the horizontal plane Oxy are projected in turn onto the LCS axis. Abscissa are the coordinates (called Min and Max) of the endpoints of the smallest segment (lying on this axis) containing all these projected segments (see figure 6). From now on, this segment will be called MinMax segment. So, a MinMax segment is associated with each splitting plane.

5 Cell construction

Once all the splitting planes have been determined, the partitioning of the scene into cells is performed. Note that if the value of the *area* field in the data structure associated with the splitting planes (see

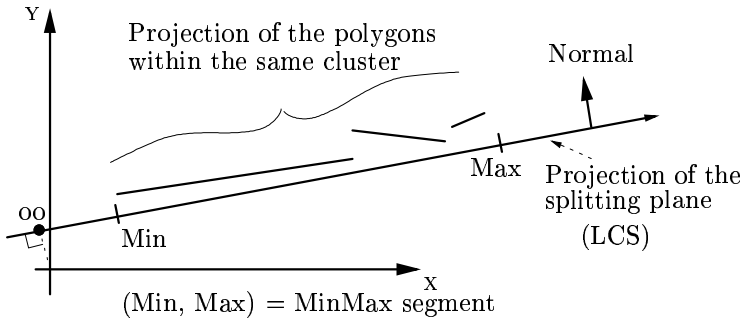

Figure 6: MinMax segment

figure 5) is smaller than a certain threshold (fixed by the user) then this plane and consequently the associated furnitures are not accounted for in the partitioning process.

As with each splitting plane is associated a half-space defined by $N_s \bullet P + \rho_s > 0$, a convex cell is no more than the intersection of such half-spaces. The normal to each splitting plane points toward the interior of the associated cell. Since our objective is to partition the scene into a reduced number of cells fitting with the scene topology at best as possible, the partitioning process is guided by some a priori architectural knowledges. For example, most of the rooms are rectangular (axial or non-axial) and have a width larger than $Size_{min}$ meters and smaller than $Size_{max}$ meters. Corridors may be delimited by two parallel walls, etc.

Before showing how these rules are used, let us give some definitions. A splitting plane $P1$, defined by (θ_1, ρ_1) in the dual space, faces another splitting plane $P2$, defined by (θ_2, ρ_2), if $\theta_1 = \theta_2 + \pi$ and $\rho_1 + \rho_2 > 0$, and if their MinMax segments overlap (see figure 7). Two MinMax segments overlap if the orthogonal projection of one projects partially or completely on the other.

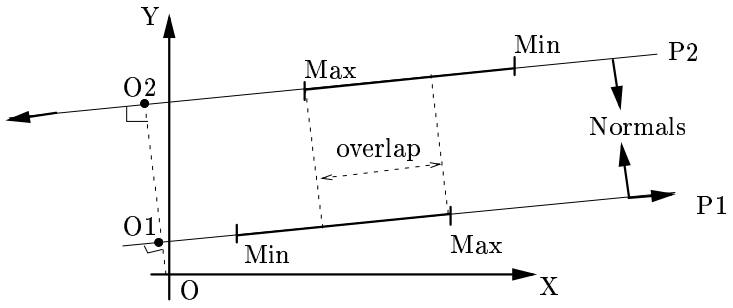

Figure 7: Two splitting planes facing each other. $\rho_1 < 0$, $\rho_2 > 0$.

5.1 Applying the construction rules

The used rules help to determine several kinds of cells whose list is not exhaustive but allow the extraction of all the cells.

5.1.1 Rectangular cell

In a building, most of the rooms, offices and corridors are rectangular and have a width and a length whose values range over $[Size_{min}, Size_{max}]$. By exploiting this fact we can extract 3D rectangular cells corresponding to rooms with 4 walls as explained in figure 8.

5.1.2 Cell with 2 parallel walls

Now the objective is to extract cells defined by at least two parallel splitting planes (2 parallel walls) facing each other as shown in figure 9. Figure 10 gives the algorithm allowing the extraction of these kinds of cell.

1- Choose the most occlusive splitting plane P_0, i.e. whose area field defined in its associated data structure is maximum
2- Search the binary tree B_θ (the upper part of B) to find the set E of splitting planes facing P_0. E is a leaf i of B_θ and corresponds the root of a binary tree B^i_ρ.
3- Let P_1 be the element of E closest to P_0 so that $Size_{min} \leq \rho_0 + \rho_1 \leq Size_{max}$, and the $MinMax$ segments of P_1 and P_0 overlap. P_1 is determined by searching B^i_ρ.
4- Choose a splitting plane P_2 perpendicular to P_0, i.e. so that $\theta_2 = \theta_0 + \pi/2$ by searching B_θ.
5- Search the binary tree B_θ (the upper part of B) to find the set E' of splitting planes facing P_2. E' is a leaf j of B_θ and corresponds the root of a binary tree B^j_ρ.
6- Let P_3 be the element of E' closest to P_2 so that $Size_{min} \leq \rho_2 + \rho_3 \leq Size_{max}$, and the $MinMax$ segments of P_2 and P_3 overlap. P_3 is determined by searching B^j_ρ.

Figure 8: Rectangular cell with 4 walls

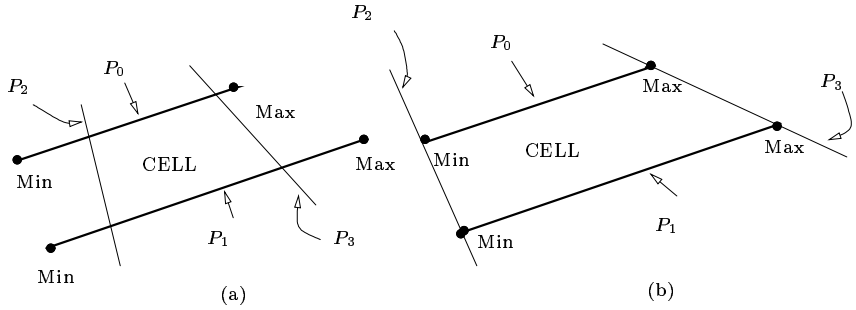

Figure 9: Cell with two parallel walls

5.1.3 Other convex cells

Once all the rectangular cells as well as those defined by at least two parallel splitting planes have been found, we determine the remaining convex cells. As our objective is to partition the rest of the scene into a gap-less tiling of abutting cells, we propose a method which extracts these cells by following their contours as explained hereafter. Let us consider the example given in figure 11 to explain how these convex cells are extracted. This figure shows the orthogonal projection of the splitting planes onto the horizontal plane. First of all, we choose the most occlusive splitting plane P_0 with a Minmax segment having Min and Max as endpoints. Then we look for the plane P_1 which is closest to the point Min. Next, we repeat the same process by searching for the plane $P2$ which is closest to the point Min of the $P1$'s MinMax segment. In the same manner we find the closest plane P_3 to the point Min of the $P2$'s MinMax segment. As P_3 intersects the initial splitting plane P_0 at a point lying on the P_0's MinMax segment, the cell searching process ends up and the resulting cell (shaded in the figure) is delimited by the splitting planes : P_0, P_1, P_2, P_3. This partitioning results in convex cells, some of them may not fit with the topology of the scene, for example a concave room may be divided into many convex cells. As will be seen later on, the user can interactively decide to merge several cells into one.

1- Choose 2 parallel splitting planes P_0 and P_1 facing each other whose MninMax segments M_0 and M_1 overlap
2- Find 2 other splitting planes P_2 and P_3 whose splitting line intersect M_0 and M_1
3- If these planes exist then construct the cell (figure 9-a)
4- Else create them as planes passing through the Min and Max endpoints of M_0 and M_1 (figure 9-b)

Figure 10: Cell with 2 parallel walls

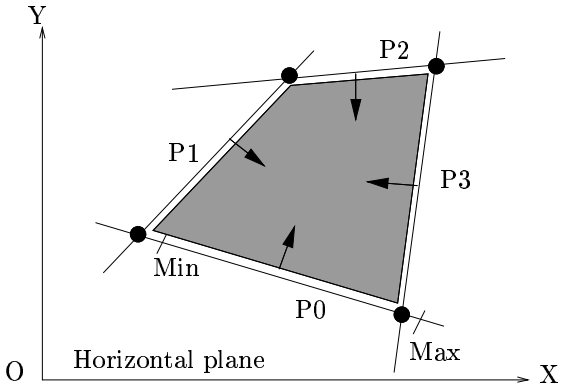

Figure 11: Other convex cells

5.2 Guiding the search for cells

For a reason of efficiency it is worth to guide the cell search process. To this end, we have devised and implemented a method which has been presently applied only for rectangular cells. This method operates as follows (see figure 12). Suppose a rectangular cell C_1 has been determined, and let P_1, P_2, P_3, P_4 be the associated splitting planes. First, we choose one of these planes lying along the longest side of C_1, for example P_4. To extract a new cell C_2 we keep P_1 and P_2 and consider P_4 as the new third plane. P_4 becomes P_3 for C_2. Now, we only have to determine the new plane P_4 facing P_3. This process is repeated till all the cells are extrated : C_3, C_4, etc.

5.3 Merging splitting planes

As our objective is to extract abutting cells, we try to place the splitting planes into the centers of the walls as explained hereafter. Figure 13 shows two adjacent cells C_1 and C_2. They share a common

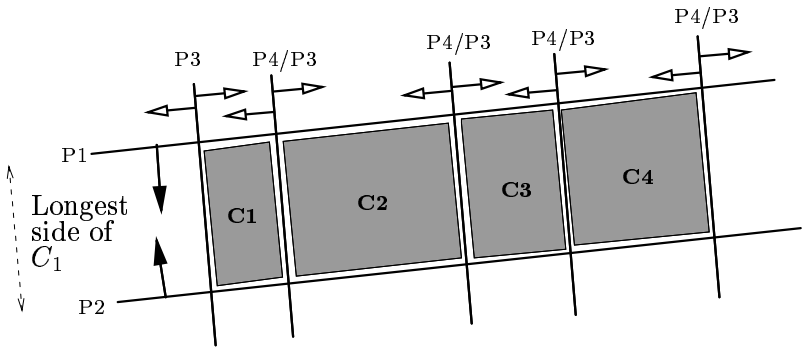

Figure 12: Guided cell search

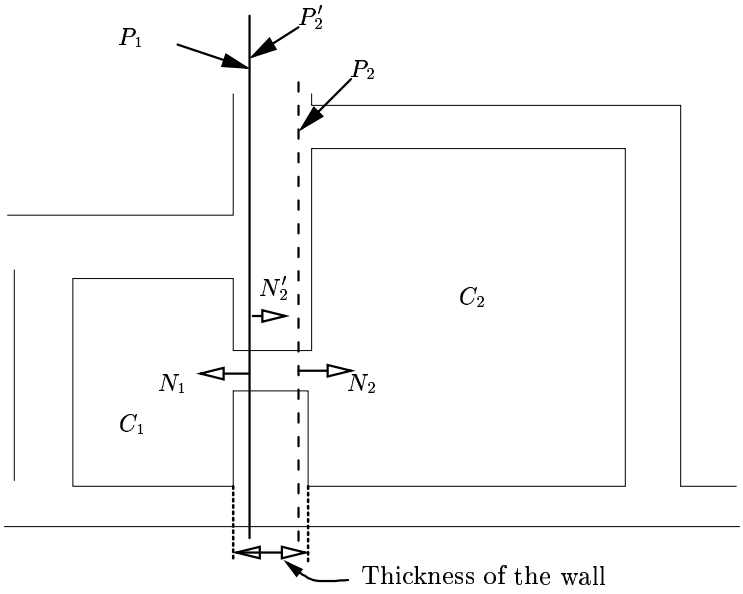

Figure 13: Splitting planes of adjacent cells.

wall which has a certain thickness (for all our geometric models of architectural environments, the walls do have a thickness). One of the splitting planes delimiting C_1 (plane P_1 of normal N_1) passes through this common wall and so does one of the splitting planes forming C_2 (plane P_2 of normal N_2). These two planes have opposite normal ($N_1 = -N_2$) but their associated ρ's are different ($\rho_1 + \rho_2 < 0$). Unlike the BSP-based partitioning method, our approach avoids unnecessary cells (like the interior of a wall) by merging the planes P_1 and P_2 so that P_1 remains unchanged but P_2 is replaced by the plane P_2' whose normal and ρ are $N_2' = -N_1$ and $\rho_2' = -\rho_1$. In this way, the cells do not overlap. Whenever a cell C is determined, the program tries to merge its associated splitting planes with those defining the cells extracted before C.

6 Visibility graph

The visibility graph is determined from the adjacency graph. In this latter a vertex is a cell and an edge between two vertices means that there exists at least one portal between the adjacent cells corresponding to these vertices. Let us see now how these portals are determined.

6.1 Determining the portals

A cell views another cell through portals such as windows and doors. The problem of determining such portals has already been addressed by Airey [1]. Airey extends and generalizes the plane-sweep triangulation algorithm to triangulate a region of the plane defined by set operations such as: union, intersection and difference. As pointed out by Airey, this approach is quite complicated and time-consuming.

For this reason, we have devised a simple and efficient method for determining the portals as explained hereafter. Our method is capable of computing even non-convex portals.

Recall that a cell C is delimited by splitting planes whose normals point toward the interior of C. Let P_s be a splitting plane. It partitions the 3D space into two half-spaces: V_+ and V_-. Let V_+ be the half-space containing the cell. If a scene's polygon P is such that some of its vertices are in V_+ and the others in V_- then it is clipped by P_s. The intersection of P by P_s results in a segment called *portal segment* from now on. A portal is a polygon whose edges are portal segments. A polygon P clipped by a splitting plane P_s is divided into two other polygons: P_{in} (inside the cell) and P_{out} (outside the cell). let S be the resulting portal segment shared by P_{in} and P_{out}. Recall that the vertices of each object polygon are ordered so that its normal points toward the outside of the object containing it.

If we visit the vertices of P_{in} in the sense defined by the orientation of P, S becomes automatically oriented (see figure 15 and 16). In this way all the portal segments become oriented, which facilitates the construction of the portals as shown in figure 14. For each splitting plane of a cell, the algorithm given in figure 14 is invoked. Note that for each cell we determine its own portals, independently of the other cells. Thus, two adjacent cells contain at least one same portal. Consequently each portal is determined twice.

```
typedef struct VECTOR {
    point : A        /* starting point */
    point : B        /* end point */
} portal_segment          /* vector AB */
C : List_of_portal_segments = empty;        /* delimiting a portal */;
L : List_of_portal_segments;
S_0, S_1, S_first : portal_segment;

L = Determine_all_the_portal_segments();
while L ≠ empty do
{
    Take a portal segment S_0;
    S_first = S_0;
    L = L − {S_0};
    C = C + {S_0};
    Portal_found = false;
    while not Portal_found do {
        Find the segment S_1 so that S_1.A = S_0.B;
        L = L − {S_1};
        C = C + {S_1};
        S_0 = S_1;
        if S_0.B = S_first.A then Portal_found = true;
    }
}
```

Figure 14: Determining the portals

It may happen that some furnitures that penetrate a cell wall will also produce oriented segments and consequently bad portals. Presently, our algorithm handles bad and real portals similarly. Fortunately, as these bad portals are included in real portals (like windows and doors), the correctness of the visibility calculations is not affected. Note that it is easy to remove these bad portals by means of a simple inclusion test.

6.2 Visibility graph

Whenever a cell C has been extracted, we determine its portals and compare them to those of the other cells already extracted. The cells having at leat one same portal are adjacent to C. In this way, an adjacency graph G_a is progressively constructed. In this graph, a vertex is a cell while the existence of an arc between two vertices means that the associated cells are adjacent and visible from each other through at least one common portal.

As for the visibility graph G_v, it provides information on visibility between two cells not necessarly adjacent. A vertex of G_v corresponds to a cell. An edge (of G_v) links two cells C_i and C_j if there exists at least one point I lying on a portal of C_i and a point J on a portal of C_j such that I views J through

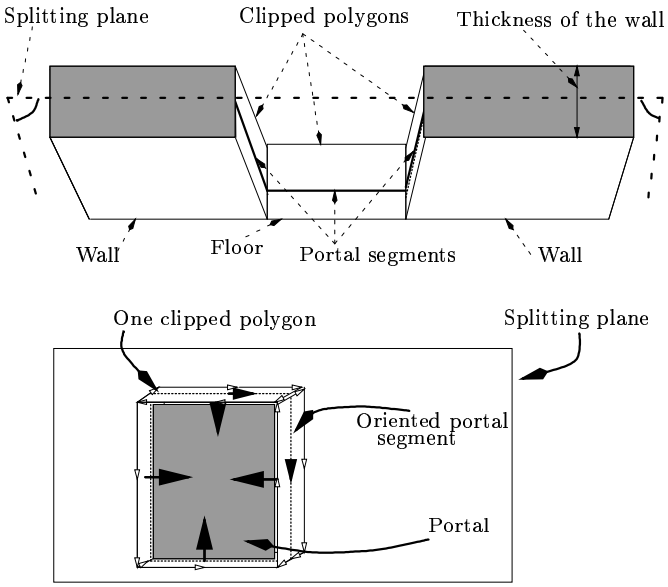

Figure 15: Portal in 3D

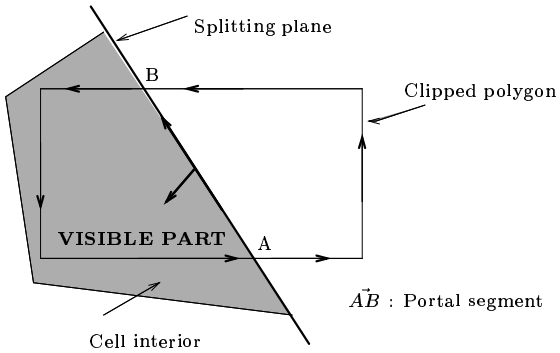

Figure 16: Portal segments

a sequence of portals $O_1, O_2, ...O_n$ located between C_i and C_j. For example, let C_1, C_2, C_3, C_4 be four cells such that (C_1, C_2), (C_2, C_3), (C_3, C_4) are successive edges of the adjacency graph. Consequently, C_1 and C_2 share at least one common portal (O_{12}), and so do C_2 and C_3 (O_{23}), C_3 and C_4 (O_{34}). Visibility between C_1 and C_4 is computed as follows (see [1] for further details). Points are randomly sampled on the portals O_{12} and O_{34}. Then rays are traced between sample points belonging to these two portals. As soon as a ray passes through all the intermediate portals (O_{23} in this case), C_1 and C_4 are said visible. If none of the N traced rays (N fixed by the user) passes through all the intermediate portals, C_4 is not visible to C_1. This process is repeated for the remaining cells by recursively traversing the adjacency graph.

7 Discussion

Presently, our partitioning method handles three kinds of construction rules allowing the extraction of rectangular cells, cells with two parallel walls and any convex cell. Once a cell has been extracted, all the scene's polygons within this cell (such as furnitures, floors, ceilings, roofs, etc.) are determined. This is simply done by checking if such polygons are in the half-spaces pointed by the normals to the splitting planes delimiting the cell.

In case of buildings with several floors, we extract each floor (as well as the portals between the floors) with the help of a BSP technique using only the horizontal polygons for splitting. Next, our partitioning method is applied to each floor separately.

Note that as we have no a priori knowledge about the different kinds of objects making up the scene (i.e. walls, furnitures, ceilings, etc.), the polygons modeling the furnitures are likely accounted for in the partitioning process, which entails too many cells not fitting with the topology of the scene. For example, if a classroom is furnished with aligned chairs and tables, this aligning implies the creation of many splitting planes and consequently several cells are generated. In the next section we will see how the user can interactively remove the furnitures from the list of polygons involved in the partitioning process with the help of the user interface we have developped.

8 Interactive partitioning

Without the intervention of the user in the partitioning process, many problems can arise as mentioned in the previous sections. Let us now recall these problems. It is preferable to ignore in the partitioning process the vertical polygons making up the furnitures. This would avoid the creation of too many cells not fitting with the topology of the scene. For example in a classroom the chairs and the tables must not participate in the partitioning process. In addition, if some of the cells are particular (like concave rooms for example) they cannot be extracted as they are but are partitioned into several cells because they need a specific construction rule. One solution to these problems is to let the user intervene in the partitioning process. This is why our software comprises a user interface giving the user the possibility to intervene in the partitioning process. This user interface makes use of the Motif and OpenGl libraries. When running this software, two windows are displayed on the screen: *working* window and *walkthrough* window.

8.1 *Working* window

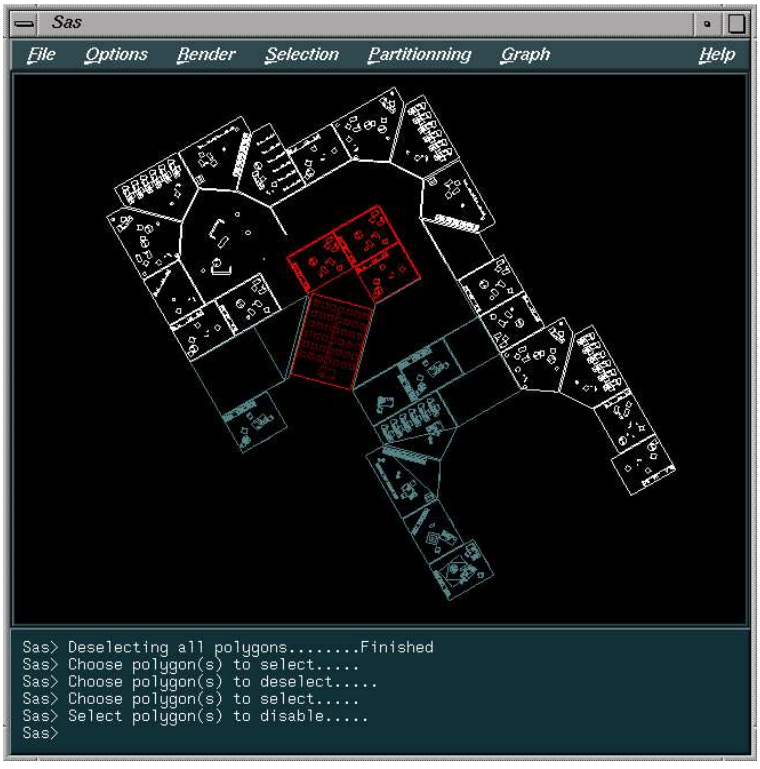

Figure 17: Working window

In this window are displayed, in a wireframe form, only the vertical polygons of the scene after perspective or orthographic projection (see figure 17). With the help of some keys and the buttons' mouse it is possible to move above the scene, to zoom in and out, to select one polygon or a group

of polygons, or several groups. The cells to be merged into one may also be selected and displayed in purple color. The selected polygons are in red color while the others are white. The partitioning is applied only to these selected polygons. Among the selected polygons within a group, some of them may be marked by the user as polygons not participating in the partitioning process. These particular polygons are displayed in dark red color. As soon as a cell has been determined, its polygons are displayed in blue color.

Let us see now the role of the menus offered by our user-interface (figure 17).

- **Menu options**
  This menu allows the modification of all the optional parameters used by the different steps of the partitioning process (height of the ceiling, minimum and maximum width of the rooms, corridors, parameters for clustering in the dual space, etc.).

- **Menu partitioning**
  With this menu, the user selects the construction rule to be applied to the selected polygons then starts the partitioning process. Each determined cell is shown to the user who decides wether to accept or to reject it. Next, a new construction rule may be selected by the user till the selected portion of the scene be completely partitioned.

- **Menu selection**
  Different selection modes are offered by this menu: selection or deselection of a polygon or a group of polygons or a cell.

- **Menu graph**
  When using this menu, the user can ask for the creation of the visibility graph from the adjacency graph. This visibility graph is used to walk through the already determined cells displayed in the *walkthrough* window.

- **Menu file**
  With the help of this menu, the scene to be partitioned can be loaded, the determined cells and the visibility graph can be written on disk.

8.2 *Walkthrough* window

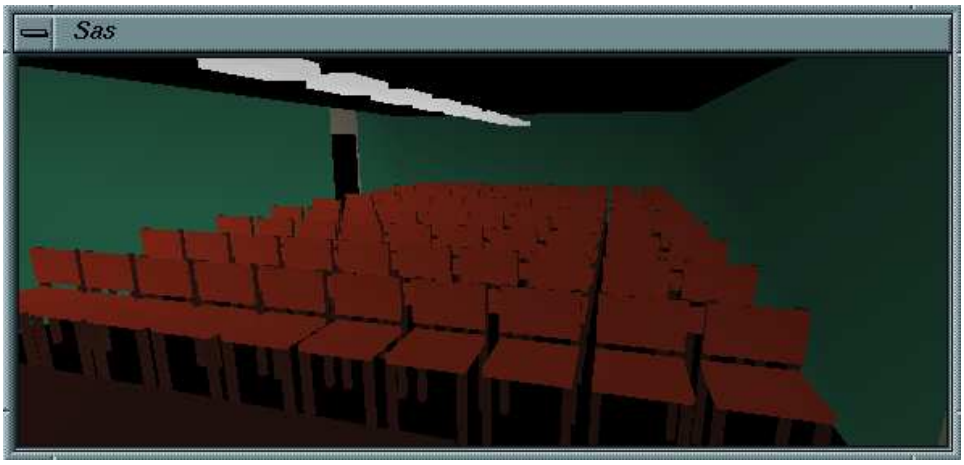

Figure 18: Walkthrough window

In this window, as soon as cells are extracted they are rendered by using the OpenGl library. Different kinds of rendering can be selected (*render* menu of the *working* window): wireframe, flat or Gouraud shading. This window is shown by figure 18 and contains the classroom also displayed at the center of figure 17. In this window, walkthrough can be performed by means of the mouses' buttons.

8.3 Input/Ouput file formats

The input file contains the NFF description [7] of the scene to be partitioned. The partitioning results in several files: One NFF file for each cell and one file storing the visibility graph. The obtained files are used by our radiosity (adapted to complex environments) and interactive walkthrough programs.

9 Results

This section shows some results obtained with our partitioning method and with the one proposed by Airey and Teller which is based on a binary space partitioning. Two scenes have been considered. The first one represents the first floor of *Soda hall* modeled by Teller and the second, named *Castle*, has been built with our modeler. Note that *Castle* is a non-axial scene.

This *Soda hall* floor is an axial scene composed of 2003 rectangular polygons and does not contain any furnitures. It is made up of 150 rooms and corridors. All the most occlusive polygons (walls) are axial, say perpendicular to one of the coordinate system axes. This scene has been partitioned with our method and with the BSP technique (that we have implemented) proposed by Airey and Teller in which the splitting planes are determined by a heuristic function combining three criteria (see section 2) and the portals (rectangular in this case) have been determined with set operations.

Table 1 gives some results on the *Soda hall* scene obtained with both methods.

	BSP	Our method
Partitioning time	5 s	141 s
Number of cells	1528	155
Portal computing time	12 s	< 1s
Visibility graph computing time	310 s	< 1s

Table 1: Comparison with the BSP technique

This table shows that the number of cells obtained with the BSP technique is equal to ten times the one obtained with our method. Indeed, some cells created by the BSP technique, like those located between two adjacent rooms and due to the thickness of the walls, are useless. This kind of cell cannot be extracted by our partitioning method. The partitioning time is more important with our method but the time for computing the portals is far lower. Figure 19 represents a top-view of the partitioning result obtained with our method, a unique color is associated with each cell.

Castle is a furnished scene composed of 28192 input polygons and contains 38 rooms. Table 2 gives some results obtained with our method.

A wireframe top-view of the resulting cells is given by figure 21, in which the portals are filled polygons with yellow color.

In figure 22 the furnitures as well as the other polygons are displayed in a wire-frame form and each cell has a unique color.

We can remark that the resulting cells fit with the topology of the scene which explains the low number of cells. Like for the *Soda hall* scene, time for computing the portals is insignificant.

Recall that with each vertical scene's polygon is associated a point in the 2D dual space. This point is assigned the area of this polygon. Once clustering has been performed, each cluster is in its turn assigned an area value equal to the sum of areas associated with its points. These area values are shown in figure 20 for scene *Castle* and correspond to height values. The more a group of polygons is occlusive the larger is the area value of its associated cluster. In this figure there broadly exist two kinds of cluster: clusters with a low height and clusters with a high height. The low height clusters are randomly distributed. They correspond to the least occlusive groups of polygons such as furnitures. The other clusters are regularly distributed, they represent the walls. Note that the existence of two kinds of clusters makes easy the search for splitting planes.

Number of cells	38
Number of portal segments	200
Number of portals	42
Portal computing time	< 1s
Number of clusters	4376
Number of used clusters: Equal to the number of splitting planes	154
Number of polygons making up the furnitures	10368
Visibility graph computing time	< 1s

Table 2: Results for the *Castle* scene

10 Conclusion

A new algorithm for partitioning architectural environments (building interiors) has been described in this paper. Unlike the methods proposed in the literature, the algorithm does not make use of binary space partitioning (except for separating the different floors of the building) but uses a dual space that represents the most occlusive polygonal objects. The cells are determined according to construction rules defined by cell models. This model-based approach avoids the creation of cells which do not fit with the topology of the scene, which is very useful for walkthrough and Computer Supported Cooperative Works. In addition, it is easier and more intuitive to rely on geometric rules rather than tuning parameters (such as the ones described in section 2) empirically. Furthermore, the addition of new construction rules and consequently new cell models entails a very slight modification of the algorithm. Another advantage of the algorithm is its capability of similarly handling axial and non-axial scenes. As for the determination of the portals, this algorithm makes use of a simple and efficient method. We ascertained that for avoiding the problems cited in section 8, the user must have the possibility to intervene in the partitioning process through a user interface. This possibility is offered by our software.

We are already using this algorithm for walkthrough and radiosity calculation in complex architectural environments.

One of the perspectives of this work is to devise a method which would refine the resultimg partioning so as to reduce the size of the PVS's. Another perspective is the extension of the proposed partitioning method to outdoor scenes like urban environments and to digital elevation maps (DEM). For example, in the case of urban environments the streets and squares are cells and the façades are the cell boundaries. Another direction to be invistigated consists in finding out a dual space so as to handle oblique occlusive polygons.

References

[1] John M. Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision And Potentially Visible Set Calculations.* PhD thesis, University of North Carolina at Chapel Hill, 1990.

[2] John M. Airey, John H. Rohlf, and Frederick P. Brooks. Towards image realism with interactive update rates in complex virtual building environments. *ACM Siggraph*, pages 41–50, May 1990.

[3] T. Funkhouser, S. Teller, C. Séquin, and D. Khorramabadi. The uc berkeley system for interactive visulalization of large architectural models. *Presence*, 5(1):13–44, 1996.

[4] Seth Teller, Celeste Fowler, Thomas Funkhouser, and Pat Hanrahan. Partitioning and ordering large radiosity computations. In *Computer Graphics Proceedings, Annual Conference Series*, pages 443–450, 1994.

[5] Seth Jared Teller. *Visibility Computations in Density Occluded Polyhedral Environments.* PhD thesis, University of California at Berkeley, 1992.

[6] Seth Teller and Pat Hanrahan. Global visibility algorithms for illumination computations. In *Computer Graphics Proceedings, Annual Conference Series*, pages 239–246, 1993.

[7] E. Haines. A proposal for standard graphics environments. *IEEE Computer Graphics and Applications*, 1987.

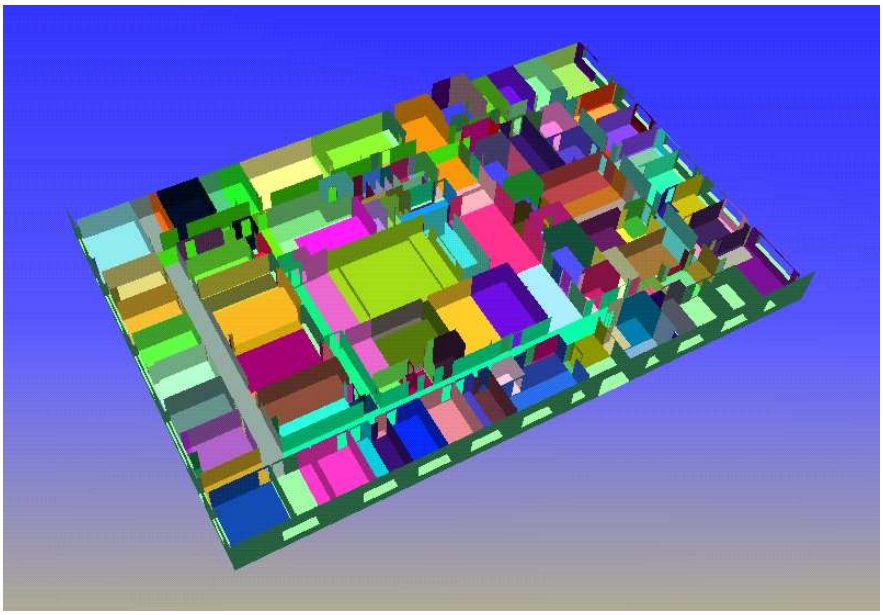

Figure 19: top-view of *Soda hall* with our method.

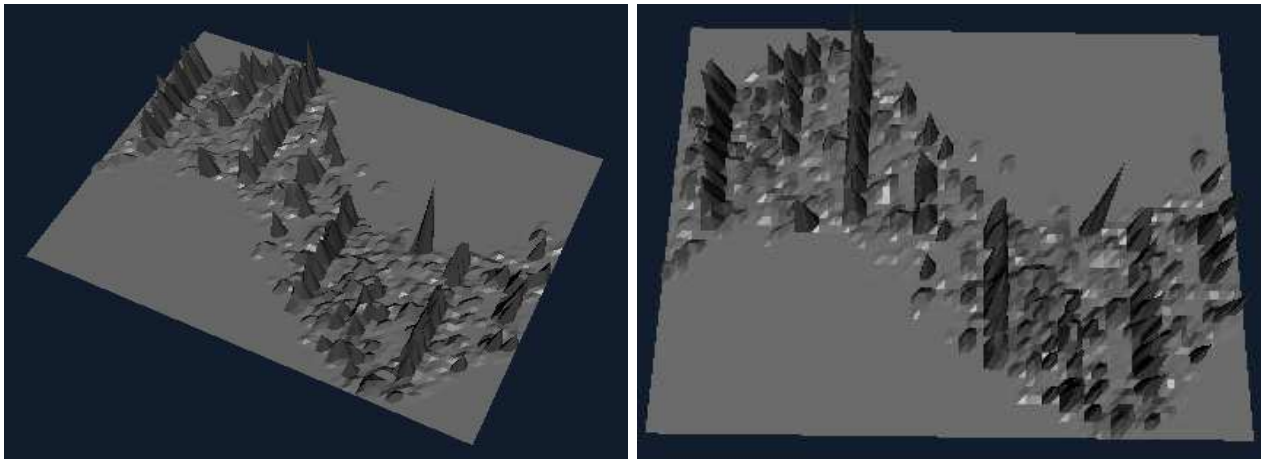

Figure 20: 3D representations of the dual space: The height of a point is the area of the associated cluster.

Figure 21: Top-view of the *Castle* scene: The portals are the yellow filled polygons.

Figure 22: A top-view wire-frame image of *Castle*: Each cell has a unique color.