EUROGRAPHICS 2003 / P. Brunet and D. Fellner
(Guest Editors)

Volume 22 (2003), Number 3

Volumetric cell-and-portal generation

D. Haumont[1], O. Debeir[1] and F. Sillion[2]

[1]SLN, Université Libre de Bruxelles [2]Artis, INRIA[†]

Abstract

We present an algorithm to generate a cell-and-portal decomposition of general indoor scenes. The method is an adaptation of the 3D watershed transform, computed on a distance-to-geometry sampled field. The watershed is processed using a flooding analogy in the distance field space. Flooding originates from local minima, each minimum producing a region. Portals are built as needed to avoid the merging of regions during their growth. As a result, the cell-and-portal decomposition is closely linked to the structure of the models. In a building, the algorithm finds all the rooms, doors and windows. To restrict the memory load, a hierarchical implementation of the algorithm is presented. We also explain how to handle possible model degeneracies -such as cracks, holes and interpenetrating geometries- using a pre-voxelisation step. The hierarchical algorithm, preceded when necessary by the pre-voxelisation, was tested on a large range of models. We show that it is able to deal with classical architectural models, as well as cave-like environments and large mixed indoor/outdoor scenes. Thanks to the intermediate distance field representation, the algorithm can be used regardless of the way the model is represented: it deals with parametric curves, implicit surfaces, volumetric data and polygon soups in a unified way.

1. Introduction

The visibility determination problem (i.e. the process of finding the geometry that is visible from the viewer location) is central in many computer graphics algorithms. Visibility has been studied for a long time and many solutions have been proposed in the literature. Most visibility culling algorithms make use of particular scene characteristics to speed up the determination process. An important class of models is indoor scenes where algorithms can benefit from dense occlusion and structural coherence of visibility. In this kind of environment, the cell-and-portal graph (CPG) is commonly used to solve the visibility queries. A CPG is a graph that encodes the visibility structure of the scene. The nodes of this graph are *cells*, which correspond to the rooms of a building. They are connected by *portals* that correspond to openings (e.g. the doors and windows). A cell can only see other cells through portals.

CPGs are often constructed by hand, with some helper tools implemented in the modeling application. Therefore this task remains difficult and the results non satisfying: being able to create automatically a cell-portal subdivision for arbitrary models would result in substantial savings of time and money in the game industry[4]. Furthermore, it would free the non specialist from the tedious portal placement and cell creation process. Unfortunately, it is difficult to compute automatically an optimal decomposition, since a scene can be partitioned into an unlimited number of different cell-and-portal graphs. Some solutions have been proposed for very particular scenes: architectural BSP [23] [5] and tubular-like models [16]. To the best of our knowledge, no previous published method is usable in all situations.

In this paper, we propose to restate the problem of the CPG generation algorithm in terms of 3D image segmentation (in image processing, segmentation is the process of separating the different objects in an image). The image we use is an intermediate volumetric representation of the scene, the objects to separate being the cells and the separators being the portals. This formulation enables us to adapt a powerful tool from the image segmentation literature, the watershed transform (WST)[18]. Contrary to other approaches, the resulting algorithm is able to compute automatically the CPG of any

[†] Artis is a research team of the GRAVIR/IMAG Laboratory, a joint effort of CNRS, INRIA, INPG and Université Joseph Fourier-Grenoble I

indoor scene without imposing any modeling constraint.
The remainder of the paper is organized as follows: after reviewing some relevant work, we present the CPG generation algorithm in section 3, followed by its implementation (section 4). To limit the memory cost, we present a hierarchical version of the algorithm in section 5. Potential problems and their solutions are investigated in section 6. Finally, we discuss the results and present avenues for future work in section 7.

2. Related Work

2.1. Visibility

Many visibility algorithms were proposed to speed up rendering in interactive walkthrough applications, and the interested reader will find exhaustive surveys[10 8]. The ultimate goal of visibility determination is to detect efficiently the part of the scene that is visible from the camera. Before the z-buffer algorithm became a *de facto* standard, exact visibility determination (i.e hidden face removal) was the central issue. Nowadays, the research has shifted to occlusion culling, which concentrates in eliminating invisible geometry as early as possible in the rendering pipeline. Occlusion culling gives a rough conservative estimation of the visible geometry: the PVS (Potentially Visible Set), that may still contain invisible parts of the model which are later removed by an exact visibility algorithm.
Schaufler *at al.* made the important observation that visibility could be solved efficiently by using a volumetric data structure instead of the original scene description[21]. For this reason, they use a binary volumetric representation of the scene for occlusion culling: the inside voxels and the hidden voxels act as blockers to eliminate the invisible geometry.
In indoor scenes, cell-and-portal graphs have been widely used for visibility determination [17 23 5 9 4]. Previous work on CPGs mainly focused on the use of the graph rather than its generation, since it was often created manually by the user [9 4]. However, some automatic generation algorithms have been proposed. The most common are the decomposition of the scene into BSP, which are CPGs themselves[23 5]. The resulting portals are aligned with the features of the scenes, constraining the method to a restricted set of geometries. Furthermore, these approaches are limited to polygonal datasets and lead to unusable CPG when the number of polygons increases, because each triangle creates a cut plane and adds a portal to the decomposition. An exception to the BSP decomposition is the algorithm proposed by Lichan et al. in the context of virtual colonosopy [16]. They make use of the particular structure of their scene, the human colon, to create the CPG. On the basis of a distance field, the center-line (i.e. the skeleton) of the colon is extracted. The colon is then partitioned with regulary spaced cross-sections, placed perpendicular to the skeleton.
As in [21], our method is based on a volumetric description of the scene. Because this representation is a distance field, our work can be seen as an extension of the algorithm presented in [16]. The main difference is that our technique was designed to deal with arbitrary scenes, in place of the very particular human colon geometry.

2.2. Distance field

A sampled distance field is a discrete scalar field, each sample point storing the distance to the closest scene geometry. It has proven to be useful in a wide range of applications in computer graphics: rendering , virtual sculpting, collision detection, path planning, morphing between objects, surface reconstruction, etc. Distance fields have been widely used in image analysis too, for example for image segmentation. Several methods, such as the distance transform [20], were introduced to cope with the high computational cost implied by the distance field computation. To reduce the memory needs, hierarchical representations [13] were proposed. Our implementation uses both of these improvements.

2.3. Watershed transform

The watershed transform (WST)[18] is a powerful image segmentation technique initially designed for 2D gray level images. One can find in [19] an introduction to the WST and its several algorithmic implementations. Although the watershed transform can be extended from 2D to more dimensions, one has to face increasing memory and computation costs when using voxel data, therefore applications of the 3D WST remain rare [24]. However, an optimized implementation, minimizing the memory consumption to treat large 3D medical datasets, has been proposed recently [11].

3. Volumetric CPG computation

In this section, we present the general principle of our approach.

3.1. Cell and portal Graph (CPG)

A scene can be decomposed into an unlimited number of cell-and-portal graphs: defining what the ideal decomposition should be is still an open issue. In terms of culling performance, a good partition is a trade-off between the number of portals and the computational cost of using them. Ideally, one wants to minimize the number of portals while trying to place them where they will help to cull the scene's geometry. In architectural scenes, this tradeoff is usually achieved by using the 'classical decomposition', where the cells are the rooms and the portals are the openings.
When using a pre-calculated CPG to speed up rendering, the visibility determination consists first in finding the cell in which the viewer is located, called *the view cell*. All the objects of the view cell are classified as potentially visible. Then the algorithm recursively checks the cells that are connected to the view cell by portals. The objects of these cells are also classified as potentially visible. A more exact visibility test with respect to a given viewpoint will classify the objects as visible through a portal if they belong to the pyramid supported by the portal's silhouette and whose apex is the view point. The set of all the visible objects is found by

testing the visibility through all the visible portals.
We propose to automatically create an approximation to the classical CPG decomposition using the segmentation of a volumetric representation of the scene. As will be shown in this section, the watershed transform is well adapted for this task.

3.2. Watershed transform

Let us begin with a 2D explanation of the watershed transform. The transformation extends easily to supplementary dimension, and the 3D watershed transform is a straightforward adaptation of the 2D process.
The watershed transform is a morphological tool that considers a function D as a topographic surface S and defines the *catchment basins* and the *watershed lines* of D by means of a flooding process: a hole is pierced at each local minimum of the surface S, afterwards the surface is plunged into a lake at a constant vertical speed. The valleys of S are flooded by the water entering through its holes (see figure 1(a)), creating basins (i.e. the catchment basins). During this process, the water coming from different minima may merge. To avoid this, we build a dam on the point where these floods would merge (see figure 1(b)). A difficulty arises when the location of the fusion occurs on a plateau. Suppose that two regions, noted A and B present a the iteration t-1, merge during the iteration t into a single region C. The merge is detected and a dam is built to separate A and B. In the classical watershed transform, the dam is placed on the SKIZ (Geodesic Skeleton by Zones of Influence, defined in the appendix) of A and B into C. At the end of the process, all the surface is under water: only the dams emerge 1(c)). The watershed of D is defined by the set of dams, separating the catchment basins. Each of these basins contains a single minimum of D.

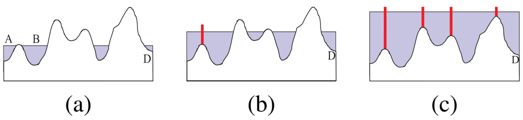

Figure 1: *Different steps of the watershed algorithm.*

The watershed transform is widely used for image segmentation. For example, if the watershed transform is applied to the gradient of a greyscale image, the watershed lines correspond directly to the contours and each catchment basin corresponds to an object of the image.

3.3. CPG generation algorithm

When applied to the distance map representation of the scene (see figure 2), the classical watershed algorithm can be adapted for the generation of a CPG. To remain compatible with a flooding metaphor, the field D is inverted before the algorithm takes place (i.e. each distance is given a negative sign).

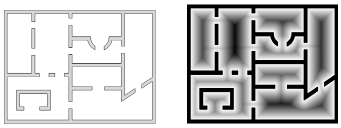

Figure 2: *Distance field representation of an architecural model (after inversion).*

The height field representation of D shows that the saddle points of the surface correspond to the portals locations (cf. figure 3). In 3D, these locations correspond to the pinches of the distance field.

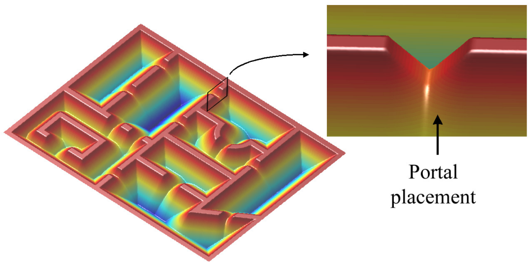

Figure 3: *Distance field used as topographic surface. A saddle point, corresponding to one of the portal, is highlighted.*

The progression of the watershed algorithm applied to D is depicted in Figure 4.

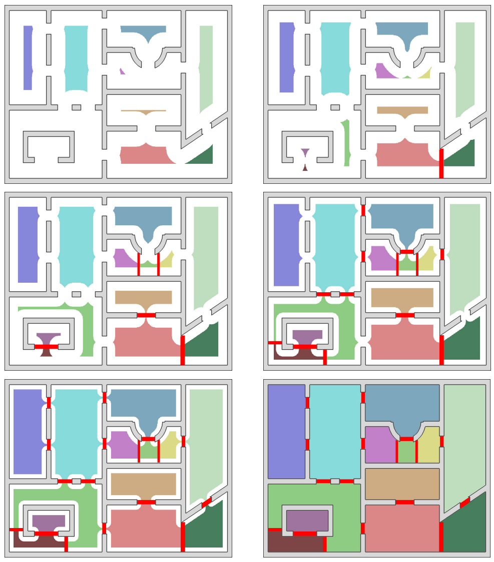

Figure 4: *Different steps of the watershed algorithm. Two catchment basins getting in contact during an iteration reveal the presence of an opening: a portal is built to separate them.*

The result of this segmentation is close to the intuitive classical CPG decomposition described above (see last image of figure 4):

- the cells correspond to the catchment basins of D. They are the volumetric space enclosed by the geometry of the scene, and separated by the portals. One can see that they correspond to the rooms of the architectural scene.
- the dams separate the catchment basins: they directly correspond to the portals. They are placed at the pinch-points of the free space in the volumetric representation of the scene. In terms of visibility, they correspond to areas where the scene's longest visibility lines (e.g. the set of segments traced between all the mutually visible points of the scene) are concentrated. This means that the portal are placed where they are supposed to be helpful for visibility culling. Moreover, in architectural models, pinches effectively encompass the doors, windows and corridors.

Note that portals are added on all pinches and that each local minima generates a cell. Due to this fact, some of the cells and portals may be unsuitable. This problem principally occurs when the initial geometry is intricate, leading to a very perturbed distance field. This is an illustration of the classical over-segmentation problem that occurs when segmenting images by watershed. We will detail in section 6 some solutions to this problem in our case.

4. Implementation

We propose an implementation organized in successive stages explained in detail below. First, the distance field from the scene representation is computed and sampled in a grid. The distance map is then used by the watershed process to find the contacts between catchment basins, where portals are to be placed. When necessary, the portals are built on the basis of the scene geometry. The cell-and-portal graph itself is computed as the watershed takes place. At the end of the algorithm, the scene geometry is assigned to the different cells.

4.1. Distance Map Construction

The distance field evaluation requires a function that computes the distance between a 3D point and its closest geometry. From the CPG algorithm point of view, the only difference between all the modeling techniques (parametric surface, volumetric representation, implicit surface, polygon soup,...) is the way this distance function is computed. For a set of independent triangles for example, the distance is the minimum distance to all the triangles in the model. For an implicit surface, the distance can directly correspond to the implicit function.

A naive brute force algorithm evaluates the distance field for every point of the distance map. Unfortunately, such a query is very costly in the case of a polygonal model representation, even if the triangles are organized in a hierarchical data structure (such as an octree, a BSP or a bounding volume hierarchy). The total distance field computation time become the limiting process.

More sophisticated algorithms make use of particular properties (continuity, bounded growth) of the distance function to speed up the computation. The distance transform [20] gives an approximation of the distance field that is very fast to compute but relatively inaccurate. This algorithm applies a local distance matrix in two successive passes over the voxel grid. Each pass propagates the local distance computed by the addition of known neighborhood value to the values obtained from the local distance matrix. The forward pass calculates the distance from the surfaces in an arbitrary top-down direction, starting from the top corner of the grid and moving away to its bottom. The backward pass calculates the remaining distances, from the bottom corner of the grid to its top. The algorithm's speed results from the simple computations involved. In our algorithm, distance transform is used because it does not compromise watershed: in practice, the distance transform only deforms the distance field locally without adding any local minima. The number of regions and their global shape are conserved. When an exact distance is required, for example in the case of portal placement (see section 4.3.2), it is computed on the fly from the initial geometry representation.

4.2. Portals Detection

This phase corresponds to the watershed algorithm. A level value, called $isoValue_t$, is initially set to the minimum value of the distance map. $isoValue_t$ defines a surface that partitions the 3D space into an inside volume -where the distance field is smaller than $isoValue_t$- and an outside volume -where the distance field is greater than $isoValue_t$- (Remember that by convention the distances have negative values). The algorithm iteratively increases $isoValue_t$. The inside volume corresponding to iteration t, called $Inside_t$, is cut into different regions. A labeling process assigns a different ID to each region, that is a unique value is assigned to voxels that belong to the same connected region. The connectivity used is a 4-connectivity in 2D and a 6-connectivity in 3D. The key step of the algorithm is to detect the catchment basins that merge during an iteration t, in order to separate them by a dam. This task is accomplished by the dilation of each regions of $Inside_{t-1}$ in the $Inside_t$ space, at the beginning of iteration t. This dilation consists in propagating the ID to the adjacent voxels (see figure 5). During this dilation, the new ID that is propagated (called NEW_ID) is compared to the ID already stored in the voxel that has been reached (called OLD_ID). Three cases are possible:

- OLD_ID is null: the dilation of the region reached a beforehand empty voxel. Nothing happens and the dilation continues (see figure 5 (b)).
- OLD_ID is equal to NEW_ID: the voxel already belonged to the region. The dilation stops.
- OLD_ID and NEW_ID are different: the voxel belonged to another region (see figure 5 (d)). The center of the voxel is the contact point P. The two colliding regions OLD_ID and NEW_ID are restored as they were before dilation, and a portal is built to separate them (see section 4.3). The dilation process restarts with the dam included (see figure

5 (e) and (f)). If nothing had been done, the two regions would have merged during the labeling of iteration t.

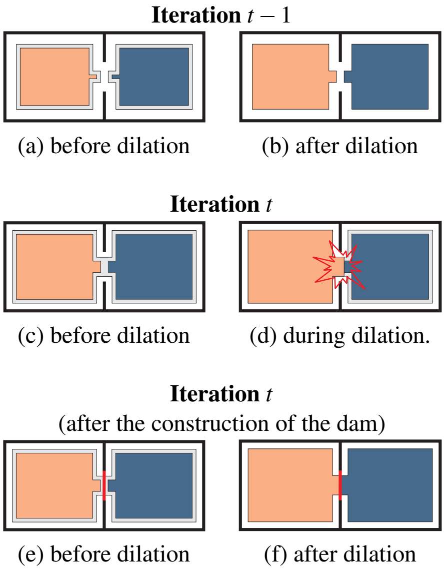

Figure 5: *ID propagation to detect the portals.*

4.3. Creation of the portals

4.3.1. Portal definition

The SKIZ, where the dam is built in the watershed transform, is not really adapted as portal definition. Firstly, it is entirely dependent on the region's shape: there is no certainty that the SKIZ touches the initial geometry. Secondly, the SKIZ is a 3D set of voxels that are not adapted for scene culling. Instead, we define the dam as being an oriented planar rectangle, located at the pinch-point of the distance field and in contact with the geometry responsible for the pinches (and bounding it). In the distance map, all the voxels that are crossed by a portal are tagged as a dam. A dam is an obstacle: the propagation algorithms (i.e the labeling and the dilation processes) can not traverse it. This dam definition is the most important difference between our algorithm and the classical watershed transform.

When a contact point (P) between regions A and B is discovered, two portals are built to separate them. The first is positioned with the distance map information only, the second with the help of the graphics hardware. The smaller of these two portals is chosen.

4.3.2. Distance map portal positioning

Just before starting the portal positioning algorithm, the $inside_{t-1}$ of the two regions to separate have been restored. This enables us to compute an estimation of the location of the portal from the distance map representation. The location of the portal is defined by its normal and a point noted M:

- Intuitively, a good normal direction is given by the skeleton of the distance map, that crosses the portal [16]. It can be approximated by the segment joining the two mutually closest points (noted C_1 and C_2) of the iso-surface of the regions A and B at iteration $t-1$. The distance used is the geodesic distance, defined as follow.

  Geodesic distance Let x and y two points of the set X. The *geodesic distance* between x and y is the length of the shortest path included in X and linking x and y.

  In practice, it means that the segment $\overline{C_1C_2}$ can not cross any scene geometry to be valid. In the case depicted in figure 6(a), the euclidean distance would give the segment $\overline{XY}$, in place of the correct segment $\overline{C_1C_2}$.
- M is located on the middle of the segment $\overline{C_1C_2}$.

We use a simple procedure to find C_1 and C_2 (see figure 6(b)). Starting from the contact point P, we search its closest point located on the surface of the region B, noted B_1. Then we search the closest point to B_1 that is located on the surface of the region A: it is C_1. Finally, we find C_2, which is the closest point to C_1 located on the surface of the region B. This procedure ensures that we find the correct pinch when there are multiple pinches of equal size between A and B.

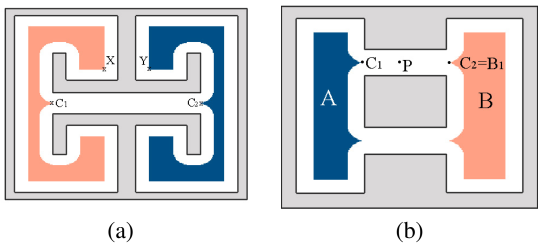

Figure 6: *(a) use of a geodesic distance: the segment joigning the two closest points can not cross the geometry. (b) Finding C_1 and C_2. In this simple case, B_1 equals C_2.*

4.3.3. Hardware-assisted portal positioning

The first positioning may fail when the portal is thin, because there is no certainty that the segment $\overline{C_1C_2}$ will give a correct orientation (see figure 7).

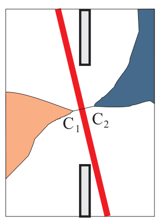

Figure 7: *Distance map portal positioning fails when the distance field is noisy and the portal is small.*

For this reason, a second positioning algorithm was developed in the case of thin portals. It consists in leaning the portal on the geometry responsible for the pinch, with the

help of the graphic hardware. At first, a camera with an large field of view is placed on C_1, pointing in the direction of C_2. A quadrilateral of arbitrary large size, is placed in C_2 perpendicular to C_1C_2. It is rendered in red color, followed by the geometry of the scene in white. The frame buffer and the corresponding depth buffer are read back. The center point of the frame buffer belongs to the portal because it intersects $\overline{C_1C_2}$: the red region of the frame buffer that contains this point is selected. The exterior silhouette of this red region defines the portal. The corresponding pixels are back-projected in the 3D space. In that way, the best fitting plane of this set of points defines the portal direction $\overline{n}$.
Each positioning algorithm give its own portal location. To choose between them, the extent of both portals is computed, and the smaller portal is chosen.

4.3.4. Extent of the portal

Graphics hardware is also used to determine the 3D extent of the portal. A camera, with a field of view of 90 degrees, is placed at the center of the portal, perpendiculary to its direction ($\overline{C_1C_2}$ or $\overline{n}$ following the case). It is used to make four successive renderings followed by z-buffer reading. The camera is rotated by 90 degrees between each rendering. The intersection of the portal plane and each of the rendered image is a pixel line: all the silhouette's pixels are back-projected. The corresponding 3D points are coplanar, since they all belong to the best fitting plane, and the portal is simply their smallest oriented bounding rectangle. An improvement (not yet implemented) would be to clip this rectangle by the scene geometry to get a more complex portal shape that maximizes its culling efficiency.

4.4. Creation of the cells

The cells are the discrete volumetric spaces enclosed by the geometry and separated by the walls. They are not necessarily convex. The distance map resolution fixes the size of the smallest room that the algorithm can find: it must be chosen high enough to have at least one voxel inside each room. The storage of a geometric description of the cell is necessary to locate the viewer during the online rendering process. Unfortunately, each possible solution has some drawbacks:

- a volumetric description is well adapted for the viewer's location, but the memory cost is very high.
- the exact polygonal representation of the surface (extracted from a marching cube algorithm for example) would require many triangles and would not be very efficient to locate the viewer.
- the storage of the surface into a BSP would solve the efficiency problem, but the memory cost would not be negligible and the construction of a good BSP is not straightforward.

Instead of storing the cell description explicitly, our implementation uses a simpler solution: each geometry element (i.e. a triangle or a triangle mesh) stores a pointer to the cell of the CPG it belongs to. In return, each CPG cell contains the list of objects composing it, organized by a hierarchical tree to speed up the portal culling (for example a binary tree of Axis Aligned Bounding Boxes). At the beginning of the visualization, the viewer is localized by casting a ray from the viewpoint through the scene (see figure 8(a)). When a portal is crossed before the hit, other rays are cast in random directions until the ray first hit the geometry. This situation can easily be avoided by tracing vertical rays, because a typical indoor scene does not contain many horizontal portals. During the rest of the walkthrough, cell changes are simply detected by testing when the viewer crosses a portal while moving.
The correspondence between the scene geometry and the catchment basins is not straightforward: as can be seen in figure 8(b), the geometry does not geometrically lie in any catchment basin at the end of the CPG generation algorithm: by definition, the geometry separates them. In our implementation, each triangle is assigned to its closest catchment basins that are seen by the triangle in the direction of its normal. To find them, we check the closest catchment basin to four points of the triangle: its gravity center and its vertices. This procedure may fail with large triangles belonging to several cells because the sampling density may not be sufficient. However, the number of test points should be increased to handle this case. The triangles crossed by portals are assigned to a unique cell: for the visualization, it is not necessary to assign them to both cells since these triangles are always classified as visible through the concerned portals. However, his can potentially lead to a localization error at the beginning of the walkthrough. This problem can easily be solved by subdividing the triangles concerned (solution not yet implemented).

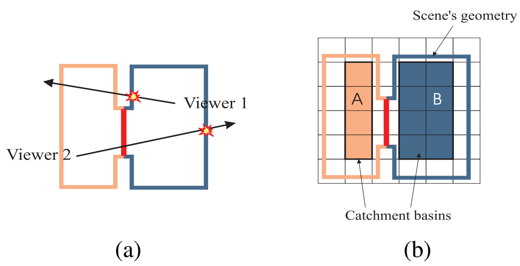

Figure 8: *(a) Viewer localization: contrary to the viewer 2, the viewer 1 is correctly located because the ray does not cross any portals before the hit. (b) Creation of the cells: The geometry is associated to its closest catchment basin.*

5. Hierarchical implementation

For complex scenes, the memory required to store the distance field becomes a concern and a compression scheme is needed. In practice, we did not implement the algorithm on a regular grid for this reason. Instead, we use the Adaptively Sampled Distance Fields (ADFs) as the basic data structure for the implementation [13]. An ADF is an octree, each node storing the distance values of its eight corners. The distance inside a cell is computed by tri-linear interpolation. In addition, minimum and maximum distance values are stored in each node. For a leaf node, these values correspond to the minimum (resp.maximum) of the eight distance values. For an internal node, these values correspond to the minimum (resp. maximum) of the distance of all the child nodes. These minimum and maximum values suffice for most of the operations. Let S be the iso-surface defined by an iso-value d.

- An octree cell is totally inside S if ($cell.maximum >= d$)
- A node contains a portion of S if ($cell.maximum >= d$) and ($cell.minimum <= d$)

In our implementation, the eight distances values are discarded to save memory. When a precise distance is required, for example for the approximate portal placement, its is computed on the fly from the initial geometry. The multi-resolution aspect is the main advantage of ADFs: the refinement predicate ensures that details are added only where necessary. For CPG generation, the areas of interest, where it is interesting to refine the octree, are the pinches of the distance field -they give the portal location- and its protrusions -they give the portal's approximate position. Both correspond to areas where the distance field is non-linear. For this reason our predicate refines the nodes for which the tri-linear interpolation gives poor results in comparison to the exact distance values. The threshold error of the refinement criterion has to be taken sufficiently small to ensure that our algorithm does not miss some rooms of the building due to the ADFs data structure. Intuitively, a small threshold imposes that the skeleton of the distance map - which passes by the portals and the center of the rooms - will be refined to the maximum depth. The ADFs construction begins with the computation of a full resolution distance field with the distance transform. Then, the octree is computed by merging the linear portion of the distance field. When the initial full resolution grid does not fit in memory, a mechanism of swap between the central memory and the hard disk is used. The hierarchical CPG generation algorithm is very similar to the grid version. Only the 'portal discovering' part of the algorithm is modified, the portal placement algorithm remaining the same. The major changes are:

- the use of min and max value stored in every node to speed up the algorithms: the portions of the octree that are not concerned by a treatment (labeling, propagation, dilation, ...) can be easily skipped.
- the navigation between neighbor node in the octree is not as straightforward as in the case of a regular grid [7 14 12].

6. Over-segmentation

In image processing, the watershed segmentation is known to be sensitive to data noise: the noise often generates many local minima and each of them become a catchment basin during the watershed transform. As a result, too many regions are created, leading to the so-called over-segmentation problem. The implication for CPG generation is that when the distance field presents many local minima (typically in the case of furnished rooms, zigzag walls,...), the model will be decomposed into many cells separated by redundant portals. To solve this problem, two approaches are commonly used. The first one consists in pre-processing the data in order to eliminate the local minima before the watershed transform. The second consists in post-processing the results in order to reduce the number of regions by merging several regions into bigger ones, on the basis of a given heuristic. We propose in this section two pre-processing techniques that solve the over-segmentation problem in most of the observed cases. The first method is the manual simplification of the scene while the second one is its voxelisation. Post-processing methods were not investigated here, but one could imagine to add a treatment at the end of the process that consists in eliminating the useless portals, on the basis of the estimation of their culling efficiency.

6.1. Manual Simplification

Manual simplification is a classical method [22], that requires little additional work during the modeling process. It consists in discarding the unimportant objects for visibility determination (e.g. the furniture) when computing the CPG, and adding them back to the cell at the end of the process. It can be seen as a median filtering of the geometry, already known to remove the noise from images. In practice, this approach provides good results, but the manual elimination can become a tedious task when it is not coupled to the modeling process. For this reason, we also propose to use voxelisation as fully automatic treatment to reduce the noise present in the distance field.

6.2. Scene voxelisation

A binary voxelisation of the scene can be used as support for the distance computations instead of the initial geometric scene representation. For sets of independent triangles, the algorithm presented in [15] is used. It was designed to treat any scene, even those containing degeneracies such as double walls, interpenetrating meshes, cracks or holes. The volumetric representation obtained is cleaner and free of problems: the noise in the distance field generated by the initial degeneracies is greatly reduced. Furthermore, the voxelisation acts as a low-pass filter, the size of the voxels fixing the size of the geometry details that are eliminated (of course walls have to be large enough to be part of the volumetric representation). The filtering eliminates many of the local minima. Another advantage is that the voxelisation provides signed distance because the representation distinguishes the inside and the outside of the objects. The inside regions do not need to be treated by the CPG creation algorithm. For all of these reasons, the voxelisation is an efficient preprocessing of the scene, that eliminates most of the over-segmentation and the potential modeling problems in a fully automatic way.

7. Results and Discussion

We implemented the hierarchical algorithm described in this paper and have tested it on different kinds of scenes to show its generality. All the experiments were performed on a computer with a Pentium III processor (800 Mhz) and an Nvidia GeForce 2 GTS graphic card. The first scene is an architectural model of a house (147k triangles), publicly available on the Hybrid Graphics company website [2]. This model is particularly interesting because it is used in their commercial visibility package *dPVS* as a demonstration of cell-and-portal graph rendering to speed up complex scene walkthroughs. A CPG, created by hand, is associated to the model and can directly be compared to the result of our algorithm. Before computing the distance field, we have discarded the doors and windows with a standard modeling package, as well as the furniture to eliminate over-segmentation. Our algorithm has then been used to create the CPG decomposition. The resulting CPG is very similar to the original one and contains all its portals (see figure 9). The only differences are the portals added by our algorithm on the exterior windows and a portal added on the central corridor.

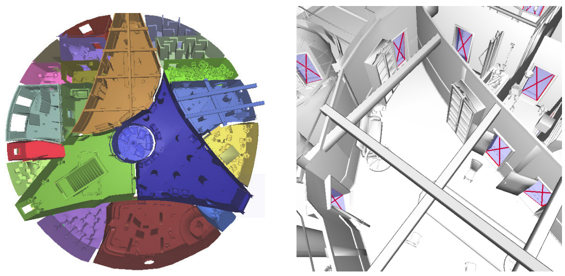

Figure 9: *Scene 1- complex architectural house (from [2]). Left: cells created by our algorithm. Right: some of the portals created by our algorithm (scene treated without the furniture, added back to the cells at the end of the process).*

This work was partially done in collaboration with the computer games company Appeal. The next two scenes are coming from their in-development game, Outcast 2. These scenes contains many degeneracies, principally coming from the modeling process based on geometry instanciation: some parts of the scene are made from different basic objects, replicated at different locations with scale factors and distortion parameters. Although instanciation is an elegant way to reduce the memory cost, it introduces artifacts such as interpenetrating geometry, cracks and holes. Without taking care of these degeneracies, the distance field would be very noisy and would count many uninteresting local minima. For this reason, a voxelisation process has been used to filter the details and the artifacts before creating the CPG.

Scene 2 is a house made of 81k triangles. For this model, our algorithm correctly found all the arches doors and handles correctly the center room made up of successive pillars (see figure 10).

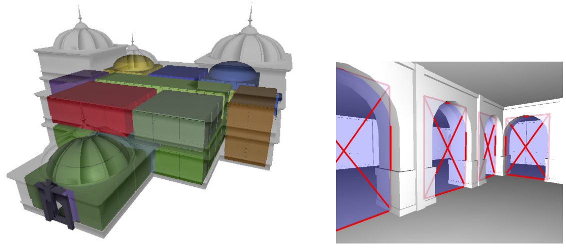

Figure 10: *Scene 2 (copyright (c) Infogrames)- house, containing many degeneracies. The figure on the right shows the center room, made off successive pillars. This scene was treated after a pre-voxelisation step.*

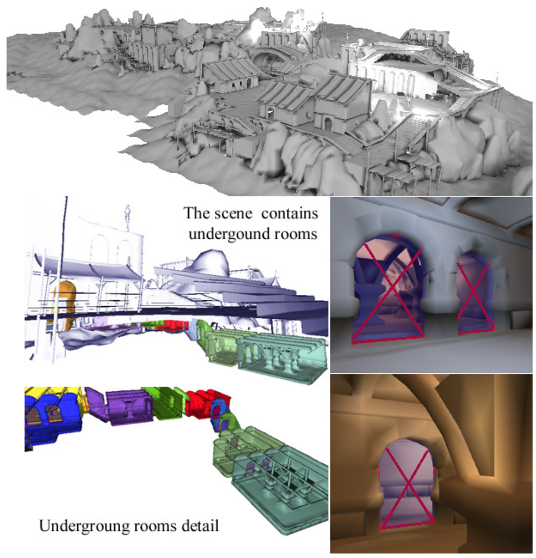

Figure 11: *Scene 3 (copyright (c) Infogrames)- Mixture of indoor/outdoor scene, containing many degeneracies. The algorithm creates a CPG for the underground rooms, while not partitioning the rest of the scene.This scene was treated after a pre-voxelisation step.*

Scene 3 is a stone landscape with underground rooms (464k triangles), and represents a mixture of indoor/outdoor scene in 3D. The algorithm created a CPG for the indoor part of the scene (the underground rooms), and does not partition the outdoor parts of the scene that are not well suited for CPG rendering (see figure 11). The reason is that the flooding originates where the model presents enclosed space that constitute local distance minima. When no such minima exist, a single region is created and no portal is built. In the case of 2D 1/2 visualization of terrain and city, one could want to use a CPG, with the portals placed between the different peaks and the different buildings. To extend the CPG creation to this kind of environment, a simple solution consist in computing a 2D watershed on the 2D height map

where the minima effectively exist. Another solution could be to design an interactive tools to add the appropriate artificial local minima in the distance field.
Scene 4 is the Berkeley Soda Hall model, without the furniture (18k triangles)[3], that was treated without preprocessing. The algorithm finds all the rooms, doors and windows of the building (see figure 12). An open issue is the treatment of the saddle lines of the distance field that typically occur in a corridor. Our method places a unique portal on the middle of this line, but a gain could be obtained by placing portals at the beginning and at the end of the saddle line (i.e. the beginning and the end of the corridor).

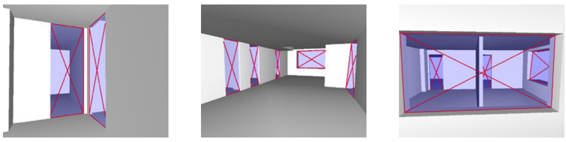

Figure 12: *Scene 4 - Soda Hall model. In a building, the algorithm finds all the rooms, doors and windows.*

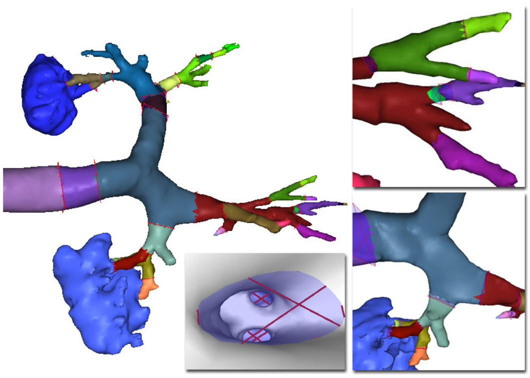

Figure 13: *Scene 5- Human airway. The cave-like environments are correctly handled by the algorithm.*

The last scene is a medical model used in virtual endoscopy and representing a human airway[1] (49K triangles). The results show that the algorithm is able to create satisfactory CPG for cave-like environment (see figure 13). The number of cells and portals and the computation times of the different modules of the algorithm are summarized in table 1. The remainder of the table gives some information about the resolution and the memory cost of the distance map. The last row indicates the compression rate obtained when using an octree in place of a regular 3D grid. The grid considered is the smallest grid containing all the scene, at the same resolution as the octree, and storing all the variables necessary for our implementation of the WST algorithm. The compression rate is good with sparse scenes (scene 3) but decreases when the scene geometry is uniformly distributed (scene 4). In that case, the octree is refined to the maximum depth nearly everywhere: computation time is high and the memory cost is still an issue. The algorithm could be adapted by treating the model region by region and combining the CPG created for the different regions in a final step.

Scene	1	2	3	4	5
# Portal	33	25	13	402	30
# Cells	31	18	22	511	49
Pre-voxelisation	-	8 min.	45 min.	-	30 s.
Distance Transform (sec.)	7	20	129	73	62
Octree creation (sec.)	14	21	82	140	35
Portals Detection (sec).	154	18.5	182	>2hours	80
Portals Positioning (sec.)	27	27.5	45.4	>2hours	39.6
Cells Creation (sec.)	30	2.2	102	4	2
Octree Depth	8	8	10	9	9
Memory Cost (Mo)	14	47	90	224	32
Compression	6/1	6/1	21/1	5/1	30/1

Table 1: *Results*

Measuring the quality of the results is a difficult task since comparison points are missing: the generation of CPG was commonly resolved manually and to the best of our knowledge, existing automatic methods can not handle general scenes. The results could be compared with the one given by a BSP decomposition for the simplest model (scene 4). Unfortunately, this kind of decomposition is not straightforward for the other scenes containing degeneracies and hundreds of thousands of triangles. Furthermore, it is not clear that a BSP decomposition could treat the cave-like scene such as human airways. In this context, we only compared our results with manually created decompositions, even though they are probably not optimal in terms of visibility. For each of our test scenes, this comparison shows that our automatic method finds all the 'classical' portals, but may add redundant portals as well due to the remaining over-segmentation. As future work, a post-process could be implemented to detect and remove them (cf. section 6). Another indication of the quality of the decomposition is given by the speedups obtained when using the CPG for rendering. For this task, we have implemented a simple cell-and-portal rendering algorithm and have tested its performance on typical paths for each test scene. Table 2 compares the frame rates (Fps) obtained without any visibility test(HR), with hierarchical frustum culling (HFC) only and with the portal rendering algorithm coupled to HFC (CPG).

Fps	HR			HFC			CPG		
Scene	min.	avg.	max.	min.	avg.	max.	min.	avg.	max.
1	7.5	8	8.5	8.5	58.9	753	23.2	188.2	1816
2	24.4	33	40.2	48.2	211	757	155	718	1650
3	6.66	7.4	7.8	13	46.1	168	183	561	1672
4	31	52	61	67	774	1632	597	1610	3170
5	16	23	26	34	156	485	42	345	2193

Table 2: *Rendering performances*

As expected, the speedups are up to an order of magnitude for the different scenes. Even if it does not prove that the generated decomposition is optimal, it shows that the automatic method presented can be used in a real-life scenario to accelerate the rendering process.

8. Conclusions

In this paper, we restated the problem of automatic cell-and-portal graph generation in terms of 3D image segmentation. For this task, we adapted a classical segmentation tool, the watershed transform. The results show that our approach finds a decomposition closely linked to the structure of the models, while not imposing any particular modeling constraint. In future work, our algorithm could directly benefit from the improvements already published in the watershed literature such as marker-controlled watershed to reduce over-segmentation or computational and memory optimized implementations.

9. Acknowledgments

We would like to thank G. Debunne, M. Cunzi, J.-D. Gascuel, C. Chaudy, P. Van Ham and N. Warzée for all their help and advice.

References

1. Diagnostic radiology department website. `http://www.cc.nih.gov/drd/endoscopy.htm`.

2. Hybrid graphics website. `http://www.hybrid.fi/index.html`.

3. Sodahall vrml jumpthru website. `http://http.cs.berkeley.edu/~kofler/`.

4. Timo Aila. *SurRender Umbra: A Visibility Determination Framework for Dynamic Environments*. PhD thesis, Helsinki University of Technology, October 2000.

5. John M. Airey, John H. Rohlf, and Frederick P. Brooks, Jr. Towards image realism with interactive update rates in complex virtual building environments. *Proceedings of ACM Symposium on Interactive 3D Graphics*, pages 41–50, March 1990.

6. S. Beucher. The watershed transformation applied to image segmentation. *Conference on Signal and Image Processing in Microscopy and Microanalysis*, pages 299–314, September 1991.

7. Parthajit Bhattacharya. Efficient neighbor finding algorithms in quadtree and octree. Master's thesis, Indian Institute of Technology, Kanpur, 2001.

8. D. Cohen-Or, Y. Chrysanthou, and C. T. Silva. A survey of visibility for walkthrough applications. *Proceedings of SIGGRAPH*, course notes, 2000.

9. Luebke David and Georges Chris. Portals and mirrors: Simple, fast evaluation of potentially visible sets. *Proceedings of ACM Symposium on Interactive 3D Graphics*, pages 105–106, April 1995.

10. Frédo Durand. *3D Visibility: Analytical Study and Applications*. PhD thesis, Université Joseph Fourier, Grenoble I, July 1999.

11. Petr Felkel, Mario Bruckschwaiger, and Rainer Wegenkittl. Implementation and complexity of the watershed-from-markers algorithm computed as a minimal cost forest. *Computer Graphics Forum*, 20(3), 2001.

12. Sarah F. Frisken and Ronald N. Perry. Simple and efficient traversal methods for quadtrees and octrees. *Journal of Graphics Tools*, 7(3):1–11, 2002.

13. Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. *Proceedings of SIGGRAPH*, pages 249–254, 2000.

14. Samet Hanan. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, 1990.

15. Denis Haumont and Nadine Warzée. Complete polygonal scene voxelization. *Journal of Graphics Tools*, 7(3):27–41, 2002.

16. Lichan Hong, Shigeru Muraki, and Arie Kaufman ans Dirk Bartz ans Taosong He. Virtual voyage: Interactive navigation in the human colon. *Proceedings of SIGGRAPH*, pages 27–34, 1997.

17. C. B. Jones. A new approach to the 'hidden line' problem. *Computer Journal*, 14(3):232–237, August 1971.

18. F. Meyer and S. Beucher. Morphological segmentation. *Journal of Visual Communication on Image Representation*, 1(1):21–46, September 1990.

19. J. B. T. M. Roerdink and Arnold Meijster. The watershed transform: Definitions, algorithms and parallelization strategies. *Fundamenta Informaticae*, 41(1-2):187–228, 2000.

20. Azriel Rosenfeld and John L. Pfaltz. Sequential operations in digital picture processing. *Journal of the ACM*, 13(4):471–494, 1966.

21. Gernot Schaufler, Julie Dorsey, Xavier Décoret, and François Sillion. Conservative volumetric visibility with occluder fusion. *Proceedings of SIGGRAPH*, pages 229–238, 2000.

22. Seth Teller. *Visibility computation in densely occluded polyhedral environments*. PhD thesis, UC Berkeley, CS department, 1992.

23. Seth Teller and Carlo Séquin. Visibility preprocessing for interactive walkthroughs. *Proceedings of SIGGRAPH*, pages 61–68, July 1991.

24. S. Wegner, H. Oswald, and E. Fleck. The 3d watershed transform on graphs. *SPIE Conference on Image Processing*, pages 264–273, February 1998.

Appendix A: SKIZ definition (from [6])

Consider a set of regions Y_i included in a region X. The *zone of influence* of Y_i is the set of all the points of X that are at a finite geodesic distance from Y_i and closer to Y_i than to any other Y_j. The boundaries between the various zone of influence gives the *geodesic skeleton by zones of influence (the SKIZ)* of Y in X. The SKIZ can be computed by successive dilations of the Y_i into the set X.