

## Carving: a novel method of visibility preprocessing for un-restricted three-dimensional environments

Yuong-Wei Lei, Ming Ouhyoung

Communications & Multimedia Laboratory, Department of Computer Science and Information Engineering, National Taiwan University, 1, Sec. 4, Roosevelt Road, Taipei, 106, Taiwan, R.O.C.  
e-mail: f0506063@csie.ntu.edu.tw  
ming@csie.ntu.edu.tw

We propose a novel idea, *carving*, for visibility preprocessing of unrestricted (nonaxial) models. It is different from visibility preprocessing applied directly to nonaxial data. As a sculptor carves a statue, carving splits the nonaxial portion of a model, and results in a model with only axial polygons and a set of cuttings. The visibility information of the carved axial model can then be computed with the cuttings as detail objects.

**Key words:** Visibility preprocessing – Interactive walkthrough – Virtual reality – Constructive solid geometry

Correspondence to: M. Ouhyoung

## 1 Introduction

Interactive walkthroughs, first introduced by Brooks (1986), simulate the sensory experience of navigating through a complex three-dimensional architectural environment. In the SpaceWalker project – our plan to construct an interactive walkthrough system (Lei 1997) – the target environment is the new building housing our computer science department that is currently partially completed. Our major challenge is that, after the radiosity computation, even just a single floor and unfurnished model will consist of 40 180 triangles. The whole building will require far more than the currently available mid-range workstations to be rendered at an interactive frame rate. However, in a densely occluded environment, especially in a building's interior, observers can only see a small fraction of the model from most view points. Therefore, only the polygons that contribute to the resulting scene need be considered in order to produce the correct scene. Since the rapid identification of the visible portion of each frame is very important, and since the architectural model is not changed during a walkthrough, the visibility information can be precomputed off-line as much as possible. While developing the visibility preprocessing module, we encountered the practical problem that our model was in general *unrestricted* i.e., the polygons of the model were not *axial* (short for *axially aligned*; in this paper we say a polygon is axial if its normal and edges are parallel to the  $x$ ,  $y$ , or  $z$ -axis).

One of the characteristics of traditional Chinese architectural design style is often demonstrated by windows and passageways that are round, arched, or of a polygonal shape such as an octagon. This tradition, combined with architectural design styles originating in the West, strongly influence today's campus buildings at our university, in which there are many nonrectangular artifacts like windows, arches, and passageways (Fig. 1). For example, the obvious nonaxial constructions in our department building are eight arch-shaped windows in each floor, a large arch shaped window in the fifth floor, and an octant passageway in the basement (Fig. 1, upper left and upper right). While an efficient and effective method of visibility preprocessing for nonaxial models has been proposed by Teller (1992b) and Teller and Hanrahan (1993), a robust and practical implementation is still difficult. The complexity of *generally oriented* polygons pose many problems both

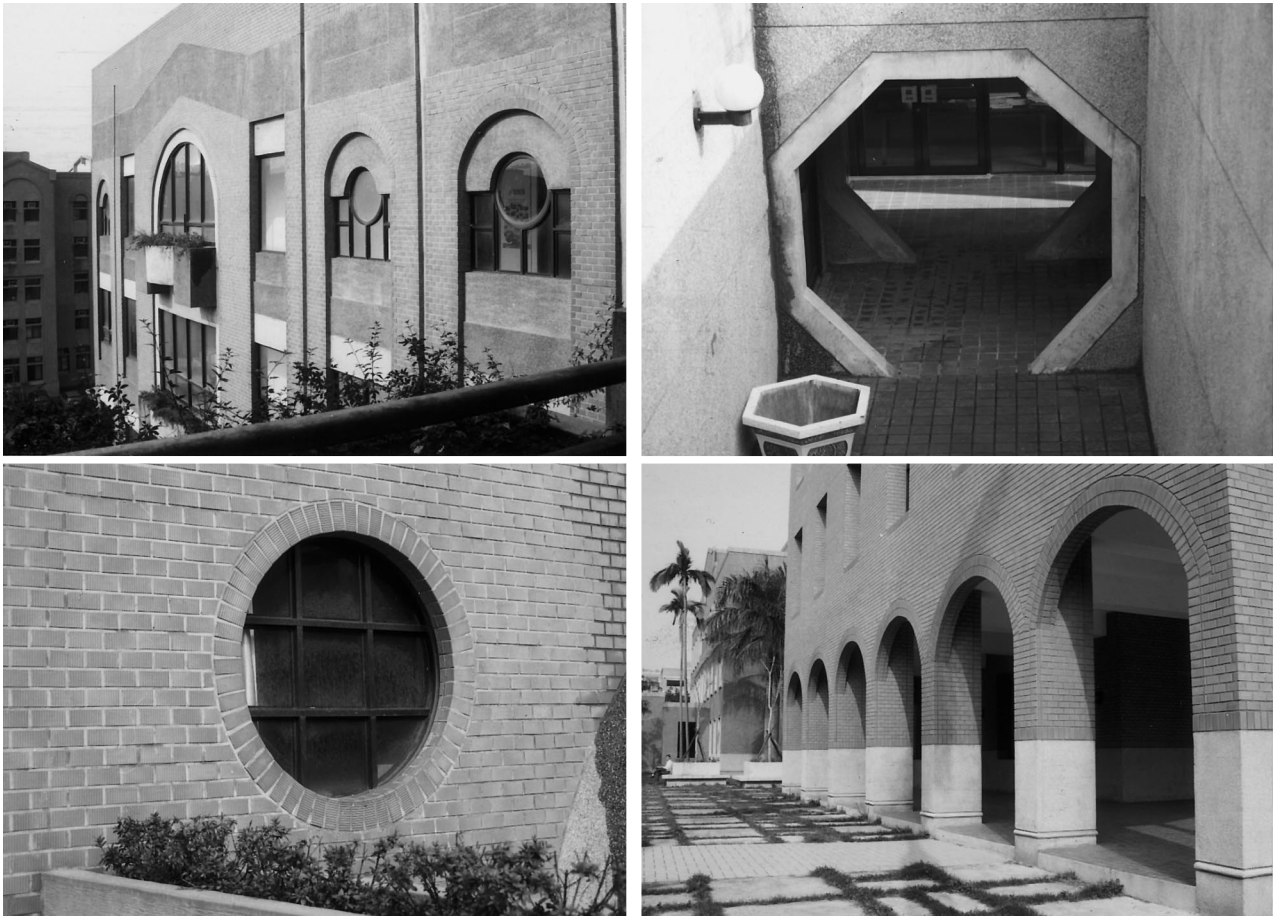


Fig. 1. Buildings at the National Taiwan University (NTU) campus with nonaxial components

conceptual and technical in nature. Based on the observation that most polygons of an architecture model are axial, in this paper we propose a new idea called *carving*. Different from employing visibility preprocessing directly on nonaxial data, carving works as follows.

Just as a sculptor carves a statue, carving splits the nonaxial portion, resulting in a model with only axial polygons and a set of *cuttings*. Then we precompute the visibility information of the carved axial model with the cuttings as detail objects [here we use the terms in Teller 1992b]. By carving we mean not only a classification of polygons, but a variation of the boolean set *SUBTRACTION* operation of constructive solid geometry to keep the resulting axial portion valid. This paper is organized as follows. In Sect. 2 we

review the visibility preprocessing. Depending on the geometry of models, we define three classes of models and make two assumptions about input data in Sect. 3. Section 4 presents the carving operation and discusses each step of the operation, and at the end of the section, a pseudo-code list is given. Practical applications of the carving operation in the interactive walkthrough system are given in Sect. 5 with some discussions.

## 2 Visibility preprocessing for interactive walkthroughs

In this section, we describe the visibility precomputation briefly. More detailed descriptions can be found in Teller (1992b) and Teller and Séquin (1991).

Since computing the *exact* visible portion for every viewpoint and every direction is conceptually and computationally intractable, Teller (1992b) and Teller and Séquin (1991) propose a method to partition the model spatially into convex cells, and then compute approximated visibility information of the cells off-line. Given this potentially visible set (PVS) (Airey 1990; Airey et al. 1990), which overestimates the set of exactly visible polygons, modern, widely available, polygon-rendering hardware can solve the hidden surface problem. In Teller's original assumption, a building model is comprised of *major occluders* and *detail objects*. The major occluders are large, simple, structural elements – for example, walls, floors, beams, and ceilings that generally cause substantial occlusion. The detail objects are small, complex things such as furnishings and utensils that generally do not occlude much. With such an assumption, the visibility information is constructed and maintained in four stages.

*Spatial subdivision.* The geometric model is subdivided along occluders into convex polyhedral cells that are of limited extent compared to the entire model. Any spatial subdivision method supporting a *point location query* (given a point, find the cell containing the point), *portal enumeration* on cell boundary, and *neighbor finding*, such as the BSP tree (Fuchs et al. 1980), will satisfy the need. We say that a spatial subdivision method that supports such operations is *conforming*.

*Portal enumeration and object population.* The *portals*, which are nonopaque portions of the cell boundary, are enumerated and stored with each leaf cell. Along with each portal is an identifier for the neighboring cell to which the portal leads. Thus the portals and the identifiers comprise a cell adjacency graph over the subdivision leaf cells, in which two cells are adjacent if and only if there is a portal on the shared boundary.

Detail objects are then distributed into cells. Detail objects are treated differently from major occluders. A spatial cell is *populated* with an object if the object, or the bounding volume of it, intersects the cell. A critical operating assumption that *all detail objects are unoccluding* means that detail objects do not block the propagation of observers sightlines.

*Visibility propagation.* The *cell-to-cell visibility* information is then computed. A cell's visibility is the region to which an unobstructed sightline can

lead from some points inside the cell. In other words, a given cell can see a neighbor only through a shared portal, and into more distant cells only through portal sequences. Therefore, a given cell can see into distant cells only if there are sightlines emanating from this cell intersection, or *stab*, the ordered lists of portals (Hohmeyer and Teller 1992; Pellegrini 1990; Teller 1992a). The portal sequences are generated incrementally by a depth-first search (DFS) of the cell adjacency graph and terminated when a sequence no longer admits a sightline. A *stab tree* for *static* visibility information is stored for each leaf cell.

*On-line culling.* The cell-to-cell visibility is an upper bound on the viewing capabilities of an unconstrained observer, who is able to look simultaneously in all directions from all positions inside the cell. However, the observer in a walk-through is at a known point and has a limited view cone. Therefore, the *eye-to-cell visibility* – the set of cells partially or completely visible to an observer with a specified view – is a subset of the cell-to-cell visibility. Several methods for computing the eye-to-cell visibility, different in effectiveness and complexity, are discussed by Teller (1992b).

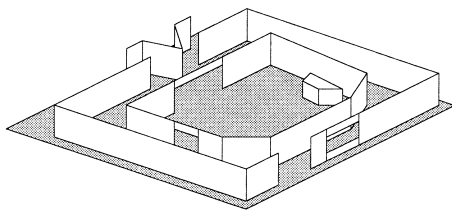
### 3 Classification of building models

Since the complexity of a robust visibility preprocessing program depends on the geometrical characteristics of polygons in a model, we classify architectural models into three classes. In the following definition, we focus on the major occluders, and we assume that the floor and the ceiling are parallel to the  $x$ - $y$  plane, i.e., perpendicular to the  $z$ -axis. First we officially define three types of occluders.

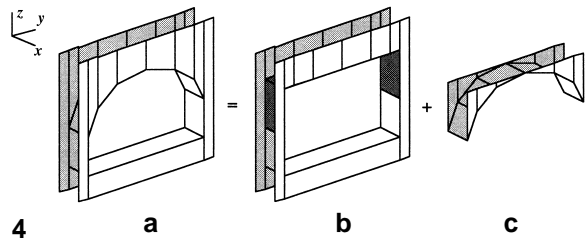
An axial occluder is a rectangle with a normal and edges parallel to the  $x$ ,  $y$  or  $z$ -axis. Since axial rectangles occur so often in architectural models, they are worthy of special treatment.

A semiaxial occluder is not axial, but is a rectangle with edges either parallel or perpendicular to the floor plane (e.g., walls of a pentagonal room).

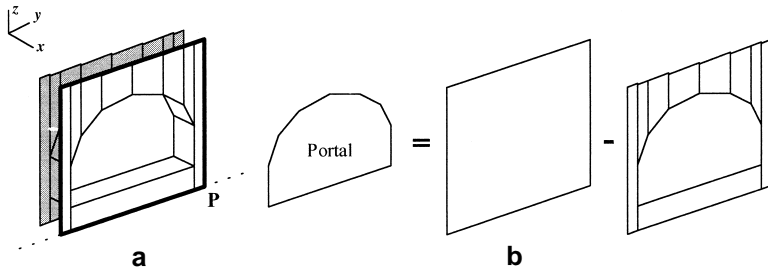
A nonaxial occluder is a generally oriented polygon that is neither axial nor semiaxial.



2



4



3

Fig. 2. A sample of a semi-axial model

Fig. 3a,b. The problem with non-axial models

Fig. 4a-c. An example of the carving operation

From the three types of occluders, we can further define an axial model as one that contains only axial occluders and a semi-axial model as one that contains axial occluders and semi-axial occluders. A non-axial model contains at least one non-axial occluder.

### 3.1 Visibility preprocessing of semi-axial models

The introduction of the semi-axial model is motivated by the observation that, in architectural models, most occluders that are not axial are frequently semi-axial. Figure 2 shows a sample semi-axial model. In this section we briefly discuss extensions of the visibility preprocessing algorithm for the semi-axial model.

The visibility preprocessing for axial models can be enhanced for semi-axial models without much loss of efficiency. When subdivision proceeds along an occluder whose normal is parallel to the  $z$ -axis and whose edges are not parallel to either the  $x$  or  $y$ -axis, the portals are no longer rectilin-

ear. By adjusting the spatial subdivision strategy, we can make the subdivision along such occluders generate no portal. The strategy we use is to make the subdivisions along axial occluders occur first. In other words, axial occluders have a higher priority to be the partition planes on the recursive spatial subdivision. Vertical semi-axial occluders (with normals parallel to the  $x$ - $y$  plane) have second priority. Up to this point, the portals are always rectangles, so the portal enumeration is tractable. Finally, when the model is partitioned along horizontal semi-axial occluders (with normals parallel to the  $z$ -axis), there is no portal, so the portal enumeration can be omitted. We use this trick to handle semi-axial models, and in the following discussion we focus on non-axial models.

### 3.2 Assumption about input

Considering the input of our proposed method, we have to make some basic assumptions. First, we assume that axial polygons occur more frequently than non-axial polygons. Although there is

no critical restriction on the ratio of nonaxial polygons to axial polygons, increasing this ratio in effect increases the number of detail objects. Second, we assume that nonaxial polygons are aggregated in clusters in small numbers, such as less than 100 polygons, and the clusters are distributed roughly evenly in the entire model. If there is a large number of nonaxial polygons in a single cluster, the rendering update rate decreases severely from some view points.

## 4 Carving operation

While there is no conceptual obstacle to extending visibility preprocessing for nonaxial models, the implementation of a robust and practical conforming spatial subdivision is still difficult. Consider the model of a section of wall that has a window (Fig. 3a). When this model is subdivided along the plane  $P$ , the portal on the subdivision plane can be enumerated by a set difference operation on polygons, as demonstrated in Fig. 3b. In other words, the portal enumeration is a subset of the *boolean mask problem* (Lauther 1981), a non-trivial problem in computational geometry, especially for nonaxial polygons. Airey (1990) suggests that portal enumeration requires a robust package for constructive solid geometry operations on planar polygons. One of the attacks on this difficulty is to avoid portal enumeration. Portals are treated as augmented polygons and are embedded directly in the model's representation (Luebke et al. 1995).

This paper presents a different approach to deal with nonaxial models. Instead of searching for a more robust and simpler implementation, we propose a novel idea. The philosophy of the proposed idea is that, given the condition that most of the occluders in a building model are axial, it seems reasonable to look for a method that modifies the model, in a constructive solid geometry manner, by splitting the portion of the model that is not axial. Visibility preprocessing for axial environments can then be used for the modified model. Since this method modifies the models just as a sculptor carves the sculpture, we call it a *carving operation*. This idea can be stated more clearly with the following example.

Figure 4a shows a polygon approximation of an arch window. Obviously this model is nonaxial

since some polygons are in a general position. When the carving operation is applied to the model to remove the arch part, the resulting model (Fig. 4b) is an axial model with a cutting (Fig. 4c). The cutting (Fig. 4c) can be treated as a special detail object (since this object is unmovable during the walkthrough) of (Fig. 4b) in visibility preprocessing. Another point worthy of mentioning in this example is the pasted dark polygons in Fig. 4b. Since the removal of Fig. 4c makes Fig. 4b no longer a 2-manifold (Mäntylä 1984; Requicha et al. 1983), corresponding polygons must be pasted to sew the holes. In other words, the carving operation is a variation of the boolean set *SUBTRACTION* operation in constructive solid geometry.

The carving operation algorithm to be presented can be broken into two parts. First is the automatic determination of the volume that needs to be split, i.e., the subtrahend. Then the model is split by a modified boolean set *SUBTRACTION* operation.

### 4.1 Determination of the carving volume

We first identify the portion of a model that needs to be split, and then we can define the carving volume from the result. In the identification process, the focus is on generally oriented polygons and the grouping of the identified polygons according to edge-sharing information. For clarity in the following discussion, we use the term *axial edge* to denote an edge of a polygon that is parallel to principle axes; otherwise it is a *nonaxial edge*. Since only nonaxial polygons are relevant in the identification stage, at first nonaxial polygons are located and then grouping information is constructed according to the shared edges. Depending on the type of shared edges, two groupings are defined.

A *nonaxial element* (NAE) is the set of non-axial polygons connected by nonaxial edges.

A *nonaxial cluster* (NAC) is the set of connective nonaxial polygons.

Figure 5 depicts the identification process of the model in Fig. 4a. Figure 5a shows the nonaxial polygons of the model. The bold lines represent the nonaxial edges. When only nonaxial edges are

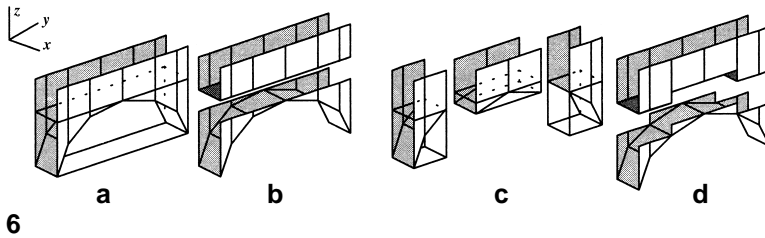
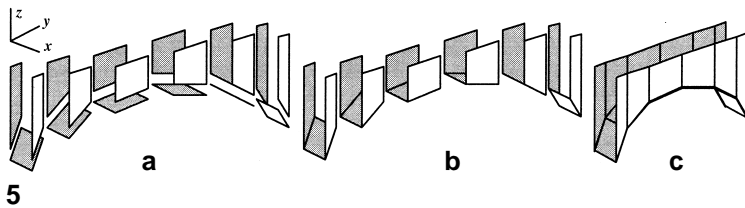


Fig. 5a–c. Polygon classification of the model in Fig. 4

Fig. 6a–d. The scale of carving volume affects the shape of the carved model: a only one carving volume; b three carving volumes

considered, we have six nonaxial elements in Fig. 5b. When all edges are taken into account, there is only one nonaxial cluster, as shown in Fig. 5c.

Once the polygon grouping is constructed, nonaxial polygons are split to remove the nonaxial portion. To do so, first a *bounding box*, called the *carving volume*, along the principal axes is constructed for the nonaxial edges of the nonaxial cluster. Then a boolean set *SUBTRACTION* operation is performed on the model; the carving is used as a subtrahend. Figure 6a and b shows the carving processing of the model in Fig. 5. It can be verified that the carving volume determined in this step ensures that the residual polygons of the carving operation are axial, and we omit the detail of proof in this paper.

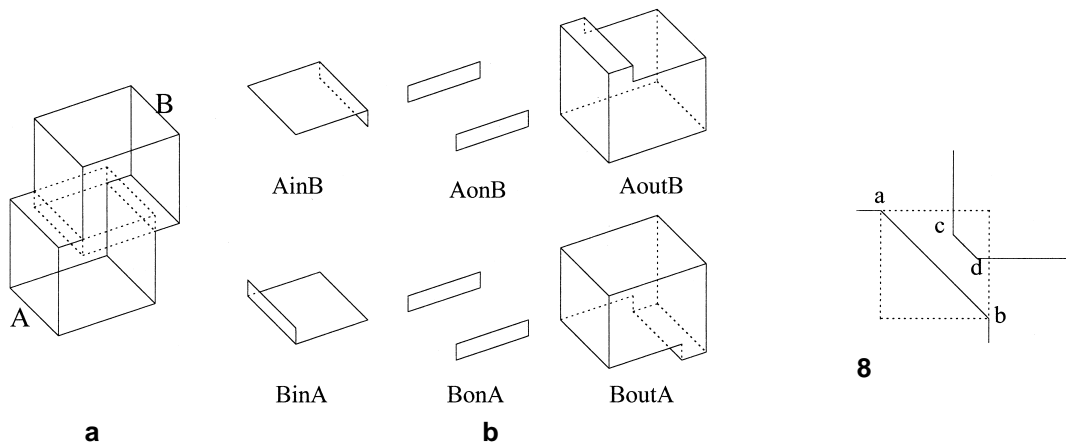
Since using the nonaxial cluster to determine the carving volume (hereafter we call a set of such polygons a carving unit) of the *SUBTRACTION* operation generates a larger portal, we can use nonaxial elements and their combinations as a carving unit. In short, the nonaxial element and the nonaxial cluster define the scope of the carving unit, where “nonaxial element” stands for the smallest carving unit, and no polygon in a nonaxial element can be considered individually. For example, the three polygons in each nonaxial element in Fig. 5b must be processed simultaneously. In contrast, a nonaxial cluster is the largest carving unit. Between the scale of

a nonaxial element and a nonaxial cluster, any nonaxial elements that are connected by edges can be used as a carving unit. According to this rule, any segments of adjacent nonaxial elements in Fig. 5b can be combined as a carving unit. When six nonaxial elements are linked together, the result is in fact the nonaxial cluster. The algorithm for determining the carving volume automatically is given in Sect. 4.4.

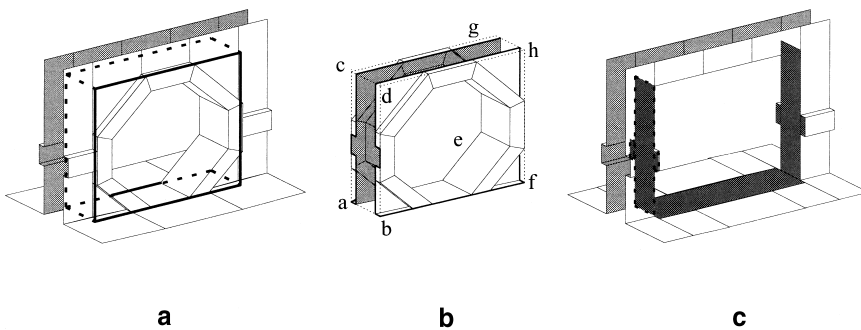
Figure 6 shows how the combination of nonaxial elements affect the convexity and size of the portals. When the nonaxial cluster is used as a carving unit, as in Fig. 6a, there is only one resulting portal (Fig. 6b). On the contrary, if the six nonaxial elements are divided into three carving units, as shown in Fig. 6c, the resulting concave portal must be decomposed into at least two portals, but with smaller areas (Fig. 6d). It is a trade-off between the portal number and the portal area. In our implementation, we prefer to minimize the number of portals in order to minimize stab-tree traversal time, so only one heuristic is used, making the carving unit as large as possible.

## 4.2 Carving

The carving operation is implemented as a simplified boolean set *SUBTRACTION* operation, since one of the operands is convex axial object.



7



9

Fig. 7a, b. Boundary classification of two intersecting blocks

Fig. 8. The difference between carving and *SUBTRACTION*

Fig. 9a-c. Computing  $BinA$  directly from the cutting

We use the idea of boundary classification from Mäntylä (1986). Consider the splitting of a polyhedron  $A$  according to a reference polyhedron  $B$ : the three objects resulting from the splitting operation are denoted by  $A_{inB}$  (for the part of  $A$  inside  $B$ ),  $A_{onB}$  (for the part of  $A$  on the boundary of  $B$ ) and  $A_{outB}$  (for the part of  $A$  outside  $B$ ). Given two polyhedra  $A$  and  $B$ , the collection of six objects  $A_{inB}$ ,  $A_{onB}$ ,  $A_{outB}$ ,  $B_{inA}$ ,  $B_{onA}$ , and  $B_{outA}$  formed by splitting  $A$  against  $B$  and symmetrically  $B$  against  $A$  is called the *boundary classification* of  $A$  and  $B$ . For example, Fig. 7a depicts two intersecting blocks with coplanar front and back polygons (the intersection region is indicated with dashed lines). The boundary classification of the two blocks is shown in Fig. 7b.

Let us set the model as  $A$  and the carving volume as  $B$ . From the boundary classification of  $A$  and  $B$ , the carving operation is readily computed:

$$A \text{ carve } B = A_{outB} \& (BinA)^{-1}, \quad (1)$$

$$Cutting = A_{inB} \& A_{onB} = A - A_{outB}, \quad (2)$$

where “&” denotes the “gluing” of two boundary surfaces, “-” denotes the difference of two surfaces, and “ $( )^{-1}$ ” denotes inverting the normal vector of all polygons of the surface. Since the cutting is the complement of  $A_{outB}$  with respect to  $A$ , we are only concerned with the computation of  $A_{outB}$  and  $BinA$ .

### 4.2.1 Computing $A_{out}B$

To compute  $A_{out}B$ , the first step is to split the model so that the polygons in the model do not intersect the polygons of the carving volume. Since the carving volume is always a rectangular box, the splitting can be implemented as a series of spatial partitions along the six polygons of the box, i.e., the bounding polygons of the carving volume. Once the polygons in  $A$  have been split, each polygon in  $A$  is classified as lying OUTSIDE of carving volume or OTHERWISE (in our implementation, we do not classify  $A_{in}B$  and  $A_{on}B$ ). Again, since object  $B$  is convex, this classifying is straightforward and uses the normal vectors of bounding polygons.

While this method is similar to a typical implementation of a boolean set operation, there is a major difference: the splitting should be controlled in the local region. For example, Fig. 8 depicts a two-dimensional case in which segment  $ab$  is carved and the carving volume is indicated with a dashed box. Although segment  $cd$  and the other two segments intersect the carving volume, these three segments should not be split. To confine splitting to the local region, the polygons that are taken into consideration must form a single *connected component*, starting from a polygon of the carving unit (the set of polygons that define carving volume), each polygon of which has a nonempty intersection with carving volume.

Assume that it can be recognized whether a polygon belongs to the carving unit. The procedure *ComputeAoutB*( ) listed here computes  $A_{out}B$  with model  $M$  as object  $A$  and carving volume  $CV$  as object  $B$  is presented. In this procedure, the normal vectors of the carving volume are assumed to face inward.

**procedure** *ComputeAoutB*

(Input: model  $M$ , carving volume  $CV$ )

Output:  $A_{out}B$ , cutting  $C$ )

Move polygons in  $M$  that belongs to carving unit to *PolyPool*

$C = \emptyset$ ,  $A_{out}B = \emptyset$

**for each** *polygon* **in** *PolyPool* **do**

**if** *polygon* and *polyhedron* do INTERSECT **then**

$A_{out}B = A_{out}B \cup (\text{polygon} \cap CV)$

        // put axial portion to  $A_{out}B$

$C = C \cup (\text{polygon} \cap CV^+)$

        // put nonaxial portion to  $C$   
        Move polygons in  $M$  that share edge with *polyhedron* to *PolyPool*  
    **else if** *polyhedron*  $\subset CV^-$  **then**  
         $A_{out}B = A_{out}B \cup \text{polyhedron}$   
        // put axial portion to  $A_{out}B$   
    **else**  
         $C = C \cup \text{polyhedron}$   
        Move polygons in  $M$  that share edge with *polyhedron* to *PolyPool*  
    **end if**  
**end for**  
 $A_{out}B = A_{out}B \cup M$   
// the rest polygons must be  $A_{out}B$

### 4.2.2 Computing $BinA$

For  $BinA$ , an approach different from splitting is employed. If we try to split  $B$  (the carving volume) so that the polygons in  $B$  do not intersect the polygons in  $A$  (the model), since there are more than six polygons and the arrangement of polygons is not predefined, the program would be time consuming and complicated. Instead,  $BinA$  is computed directly from the cuttings. We introduce the concept of external edges and external polygons. We say an edge in a cutting is *external* if it is owned by only one polygon. Since dangling edges only occur along a split plane for a 2-manifold polyhedron, it is obvious that external edges must lie on the bounding polygons of carving volume. For example, Fig. 9a depicts a model of the octagonal arch in the upper right of Fig. 1 where the bold line box represents the carving volume. This carving operation generates the cutting shown in Fig. 9b and the axial model shown in Fig. 9c. (The external edges are indicated in bold lines in Fig. 9b.) Since  $BinA$  is generated from the bounding polygons, the computation of  $BinA$  can be reduced to the two-dimensional boolean mask problem on the bounding polygons as follows.

For each bounding polygon, if there are any external edges that lie within the interior of the bounding polygon, such as the bounding polygon  $abcd$  in Fig. 9b,  $BinA$  on the bounding polygon are either the polygons generated from this external edges chain (together with part of the edges of the bounding polygon) or the complement of the generated polygons, depending on the orientation



of the vertices in the external edges chain. The *BinA* polygons generated from bounding polygon *abcd* are shown as dashed lines in Fig. 9c. However, if all external edges lie on the boundary of the bounding polygon (the bounding polygon *abfe* in Fig. 9b), or there is no external edge lying on the bounding polygon, the bounding polygon is a *BinA* polygon, or there is no *BinA* on this bounding polygon. The three *BinA* type polygons in Fig. 4b (the darkened polygons) belong to this category. The decision depends on whether the bounding polygon is in the interior or exterior of the model, and a routine based on raycasting (Laidlaw and Hughes 1986) is used. As a result, *BinA* on a bounding polygon (BP) with cutting *C* can be found with the procedure:

```

procedure ComputeBinA
(Input: bounding polygon BP, cutting C
Output: BinA)
if external edges of C lie in the interior of BP
then
  E = external edges of C lying in the interior of
  BP// Find associated edges
  P = polygons generated from E
  // A edges connectivity traversal will
  // find the polygons set
  nv = normal of P
  if nv · normal of BP = 1 then
    BinA = P
  else
    BinA = BP - P
  end if
else
  Shoot a ray from center of BP in the direction
  of normal vector (inward)
  Find the first intersecting polygon polygon in C
  If no such polygon found then
    BinA =  $\emptyset$ 
  else
    if the center of BP does not lie on the plane of
    polygon then
      if the normal of polygon points toward BP
      then
        BinA =  $\emptyset$  // this BP is outside the model
      else
        BinA = BP
      end if
    else
      if the normal of polygon points outward
      then

```

```

        BinA =  $\emptyset$  // this BP is outside the model
      else
        BinA = BP
      end if
    end if
  end if
end if

```

#### 4.4 Pseudo-code

To sum up the discussion in the preceding section, we present a filter procedure *CarveIt*( ) that receives a model and outputs an axial model and a set of cuttings:

```

procedure CarveIt
(Input: model M
Output: axial model A, cuttings C)

//Step 1. Determine carving volume
Classify polygons in M as axial set I and generally
oriented set G
Construct edges connectivity information CI of G
According to connectivity information CI, group-
ing nonaxial clusters set NACluster
and nonaxial elements set NAElement of each
nonaxial cluster
C =  $\emptyset$  // cuttings set
P =  $\emptyset$  // BinA set

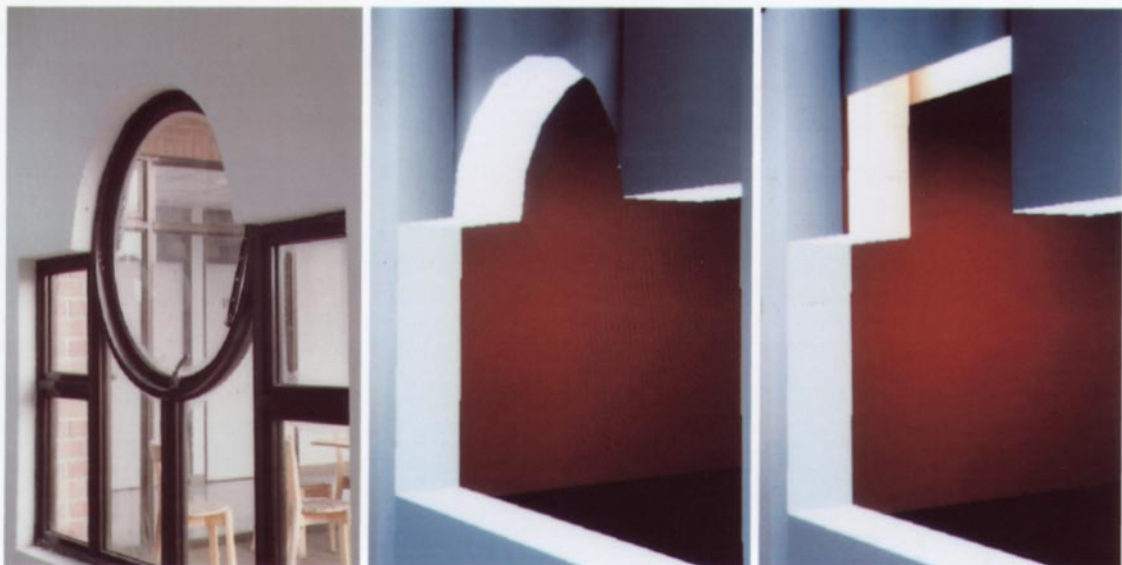
//Step 2. Carving
for each nonaxial cluster nacluster in NACluster
do
  Determine carving unit set CU of nacluster
  for each carving unit cu in CU do
    Construct carving volume CV
    ComputeAoutB(M, CV) to obtain output
    AoutB and c
    M = AoutB
    C = C  $\cup$  c
    for each bounding polygon bp of carving vol-
    ume CV do
      ComputeBinA(bp, c) to obtain output BinA
    end for
    P = P  $\cup$  BinA
  end for
end for
A = M  $\cup$  P

```



10a

b



11a

b

c



12

Fig. 10. a The corridor. b A snapshot of the walkthrough system

Fig. 11. a Close view of a arch window. b Rendered image of the arch window. c The same arch window without cutting

Fig. 12. The equipment of the SpaceWalker interactive building walkthrough system

## 5 Results

In this section, several practical applications of the carving operation in our interactive building walkthrough system are presented. The target environment is our new department building. Currently we have constructed the major structure (without furnishings) of the fifth floor in AutoCad. The original model comprises 3104 polygons, of which there are eight nonaxial clusters, the eight arch windows in the hallway. The software of the walkthrough system, mainly divided into four modules – format converter, conforming BSP tree constructor, static and dynamic visibility processor and virtual environment effect editor – are all designed from scratch.

In the visibility preprocessing phase, we implemented the carving-based visibility preprocessing program on a SUN SparcStation10 Model 41. The carving operation requires less than 3 CPU s, and results in 3040 axial polygons, 24 BinA type polygons and 192 nonaxial polygons belonging to eight special detail objects. Constructing the BSP tree requires about 20 s. The resulting BSP tree contains 1249 internal cells and 1250 external cells, and it increases the model size to 3531 polygons. The visibility preprocessing takes about 1900 s. At the last stage, the polygons are diced into 40 180 triangles and run through a radiosity computation program.

In the walkthrough phase, our interactive walkthrough system was developed on an SGI Indigo<sup>2</sup> Extreme graphics workstation. Since the system is still being developed, we have not assessed the performance precisely. Currently, the average rendering frame rate is about 20 frames/s in 640 × 480 window resolution. A view of the corridor on the fifth floor of the real building is shown in Fig. 10a. Figure 10b shows a snapshot of the walkthrough system in a similar view direction. From this view direction we can also see the arch shaped windows that are not axial. Figure 11a shows a close view of one arch window. Figure 11b shows a rendered image of the arch window. Figure 11c shows the same arch window without the cutting.

Figure 12 shows a user immersed in our interactive walkthrough system. The user wears a head-mounted display and walks on a treadmill to simulate the perception of navigating through a building. Two buttons attached on the handle of the treadmill are used for controlling the walking

direction, and one shaft encoder linked to the computer reports the walking speed.

## 6 Conclusion

We have described a novel operation for visibility preprocessing of unrestricted (semiaxial and nonaxial) three-dimensional building environments. We have demonstrated the application of the proposed carving operation in a complex environment. The carving operation was used in an interactive walkthrough system based on our new computer science department building; within the building there are eight arch windows that are not axial in each floor.

## References

1. Airey JM (1990) Increasing update rates in the building walkthrough system with automatic model-space subdivision and potentially visible set calculations. PhD Thesis, Department of Computer Science, University of North Carolina, Chapel Hill, NC
2. Airey JM, Rohlf JH, Brooks FP Jr. (1990) Towards image realism with interactive update rates in complex virtual building environments. ACM SIGGRAPH Special Issue 1990 Symposium on Interactive 3D Graphics 24:41–50
3. Brooks FP Jr. (1986) Walkthrough – a dynamic graphics system for simulating virtual buildings. Proceedings of the 1986 Workshop on Interactive Computer Graphics, Chapel Hill, NC, pp 9–20
4. Fuchs H, Kedem Z, Naylor B (1980) On visible surface generation by a priori tree structures. Comput Graph (Proceedings of SIGGRAPH'80) 14:124–133
5. Hohmeyer ME, Teller SJ (1992) Stabbing isothetic rectangles and boxes in  $O(n \lg n)$  time. Comput Geom Theory Appl 4:201–207
6. Laidlaw DH, Hughes JF (1986) Constructive solid geometry for polyhedral objects. Comput Graph (Proceedings of SIGGRAPH'86) 20:161–170
7. Lauther U (1981) An  $(N \log N)$  algorithm for boolean mask operations. Proceedings of the 18th Design Automation Conference, Nashville, IEEE, pp 555–562
8. Lei YW (1997) The SpaceWalker walkthrough system for unrestricted three-dimensional polygon environments. PhD Thesis, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan
9. Luebke D, Georges C (1995) Portals and mirrors: simple, fast evaluation of potentially visible sets. Proceedings of the 1995 Symposium on Interactive 3D Graphics, Monterey, CA, ACM SIGGRAPH, pp 105–106
10. Mäntylä M (1984) A note on the modeling space of Euler operators. Comput Vision Graph Image Process 26:45–60
11. Mäntylä M (1986) Boolean operations of 2-manifolds through vertex neighborhood classification. ACM Trans Graph 5:1–29

12. Pellegrini M (1990) Stabbing and ray-shooting in 3-dimensional space. Proceedings of the 6th ACM Symposium on Computational Geometry, Berkeley, CA, pp 177–186
13. Requicha AAG, Voelcker HB (1983) Solid modeling: current status and research directions. IEEE Comput Graph Appl 3:25–37
14. Teller SJ (1992a) Computing the antipenumbra cast by an area light source. Comput Graph (Proceedings of SIGGRAPH'92) 26:139–148
15. Teller SJ (1992b) Visibility computations in densely occluded polyhedral environments. PhD Thesis, Computer Science Department, University of California at Berkeley, Berkeley, CA
16. Teller SJ, Hanrahan P (1993) Global visibility algorithms for illumination computations. Comput Graph (Proceedings of SIGGRAPH'93) pp 239–246
17. Teller SJ, Séquin CH (1991) Visibility preprocessing for interactive walkthroughs. Comput Graph (Proceedings of SIGGRAPH'91) 25:61–69



**YUONG-WEI LEI** received his BS and PhD degrees in Computer Science and Information Engineering from the National Taiwan University, Taipei, in 1991 and 1997, respectively. He is currently on the research staff in the Communications and Multimedia Laboratory. His research interests include computer graphics, computer animation, model-based image coding, graphical user interfaces and multimedia systems.



**MING OUHYOUNG** received his BS and MS degrees in Electrical Engineering from the National Taiwan University, Taipei, in 1981 and 1985, respectively. He received his PhD degree in Computer Science from the University of North Carolina at Chapel Hill in 1990. He was a member of the technical staff at AT&T Bell Laboratories, Middletown, during 1990 and 1991. Since August 1991, he has been an Associate Professor in the Computer Science and Information Engineering Department, National

Taiwan University. He has published more than 40 technical papers on computer graphics, virtual reality, and multimedia systems. He is a member of the ACM and IEEE.