WEB CLUSTERING: A NEW APPROACH TO SPACE PARTITIONING

A. James and A.M. Day

School of Information Systems, University of East Anglia,
University Plain, Norwich, Norfolk, NR4 7TJ. England.
aj@sys.uea.ac.uk, amd@sys.uea.ac.uk

ABSTRACT

The Binary Space Partitioning Tree has provided a sound structure on which to build rendering algorithms since the early eighties. In this paper, we manipulate the BSP tree in a new way, and subsequently morph the algorithm and data structure resulting in a new, very different, technique. Our algorithm is restricted to two-dimensions in this paper and thus our input is line segments; our aim is to decrease the line segment splitting associated with BSP trees. We present the Web Clustering algorithm with a run-time competitive to Binary Space Partitioning in random scenes and better with structures that incorporate empty passageways or 'void' regions. We show how buildings, for example, can be divided by their corridors with very few splits. Our method changes a fundamental part of BSP generation, leading to the abolition of the binary restriction. This allows us to branch our partitioning of the environment where a BSP partition would slice through the scene, regardless of obstacles. In terms of the overall rendering times, the algorithm reduces line segment splitting compared to the BSP technique. Preprocessing is also attractive since it exploits the characteristics of structured scenes and is thus a faster divide and conquer technique.

Keywords: binary-space partitioning, hidden-surface removal, portals, scene clustering

1 Introduction

Binary space partitioning [Fuchs79, Fuchs80, Fuchs83] has several disadvantages. Splitting is usually necessary and leads to an increased number of line segment fragments in the (two-dimensional) scene. This has been explored significantly since BSP trees were first developed. A less explored fact is that of *binary* partitioning which, although a divide and conquer technique, is limited to (at best) *halving* the 'problems' at each stage so preprocessing progress is slow. Further, due to the binary restriction, the algorithm cannot exploit the common properties of buildings such as corridors and portals. For example, a corridor is free from potential conflicts so might be used to accommodate a divider, but if such a corridor branches, the divider will continue relentlessly into the wall and into subsequent 'busy' space.

Although our input and environment consist of line segments, our algorithm is applicable to walk-throughs of environments comprising vertical polygons or where a height restriction is used [James98c, James98b]. In this paper, we exploit dramatically the fact that corridors are mostly 'empty' and in fact, by identifying such spaces, use the corridors as the initial (top level) partitioning skeletons. We remove the binary restriction and thus are able to branch our partitioning structure when its underlying skeleton corridor branches. As such, our divide and conquer algorithm takes fewer iterations or recursions to process the scene due to the subdivision of space into several (maybe tens of) spaces, and avoids the aforementioned 'busy' space. We also show how Web Clustering can be used to subdivide random scenes without such corridors and even solve priority cycles where binary space partitioning fails.

A new algorithm, data structure and traversal are presented at the two-dimensional level reducing facet splitting (associated with the dividers) to the unit percentages. We compare our algorithm to the auxiliary BSP tree [Schum69] since this provides the closest and fairest comparisons.

Our tests are performed on models of buildings created by a semi-automatic building generator. Such buildings comprise a set of line segments (representing walls) which are arranged to form corridors (long empty regions within the building) and rooms (small, closed empty regions adjoining a corridor and surrounded by walls) - see figures 11 to 16. On testing, our rooms are filled with random line segments to simulate room furniture and fittings, etc.

The data structure, the 'Web Graph', is a multi-level di-graph whose directed arcs are dependent upon the location of the observer but, as in binary space partitioning, can use line segment dependencies [James98c] to reduce the number of necessary tests. In this paper, our technique for dividing line selection is relatively basic, but many improvements are likely to be explored in the future, including the standard BSP techniques and conflict minimization [Fuchs79] or conflict neutralization [James98a, James98d].

In section 2 we give an overview of Binary Space Partitioning and its Auxiliary variation. We then develop Auxiliary Binary Space Partitioning into our Web Clustering algorithm in section 3. The algorithm is outlined and described in section 4 using non-structured scenes and then the fundamental corridor enhancement is described in section 5. Testing is presented in section 6 and conclusions in section 7. We discuss some future enhancements in the final section.

2 Binary Space Partitioning

At each stage in the (two-dimensional) Binary Space Partitioning [Fuchs79, Fuchs80, Fuchs83] algorithm, a partitioning line is chosen which subdivides the scene into two parts, line segments in each part being assigned to the appropriate child of the root. Splitting occurs if the divider passes through any segment, thus allowing each part to be assigned to the appropriate child. Each subtree is then subdivided similarly in a recursive procedure until all nodes are atomic.

Auxiliary Binary Space Partitioning varies only in the set of candidate dividing lines. Instead of using the line segments in the input data, we can use any arbitrary line. However, the selection of an auxiliary line is complex without certain rules, restrictions, or guidelines. We have adopted the limitation that auxiliary lines can either be generated by joining line segment end points (Figure 1) or by using predetermined points such as corridor junctions. In the former case, the restriction reduces the set of auxiliary lines to $O(n^2)$ for the n line segments in each sub-scene (and we only examine a subset of these in our implementation). In the latter case the auxiliary lines are linearly related to the number of corridors in the building.

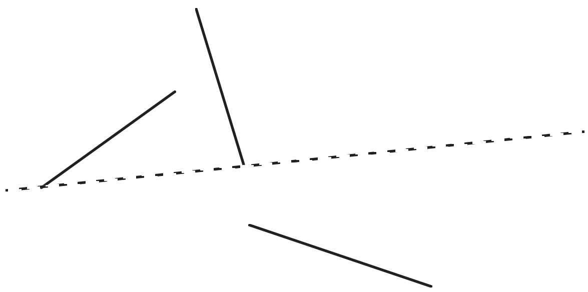

Figure 1: An end-point Auxiliary Binary Partition

3 Web Clustering

We define Web Clustering as the process of building a Web Cluster which is represented by a Web Graph. Web Clustering is achieved by subdividing a scene by a set of connected line segments resulting in a set of convex areas (exhibiting non-cyclic observer dependent priorities) each of which surrounds either a single line segment (possibly a fragment) or another Web Cluster. Thus, the Web Graph can be traversed in such a way as to sort the (sub)convex areas into priority order.

Web Clustering is achieved by exploiting an important property of binary partitions: The selected divider (line segment defined by the dividing line's intersection with the scene's convex hull) may, for a large portion of its length, cut no other line segment (Figure 2). It would be desirable to keep the 'good part' of the line segment divider, while possibly rejecting the remainder that is involved in conflicts. This can be achieved by bending the line segment away from obstructions. The line segment may then continue, or bend again if necessary. However, this winding path cannot be used to subdivide the space since some of its resulting areas are now concave and do not conform to the properties of a conventional partition, i.e. there exist no valid priority orders for the areas. Figure 3 illustrates.

To remedy the problem, each line segment in the path must be partnered by a sibling line which maintains the convexity of the division. Therefore the angle between each pair of lines at the tri-junction must be non-reflex. Figure 4 illustrates.

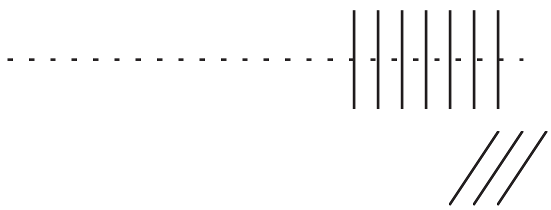

Figure 2: A (poor) Binary Partition

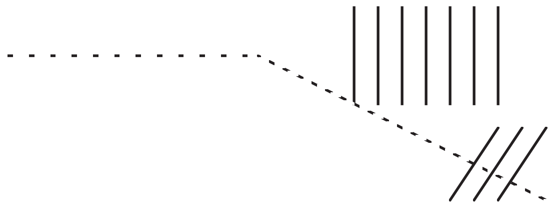

Figure 3: An avoiding bend in the path

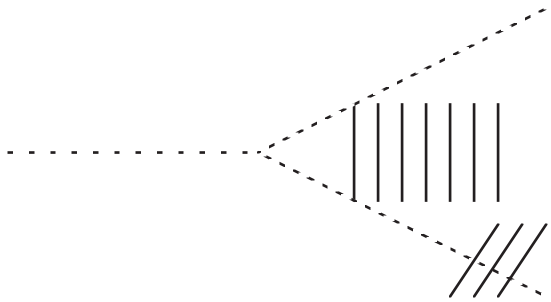

Figure 4: The sibling to the bend

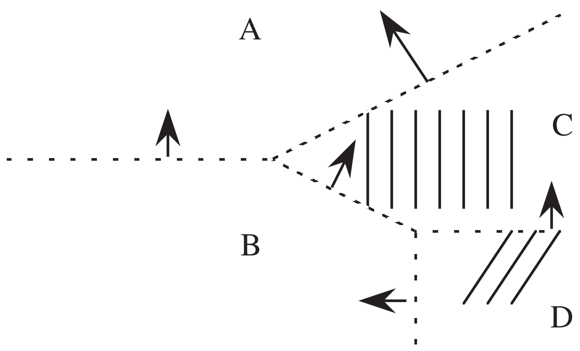

Figure 5: A first level Web Partition

Both sibling lines partnering the tri-junction then continue independently, branching again if necessary. Figure 5 shows the final first level web (created by arbitrary line segments). Subsequently, each convex area of the web would be divided similarly.

Each area has been labeled in Figure 5 and each divider is marked to face away from the observer assuming he is located in the bottom-right corner of area D. To obtain a priority ordering for the areas, we can select an arbitrary area and see if any arrow leaves such an area, if so we follow it to the next area. When no such arrow exists, our current area has the highest priority, and the contents can be drawn. The arrows adjoining this area can now be made void, and we can backtrack to the former area (or an adjoining area if no former area exists); the side tests can then be run again. The algorithm is simply a representation of the BSP property and its correctness thus follows.

To form the associated Web Graph, each area is represented by a node and each node is connected to another node if and only if there is a single divider between the two areas thus forming the dual of the web. The graph can be traversed in the same way as the web by following directed arcs (see Figure 6).

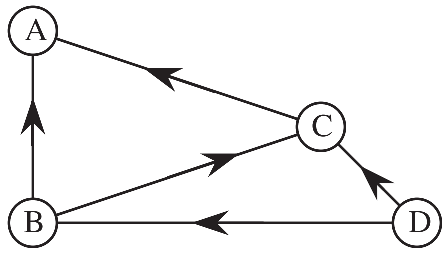

Figure 6: The graph representing the web

3.1 Cycles

The algorithm can be used to solve the problem of cyclic conflicts such as the fundamental trio of line segments shown in Figure 7. Although in this paper we restrict the selection of auxiliary lines to those attached to candidate *end points* (or predetermined line segments), we could divide the trio by a point in the centre leading to no cuts.

Sometimes splitting is absolutely necessary (even if only implicitly) in *all* rendering algorithms since from some view points, line segments may cyclically obscure each other. Consider the sequence of figures 7, 8 and 9. As the line segments

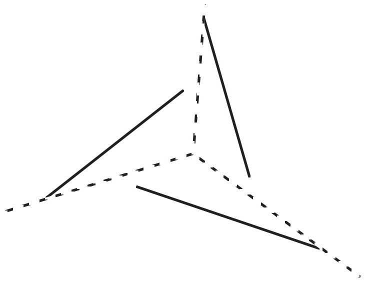

Figure 7: Cycle solved with leeway

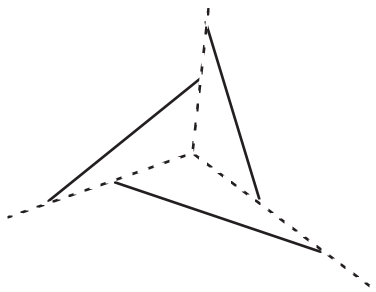

Figure 8: Cycle solved precisely

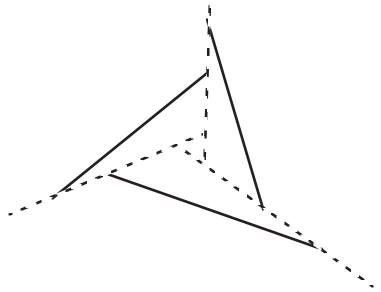

Figure 9: Cycle unsolvable

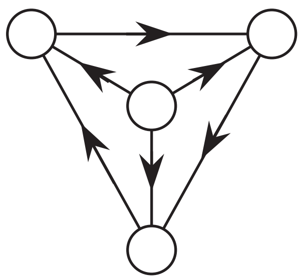

Figure 10: The graph with cycle

converge, the range of possible tri-junctions reduces until eventually there exist no valid tri-junctions: In Figure 7, there is a range of tri-junctions that allows a perfect partition (one that results with no cuts or conflicts); in Figure 8 the range is reduced to a single point which sees all segments touch but not overlap; and in Figure 9 there is a void area which sees all segments obscuring each other. Our structure *appears* to solve even this unsolvable problem since we have been able to draw a web which does not intersect the segments. However, the web itself contains a cycle which is reflected in the accompanying graph (Figure 10) assuming the observer is positioned in the center – a traversal would enter an infinite loop since no node is a terminal one. Cycles are therefore avoided, and this restriction is adhered to by disallowing branches to touch other branches (excluding their immediate sibling and ancestor branches).

4 Algorithm

In this section of the paper, we restrict dividing lines to those formed by attaching line segment end points together. Later we shall describe how corridors are used to predetermine potential dividing lines.

4.1 Outline

The algorithm is initialized by selecting a binary partition created from a peripheral end point to any other. If the partition is perfect, then a simple graph is created – two connected nodes. Otherwise, we need to examine whether the partitioning line can be branched. In our most basic algorithm, this is done be accepting the line segment part of the partitioning line as 'good' and its continuation as (possibly) 'bad'. The set of candidate branching lines formed by our current anchor (the non-peripheral end point) and a new set of end points is examined. Each valid pair (i.e., a pair that creates convex areas and does not cross other branches) that cuts fewer line segments than the parent line is a candidate. The candidate pair that creates fewest cuts is chosen if the sum of its cuts is no more than the parent. These sibling lines create a third area which is added as a node and connected to its two parent nodes.

The contents of the new node may inherit segments from its parents. Each parent segment is examined to see if it lies (maybe partially) in the new area, and if so, moved to the new node (after being split if necessary). Initially, line segments may have been split by the binary partition, and one part assigned to each node. There may now be parts of the same line segment residing in the child node; these can be rejoined since the cut has been disabled and thus the damage can be undone.

The rejoining of 'old cuts' can be made efficient by searching through the list in the child node for segments with the same 'name'. Alternatively, segments can be rejoined if they share an end point and are co-linear.

The branches are divided again recursively, if necessary, until they exit the convex hull of the scene. Each node is then processed recursively until all nodes are atomic.

The ultimate graph has a set of connected nodes, each of which contains a line segment, or another Web Graph. In the case of a nested graph, upon traversal, it is fully explored when the parent node is ready to be 'displayed'. As in our earlier example, the direction of each arc is determined by testing its corresponding divider against the location of the observer.

4.2 Data Structure

Each node in the Web Graph comprises a line segment pointer (which is null if it has a child graph), a list of neighbours, a pointer to a child graph (possibly null) and a temporary variable which tags a node as visited. The list of neighbours comprises a pointer to each node, an arc label for the side test, the independent direction (representing the normal to the segment) and the run-time observer dependent direction. Finally there exists a pointer to the next neighbour.

4.3 Pseudocode

Pseudocode for the algorithm is shown below; it has been simplified while maintaining most of the important information. Initialization and error checking have been omitted since they are 'intuitive'. We present the more complex algorithm which adopts the end point selection criterion and ignores the potential of corridor branches. (To implement corridor branching, the function *triJunct* would select branches from the corridor data set.) Our algorithm here simplifies the tri-junction rule (section 3) by allowing only forward facing partitioning arms. This means that the probability of cycling branches is reduced and allows efficient and independent selection of the branching pair.

Three pseudo-functions are shown: *webGroup* creates a web of webs and is the root function called to create a Web Graph; *webSpin* initialises (by a binary partition) and grows a single web (by calling *triDivide*); *triDivide* is passed a starting line and forms branches if this option is inexpensive otherwise the starting line is terminal. The omitted function *init* initialises the neighbour relationship explained in the previous paragraph but the run-time observer dependent direction is omitted since it can only be calculated at run-time.

```
Web_graph webGroup(lol segs)
{
  Web_graph  Web;  //root of the web graph
  lown    temp;    //used to create sub-webs

  if (segs->next==NULL) return head(segs); //atom
  else
  {
    Web=webSpin(segs);      //create one web

    temp=Web->nodelist;
    while (temp!=NULL)      //create sub-webs
    {
      temp->node->child=
        webGroup(temp->node->segs);
      temp=temp->next;
    }
  }
  return Web;
}

Web_graph webSpin(lol segs)
{
  Web_graph  W, //W is the root node
             N; //N is the brother
  seg_Rec    D; //division/partition
  int cont, //cost of partition terminating
       tot;  //cost of partition branching

  //create root node W
  W->segs=segs;

  //create first (binary) partition
  division(W->segs, &cont, &tot, &D);

  //create N, brother to W - each reps a half space

  //create neighbour relationship between N and W
  init(W->nbrs,N,D,1);
  init(N->nbrs,W,D,-1);

  //distribute segments between W and N
  distSegs(&(W->segs),&(N->segs),D,1);

  triDivide(W,N,D,cont); //Grow web

  return W;
}

void triDivide(Web_graph N1, Web_graph N2,
                  seg_Rec D, int lastcont)
{
  //branches are grown from D which partitions N1,N2
```

```
  //if it is cheaper than lastcont cost

  seg_Rec Sa, Sb;
  Web_graph N;
  int contA, contB, totA, totB; //branch costs

  if ((N1!=NULL) && (N2!=NULL))
  {
    //calls to triJunct create each branch
    triJunct(N1->segs, D, &Sa, &contA, &totA);
    triJunct(N2->segs, D, &Sb, &contB, &totB);

    //see if branches collide (disallowed)
    //of whether terminating is cheaper
    if ((touch(Sa,Sb)) || ((lastcont)<(totA+totB)))
        exit;

    //create brother N to two parents (areas)

    //attach brother to both parents and vice versa
    init(N1->nbrs,N,Sa,1);
    init(N->nbrs,N1,Sa,-1);
    init(N2->nbrs,N,Sb,-1);
    init(N->nbrs,N2,Sb,1);

    distSegs(&(N1->segs),&(N->segs),Sa,1);
    distSegs(&(N2->segs),&(N->segs),Sb,-1);

    //rejoin cuts that have been disabled
    rejoinSegs(&(N->segs));

    //Branch further
    triDivide(N1,N,Sa,contA);
    triDivide(N,N2,Sb,contB);
  }
}
```

5 Corridors

Perhaps the most beneficial part of the algorithm is how our dividers can be assigned to corridors at the initial stages of subdivision. Only when building blocks (a set of rooms), or even single rooms remain does the former description (of using end points) come into effect. As such, the initial subdivisions create no cuts (assuming corridors are empty) and the smaller problems of individual rooms remain.

This significant addition to our aforementioned algorithm is currently implemented by human intervention. The user identifies corridors and overlays the desired web. This is also performed on descendent areas if necessary. The task may be automated by tagging corridors and selecting dividers from this set alone. Only when these have been exhausted are non tagged end points used.

6 Results

Using a semi-automatic building generator, we have tested our algorithm with many types of building. Each building has several rooms in which random line segments are placed to simulate furniture. (This test data thus enables us to examine the performance of structured and unstructured line segments.) Table 1 shows the characteristics of the data presented (without 'furniture') in figures 11 to 16, which are: a simple scene; an orthogonal scene suitable for binary partitioning; a branching scene more suitable for Web Clustering; a complex scenes of multiple branches; a cyclic scene to illustrate the problem of graph cycles; and a multiply cyclic scene. Tables 2 and 3 show for each building the preprocessing times and number of splits, respectively, of auxiliary Binary Space Partitioning, Web Clustering and Web Clustering using corridor skeletons.

Desc	Segments	Corrs	Rooms
Simple	3000	4	116
Orthog	5000	13	98
Branching	5000	5	157
Multi-Branch	8000	15	264
Cyclic	7000	10	207
Multi-Cyclic	10000	20	172

Table 1: Data characteristics

Desc	Aux	Web	Corri
Simple	355	353	304
Orthog	880	869	743
Branching	937	886	730
Multi-Branch	1540	1582	1390
Cyclic	1704	1737	1178
Multi-Cyclic	2596	2594	1860

Table 2: Preprocessing times

Desc	Aux	Web	Corri
Simple	144	141	144
Orthog	358	356	315
Branching	260	240	220
Multi-Branch	560	547	463
Cyclic	361	370	324
Multi-Cyclic	762	718	589

Table 3: Splits (affecting run-times)

We can clearly see that Web Clustering has a marginally faster preprocessing time than Auxiliary Binary Space Partitioning, but utilising corridors, it is much faster and creates fewer cuts. Although both preprocessing and run-times are not improved by several magnitude by our new algorithm, we were encouraged to witness the rendering of the scene with Corridor Web Clustering because each building block was drawn separately. Thus we plan to improve upon the algorithm by several magnitudes by utilising portal-like criteria (described further in section 8). In fact, early estimates show that the portal version of Web Clustering may dramatically reduce overdraw and in a similar fashion to existing portal algorithms [James98b] [Telle91] [Telle93] [Gordo91] [Green96] [Aliag97] achieve output sensitivity.

7 Conclusions

Our initial aim was to decrease the splits associated with BSP trees, and this has been achieved. We have presented an algorithm with a run-time competitive to Binary Space Partitioning in random scenes, which performs well with structures that incorporate corridors. In the latter case, we have shown how buildings, for example, can be divided by their corridors into blocks, and by walls into rooms, with very few splits. The room contents can then be subdivided by the former case, where we have shown how the algorithm can solve some of the priority cycles which require splitting with binary partitioning.

In terms of the overall rendering times, the algorithm reduces line segment splitting compared to the BSP technique. It thus reduces rendering time sensitive to vertex and line segment numbers. Furthermore, and significantly, the algorithm takes advantage of scenes such as buildings by using their corridors to create the skeleton which forms the graph and subsequently reduces the aforementioned splits even further.

8 Future

In this paper, we have concentrated our efforts on the two-dimensional problem. We are aware that this is only suitable to walk-throughs of environments comprising vertical polygons or where a height restriction is used [James98c, James98b]. We intend to explore the extension of the algorithm to handle general three-dimensional cases by creating a 'honeycomb' of polygons (instead of a web of line segments). Similar usage of corridors will succeed with the honeycomb since floors can be divided by a plane before each corridor is divided by a wall of vertical polygons.

Finally, we believe that the use of corridors as skeletons could be used in a later part of the rendering pipeline to aid or complement a portal algorithm. In particular it should be noted that only visible *corridors* (or auxiliary dividers) need to be considered and, due to the direct correspondence between corridors and graph arcs, this points to the cells which require rendering. Similarly, if the observer is located within a room, the arcs adjoining the node representing the room would point to neighbours which are visible and thus the possibility of an output sensitive Web Clustering algorithm is realistic.

9 Acknowledgments

We would like to thank our colleague, Ben Wyatt, for implementing the building generator used to create the test data in section 6.

REFERENCES

[Aliag97] D. G. Aliaga and A. A. Lastra. Architectural walkthroughs using portal textures. *Proceedings of IEEE Visualisation*, pages 355–362, October 1997.

[Fuchs79] H. Fuchs, Z. M. Kedem, and B. F. Naylor. Predetermining visibility priority in 3d scenes. *ACM Computer Graphics Proceedings*, 13(2):175–181, August 1979.

[Fuchs80] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. *ACM Computer Graphics*, 14(3):124–133, July 1980.

[Fuchs83] H. Fuchs, G. D. Abram, and E. D. Grant. Near real time shaded display of rigid objects. *ACM Computer Graphics*, 17(3):65–72, July 1983.

[Gordo91] D. Gordon and S. Chen. Front-to-back display of bsp trees. *IEEE Computer Graphics and Applications*, 11(5):79–85, 1991.

[Green96] N. Greene. Hierarchical polygon tiling with coverage masks. *ACM Computer Graphics Proceedings*, pages 65–74, 1996.

[James98a] A. James and A. M. Day. Conflict neutralization on binary space partitioning (extended abstract). *Eurographics UK Proceedings*, 16:225–229, March 1998.

[James98b] A. James and A. M. Day. The hidden face determination tree. *Eurographics '98 Short Presentations*, pages 1.4.1 – 1.4.2, September 1998.

[James98c] A. James and A. M. Day. The priority face determination tree for hidden surface removal. *Computer Graphics Forum*, 17(1):55–71, March 1998.

[James98d] A. James and A. M. Day. Conflict neutralization on binary space partitioning. Submitted for publication, 1998.

[Schum69] R. A. Schumaker, R. Brand, M. Gilliland, and W. Sharp. Study for applying computer-generated images to visual simulation. Technical Report AFHRL-TR-69-14, U. S. Airforce Human Resources Laboratory, 1969.

[Telle91] S. J. Teller and C. H. Sequin. Visibility preprocessing for interactive walkthroughs. *ACM Computer Graphics*, 25(4):61–69, July 1991.

[Telle93] S. J. Teller and P. Hanrahan. Global visibility algorithms for illumination computations. *Computer Graphics Proceedings*, pages 239–246, 1993.

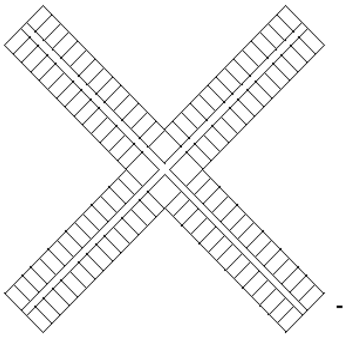

Figure 11: A simple scene

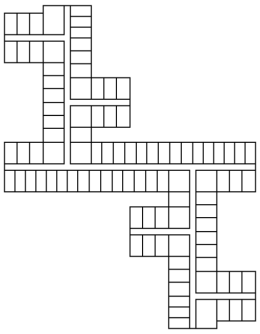

Figure 12: Orthogonal Corridors

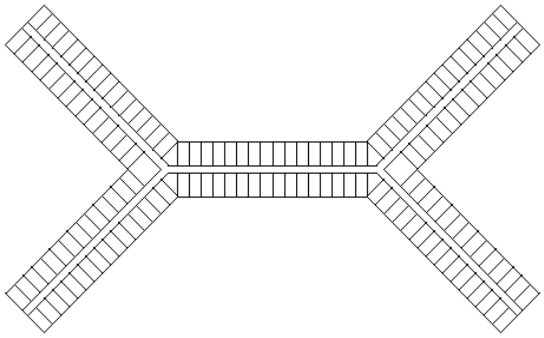

Figure 13: Branching Corridors

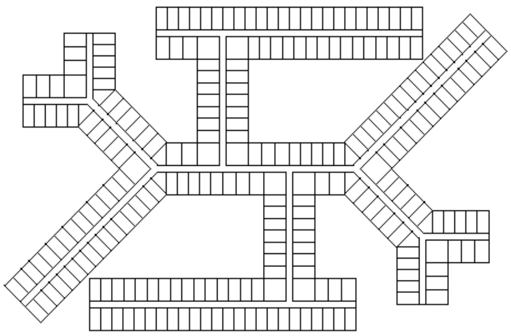

Figure 14: Multiple Branches

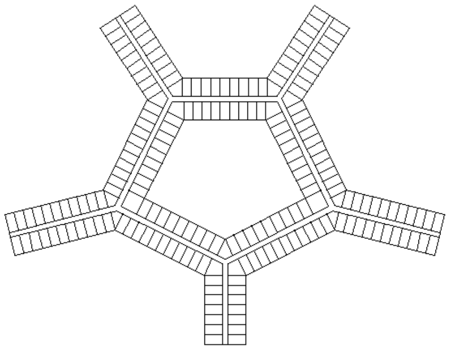

Figure 15: Cyclic Corridors

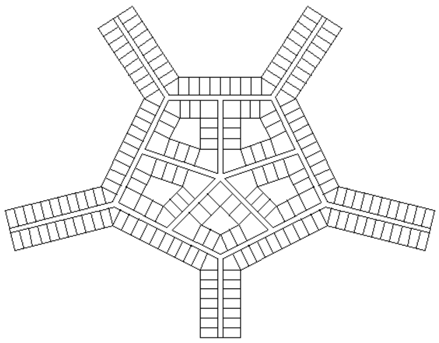

Figure 16: Multiply Cyclic