ABSTRACT

Title of Thesis: FASTER DISPLAY OF MECHANICAL ASSEMBLIES BY DETERMINATION OF PART VISIBILITY

Jeremy Ou, Master of Science, 2004

Thesis directed by: Professor Edward B. Magrab.
Department of Mechanical Engineering

We present algorithms that greatly decrease the time it takes to display a large number of 3-D mechanical part assemblies by removing all interior parts that cannot be viewed from any viewing angle. The algorithms are based on the minimum axis-aligned bounding box of each part, which avoids complicated computations often needed to determine the interactions of the geometry of the parts. The major contribution of this work is the use of exterior traces of cross sections of the bounding boxes to determine the parts' visibility. It is shown that the processing time increases almost linearly with the number of parts in an assembly of parts. A test on an assembly composed of 490 parts shows that the algorithms decrease the display time by a factor of two while only incorrectly identifying two of these parts as invisible when they should have been identified as visible.

FASTER DISPLAY OF MECHANICAL ASSEMBLIES BY DETERMINATION OF PART VISIBILITY

by

Jeremy Ou

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2004

Advisory Committee:

Professor Edward B. Magrab, Chair
Professor Satyandra K. Gupta
Professor Linda C. Schmidt

TABLE OF CONTENTS

List of Tables

List of Figures

1. Introduction[1]

In computer-aided design (CAD), the display of assemblies of parts is very important. It allows designers to identify parts and their interactions that could not be identified by looking at each part individually. However, this display becomes slow and unmanageable as the number of parts increases. The need to be able to display more intricate assemblies that include thousands of complex parts is increasing. Such assemblies include commercial vehicles, military vehicles, satellites, and machines with many moving parts. An attempt to view any of these assemblies in their entirety makes interactive viewing very difficult. It can take hours to bring the model into display for the first time. If the viewing angle is changed, it can take many minutes for the CAD program to calculate what is to be displayed in the new view. It is frequently not feasible to use these CAD programs in these situations. In order to view these assemblies at an acceptable update rate, it is necessary to reduce the amount of geometry being displayed. This is usually done manually, with the user removing subassemblies, surfaces, and polygons that are deemed unimportant. This research provides a way for the computer to do this automatically, so that it is feasible to display large assemblies in a shorter amount of time and without using a manual process.

In addition, the need to view these assemblies from data transmitted over the internet is growing. Allowing people to view assemblies over the web causes the amount of information needed to become an important consideration for bandwidth

---

1 This work was carried out under SBIR contract NAS3-00078 and is protected by SBIR laws. It is also protected by US patent #6,335,732 B1. This work was performed by Technology Promotion International, College Park, Maryland, 20740

purposes. A further problem is that CAD requires fast processors and large amounts of memory for the computers that are used for display. Computers with slower processors are more prevalent. These computers are only able to manageably display and manipulate simple assemblies. More complex assemblies will cause a display update to take at least a few seconds after a change in view or configuration, which may be too slow for practical use. The idea of making these assemblies available for viewing over the internet is to give anyone who has access the ability to view them without requiring a workstation. This research will allow rather large assemblies to be displayed via the web by reducing the amount of information that needs to be sent from the server to the web client. All of this information will need to be stored in the client's memory. This reduction of overhead will also allow more commonly used computers to be able to display these assemblies at an acceptable update rate.

2. Related Work

There have been many approaches to the simplification of models to achieve acceptable rates of display with a minimal loss of detail. Researchers have given methods for simplifying geometry for modeling purposes (Brodsky and Watson, 2000; Armstrong et al., 2000). These methods, however, decrease the geometric accuracy of the models used. This means that using the simplified models to make measurements or determine simulation paths will give results that are not as realistic as those of the unsimplified model. Another method is a multiresolution approach, where the level of detail changes as needed based on the distance from the viewer. One such approach is given by Huerta et al. (1998) and others by Krus et al. (1997). These methods may increase display speed, but increase overhead, as representations for each level of detail must be kept in memory. In a distributed environment, this method uses a large amount of bandwidth because all of this information must be transmitted from the server to the client.

Visibility determination is required for the computer graphics display of 3D scenes, which may be composed of any number of objects. Each object is described by surface polygons or surface representations that are polygonized before display. A hardware construction known as the Z-buffer computes what the 2D representation of these objects on the screen from all of the polygons in the 3D scene. It does this by determining for each polygon the color and distance from the viewpoint of each pixel for that surface. When the distance at a certain pixel of a polygon is closer than its previous value, the pixel values are updated with those from the new polygon. In this way, exact visibility is determined, meaning what is displayed is pixel-for-pixel what

the object would look like if a human being were to look at the object from the same viewing angle. However, for large scenes with many surfaces, enormous amounts of time are required to render this view because all surfaces are rendered, even those that are invisible. This may be acceptable if a static view is desired, but in simulations and in computer-aided design environments, objects and viewpoints are constantly changing. This means that the entire calculation must be performed again to reflect these changes. One solution to speeding up this process is to reduce the number of polygons sent to the Z-buffer without affecting what is ultimately displayed. Those that are not visible are not rendered.

Several researchers have considered different methods to process the visibility of surfaces or polygons to speed up their display. The most basic of these is backface culling, which keeps all polygons that face away from the viewer from being sent to the Z-buffer, since these polygons will never seen by the viewer. Several researchers have worked on methods to optimize the backface culling process (Kumar et al., 1996; Levi et al., 1999; Zhang and Hoff, 1997). Another method is called view-frustum culling. This involves the use of a rectangular-base pyramid to represent the view of the observer. All polygons that are not contained in or do not intersect this pyramid are determined to be invisible and are removed from rendering consideration (Hoff, 1997). Finally, a third method is known as occlusion culling. This process removes from consideration polygons that are “occluded” or entirely covered by polygons in front of them. Different methods are given by Bittner et al. (1998), Bormann (2001), Hudson et al. (1997), Möller and Haines (1999), and Zhang et al. (1997). The advantage of these methods is that the time it takes to make these calculations is much less than that taken

to render the polygons that have been removed by these methods. However, these methods are viewing-angle dependent, meaning that if the viewing angle changes, everything must be recalculated. This calculation will be faster than using the Z-buffer alone, but view changes will still be slow in those scenes with large polygon counts. Another weakness of these methods is that they require the same amount of information to be held in the computer's memory as with the Z-buffer method alone. In a distributed environment, this is also a problem, because there is no reduction in the amount of information that needs to be sent from a server to a client. This amount of information is a major concern for bandwidth purposes.

These shortcomings can be alleviated if some visibility preprocessing can be performed before calculations are made for a specific viewing angle. In this way, as the viewing angle changes, there will be fewer polygons to consider for that view. In addition, the preprocessing is performed only once, since the results from the preprocessing can be stored for use in later display. The calculations do not need to be performed again. Teller and Séquin (1991) offer a solution for axis-aligned architectural models where all viewpoints will be from the interior. They spatially divide the model into cells or "rooms." From each cell, they determine the visibility of all the other cells. Using this preprocessed information, only the geometry belonging to the corresponding visible cells will be rendered when a viewpoint is placed in a cell. It is only when the viewpoint moves to another cell that the geometry considered for display will change. In this way, displaying the entire model is not attempted, only those cells that are potentially visible from the current viewpoint. This is a good approach, but only works with axis-aligned architectural models.

Durand et al. (2000) give a preprocessing approach similar to Teller and Séquin's, but in a more general manner. Their approach also divides the viewing space into cells, although this processing does not require an architectural model. Using an occlusion culling technique, preprocessing is performed to calculate the visibility of polygons from each cell. This visibility is used at the time of rendering to only process those polygons that were calculated as visible from the cell that contains the viewpoint. The drawbacks of this approach are that the preprocessing time can be very lengthy, as the number of cells impacts heavily on the number of calculations needed. In order to minimize the number of polygons visible from a cell, it is necessary to make the cell relatively small in size, resulting in a high number of cells. In addition, the viewing space must be known ahead of time to ensure that all possible viewpoints have a corresponding cell. This is not acceptable for interactive applications where the user must be able to change the viewpoint to any location.

The research described here uses a bounding box approach to calculate visibility. Martin and Stephenson (1988) have investigated the placement of objects in boxes. Bounding boxes have been used in computer graphics to speed up visibility, ray tracing, and collision determinations (Iones et al., 1998). They have been used mainly to speed up intersection detection between entities. Interference or collision between an object and a view frustum, ray, or other object can first be checked with the bounding box. There can be no collision with the object if there is no collision with its bounding box. This saves on the calculation time needed, as collision detection with the box is simpler, and it eliminates many of the complicated calculations needed to detect interference between two objects that are not even close to each other. Iones et al.

(1998) studied the optimality of using bounding spheres, axis-aligned bounding boxes (AABBs), and oriented bounding boxes (OBBs) in the applications of frustum culling, ray shooting, and collision detection. The study found OBBs to be the optimum choice; however, AABBs are used in the current research. The additional benefit of OBBs over AABBs is unknown for the current application. OBBs have the benefit of more closely fitting their parts. However, the calculations using OBBs may use enough processing time to outweigh this benefit. Research has also been done into the properties of geometry that fare well in bounding box intersection determination and why the bounding box technique itself fares so well (Suri et al., 1999; Zhou and Suri, 1999). Sanna and Montuschi (1995) propose the use of bounding box groups instead of one single bounding box to increase the performance of the bounding box technique. They also offer ways to limit the number of bounding boxes surrounding an object while minimizing the volume enclosed. Kitamura (1998), Zachmann (1997), and Yu et al. (1996) have used the bounding box in collision detection. They have subdivided the box so that if a collision is detected between boxes, it is known in which subdivision(s) the interference occurs. Thus, only polygons within these subdivisions need to be checked for collision, reducing the collision detection process time dramatically.

2.1. Proposed Work

The current research addresses the display and preprocessing speed problems by adopting a part visibility method for large assemblies. In large complex assemblies, many parts may be enclosed or hidden by other parts from every viewing angle, making their inclusion in rendering unnecessary. This method makes use of bounding boxes in

preprocessing to determine the visibility of these parts. Only those parts that are visible from some viewing angle are included in the rendering of the assembly.

Some of the previously described work has been directed toward the simplification of models for display. However, this simplification modifies the geometry, making its use less accurate. The current research makes no modification to part geometry, it only removes from consideration geometry that is not visible from any angle. In addition, preprocessing is performed that does not require information about visibility of geometry from different viewpoints. This means that these algorithms can be applied to assemblies in different CAD packages without having to rewrite any of the code used for display, as other preprocessing methods (Teller and Séquin, 1991 and Durand et al., 2000) would require. The main advantage of this research comes through the advent of the internet and the world wide web. Bandwidth is a major consideration in the display of assemblies over the internet. This research first reduces the amount of geometry by removing all parts not visible from any viewing angle. The reduction of the amount of geometry will reduce the amount of data that needs to be sent and will allow these models to be viewed more quickly and easily in a distributed environment.

In order to accomplish this task, the following process is used to determine the visibility of the parts within an assembly.

1. We first determine the minimum axis-aligned bounding box (AABB) for each part and reason only with the AABBs, not with the actual geometry of the parts . This is discussed in Chapter 3.

2. We next find the AABBs that are contained in other AABBs. Parts whose AABBs are contained in other AABBs are candidates for being marked invisible and are analyzed further. This is discussed in Chapter 4.
3. Next, we determine visibility of the parts whose AABBs are not contained in other AABBs using a cross-section trace method. This method allows us to determine which parts can be seen from different viewing angles. This is discussed in Section 5.1.
4. Finally, we determine the visibility of the parts whose AABBs are contained in other AABBs. Their visibility is dependent on the visibility of the containing AABBs' parts. This is discussed in Section 5.3.

3. Finding the Axis-Aligned Bounding Box (AABB)

In this chapter, we describe the methods used to determine the AABB of each part, which will be used to determine part visibility. The definition of a part is somewhat arbitrary, as the modeler determines what he wants to use as a part. However, a part must be geometrically static, meaning that it cannot change in size or shape throughout its application. An AABB is the minimum surrounding rectangular parallelepiped whose edges are parallel to the axes of the global coordinate system. An example of an AABB is shown in Figure 3.1.

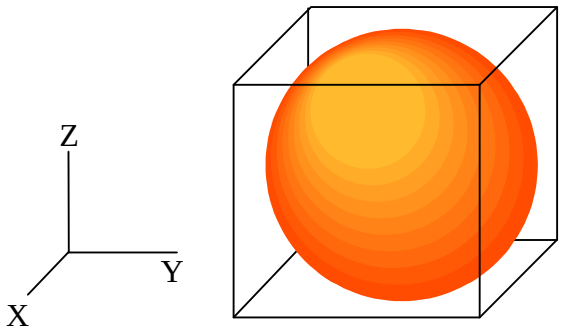


Figure 3.1. AABB of a sphere.

To find the AABB for a part, the following procedure is used for each surface of the part. These surfaces can be of many different types, including tabulated cylinders, surfaces of revolution, and rational B-spline surfaces.

1. Find the AABBs of the surface's edges. This gives us an initial basis for finding the AABB of the surface. This is discussed in Section 3.2.1.
2. Find any interior points of the surfaces that can extend the AABBs of the edges. To find the AABB of the surface, there may be points within the surface that lie outside the AABB of the edges. Thus, we need to find these

points thatextend the AABB to encompass the surface. This is discussed in Section 3.2.2.

3. If needed, we modify the dimensions of the part AABB to account for the the current surface. More detail is found in Section 3.2.

3.1. IGES specification

We require that CAD models be converted to an Initial Graphics Exchange Specification (IGES) format before each AABB is determined. IGES is a standard format that is compatible with many commercial CAD systems.

Conversion to IGES changes each part model from a solid model to a surface model. A solid model is represented as a solid block, cylinder, or other solid entity with protrusions, cuts, and rounds added to or subtracted from them. However, surface models use planes and curved surfaces to describe the exterior of the part. After being converted to the IGES format, the model becomes a collection of trimmed surfaces. A trimmed surface is a general surface that is represented by lines and curves that define its edges. Examples of trimmed surfaces are shown in grey in Figure 3.2.

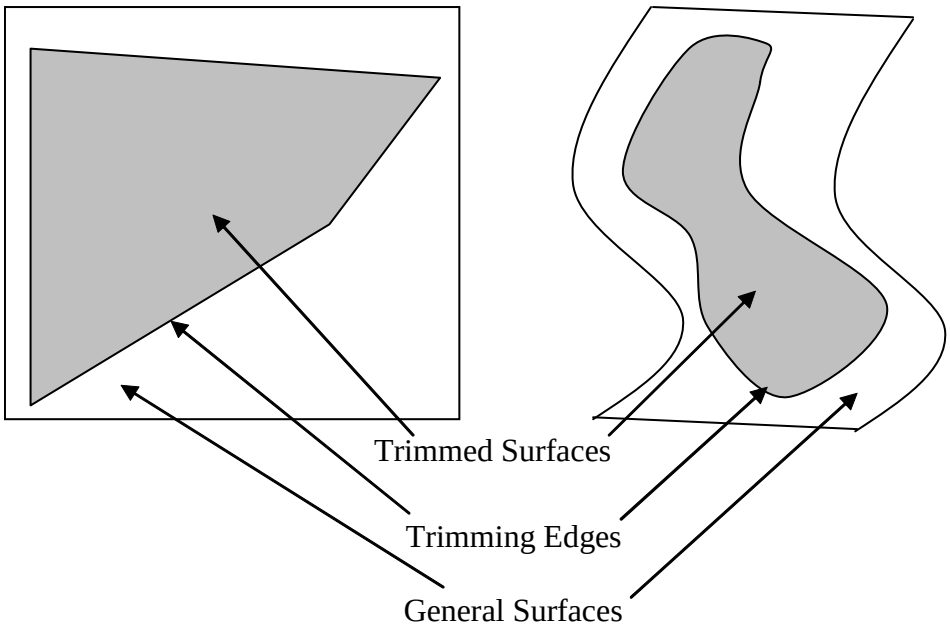


Figure 3.2. Examples of trimmed surfaces in grey.

Each trimmed surface is specified by a general surface description and the trimming edge that defines the borders of the surface. The general surfaces are either tabulated cylinders, surfaces of revolution, or rational B-spline surfaces. Each of these surfaces is defined in detail subsequently. Trimming edges are closed curves composed of lines, arcs, and rational B-spline curves. The specifications for the trimmed surfaces of a part are used to determine the AABB.

In doing this translation to IGES, it is necessary that all parts be placed in a global coordinate system, as the algorithms to be introduced will require that all AABBs' coordinate systems be orthogonal to each other. In the modeling of parts, each part is created in its own local coordinate system, usually the coordinate system that makes it easy to create the part model. However, when these parts are assembled, even though each part has its own local coordinate system, there will be only one coordinate system that pertains to the entire assembly, the global coordinate system. An example

is shown in Figure 3.3. This is a two-part assembly, with a smaller rectangular block part placed on the non-orthogonal face of the larger block part. For the smaller block, it is easiest to create this part in a coordinate system whose axes lie along the block's edges. When the smaller block is placed in its position in the assembly, its local coordinate system occurs at (u, v, w). However, the global coordinate system for this assembly is (X, Y, Z). Therefore, the specifications for the geometry of the block will use the (X, Y, Z) coordinate system. In addition, the algorithms will determine the AABBs in the (X, Y, Z) coordinate system as well. Since the smaller block is not orthogonal with the global coordinate system, its AABB (shown in Figure 3.3) does not have the same dimensions as the rectangular block itself. Since the algorithms require that the global coordinate system be used, an empty volume in the AABBs results for those parts that are of a unique shape or are not oriented orthogonal to the global coordinate axes. This empty volume can affect accuracy, with more empty volume resulting in less accurate results; that is, in the algorithm incorrectly marking an AABB invisible when it should be visible and vice versa. There are ways to mitigate these effects and they are also discussed. However, the problem of empty volume cannot be completely eliminated.

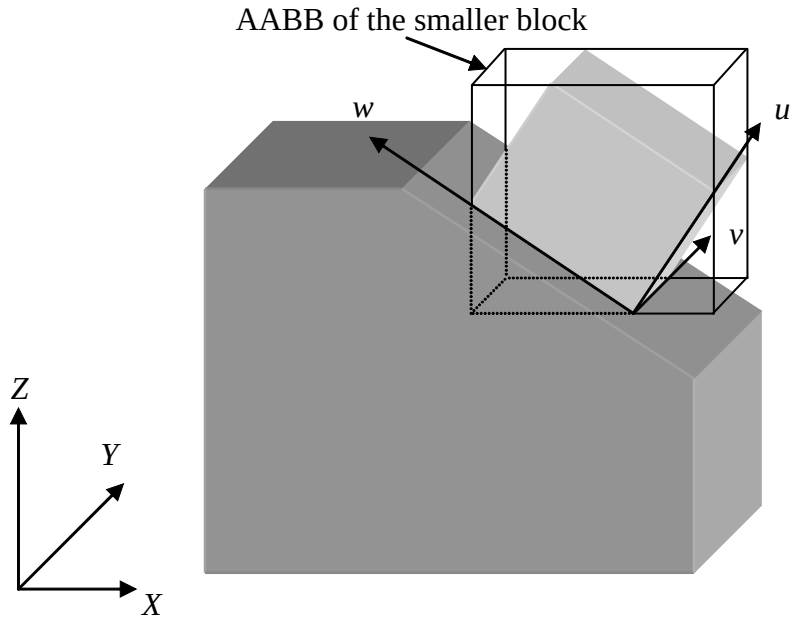


Figure 3.3. An example of local and global coordinate systems.

3.2. AABB Determination Algorithm

The AABB determination algorithm calculates in the global coordinate system the bounds of the AABB of a part. These bounds consist of the minimum and maximum coordinates in the three orthogonal directions that are encountered among all geometric points in the part. The flow chart of the algorithm, which is given in Figure 3.4, is now described.

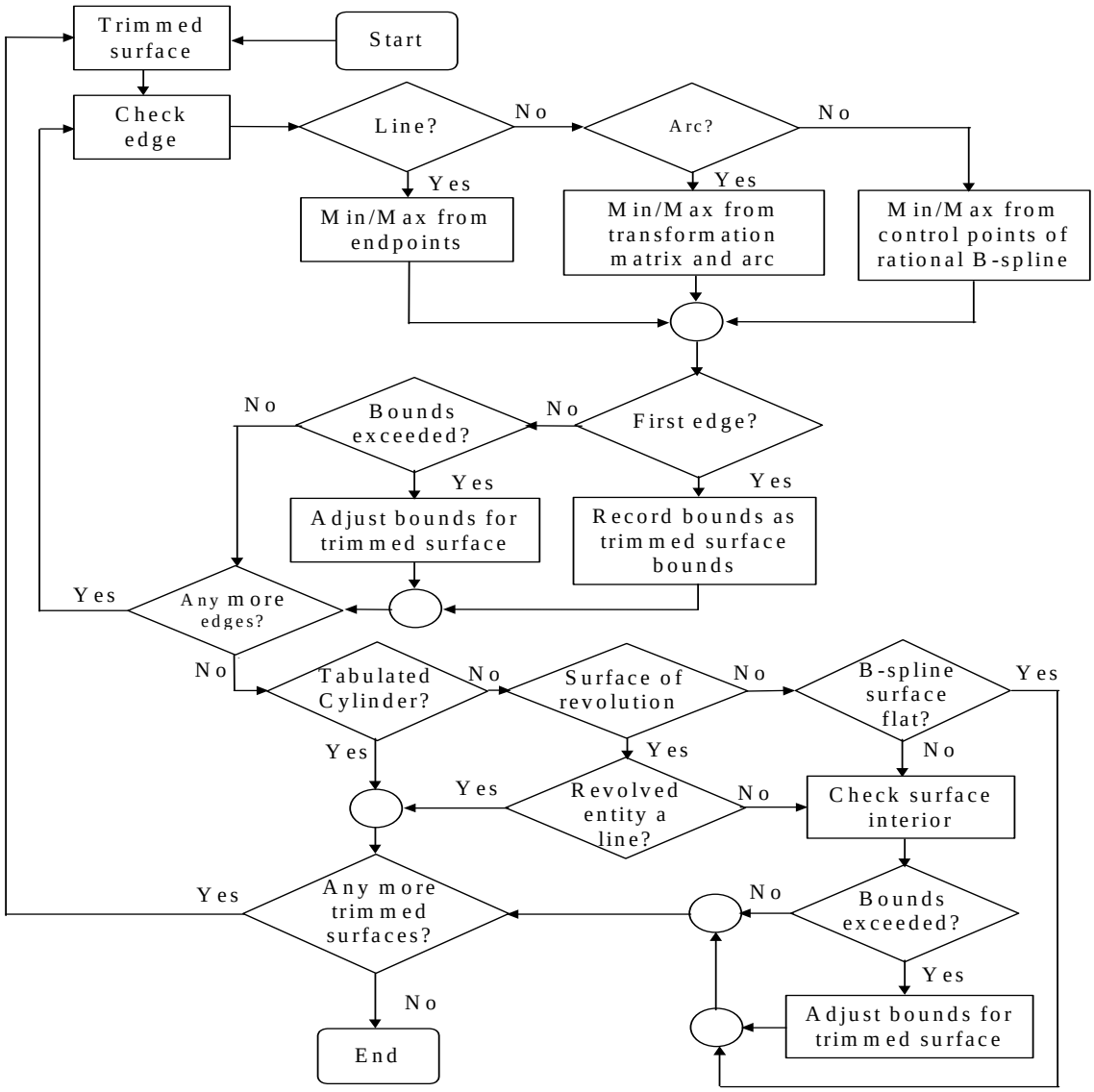


Figure 3.4. AABB Determination Algorithm Flow Chart.

First, the algorithm determines the AABB of a trimmed surface. Then the next trimmed surface is retrieved and its AABB is calculated. The values of the previous box's bounds are adjusted so that the new bounds contain both boxes. This simply entails comparing, for each orthogonal direction, the two maximum bounds and taking the larger of the two as the new maximum bound. Similarly, the lower bounds for each direction are compared and the lower of the two values is used as the new minimum

bound for the direction. This is repeated for all trimmed surfaces until the size of the final box is obtained. The final box is then the AABB of the part. The pseudo code for the above algorithm is given in Table 3.1.

Table 3.1. Pseudo code for AABB determination

```
Start with a trimmed surface S1
Determine AABB bounds C1 for S1
AABB = C1
for j = 2 to N
      Take Sj
      Determine Cj for Sj
      AABB = extreme(Cj, AABB)
endfor
```

The AABB of a trimmed surface is found in two steps. First, the AABB for the trimming edge is found. Then, for certain types of surfaces the AABB is adjusted based on the interior points of the trimmed surface. AABBs are determined by finding the coordinates of critical points such as vertices and the maxima and minima of curves and surfaces.

3.2.1. AABB of the Trimming Edge

The trimming edge of a surface is a closed curve consisting of line, arc, and rational B-spline curve entities. The approach to finding the AABB is much the same as that of Table 3.1, but instead of surfaces, edge entities are used. An AABB is found for the first entity, then this AABB is expanded, if necessary, to enclose the AABB for each subsequent entity. These AABBs are found by finding those points of the entities that have the minimum or maximum coordinate in the three orthogonal axial directions.

AABB of a Line

The minimum AABB of a line is determined as a box with the line's two endpoints at opposite corners as shown in Figure 3.5. Since the minimum AABB is determined by the maximum and minimum coordinates in the three orthogonal directions, this occurs at the endpoints of the line.

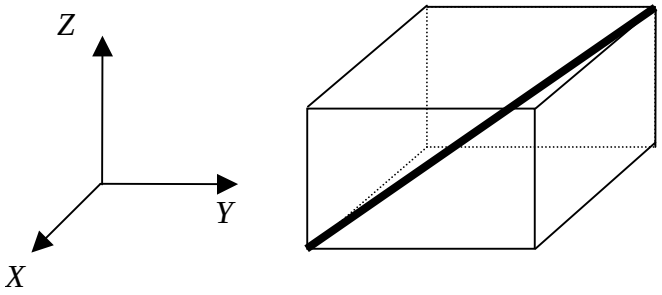


Figure 3.5. Minimum AABB of a line.

AABB of an Arc

An arc in IGES is specified by two components. The first is a 2-D representation of the arc that is parallel to the *X*-*Y* plane of the global coordinate system. This 2-D representation is geometrically congruent to the actual arc in 3-D space. The 2-D representation is specified by the (*x*, *y*) coordinates of the center, start, and terminate points as shown in Figure 3.6, and a displacement from the *X*-*Y* plane (the *z*-value).

The second component is a transformation matrix that rotates and translates this arc to its actual place in 3-D space, shown in Figure 3.7. The values x_c, y_c, x_s, y_s. x_t, and y_t in Figure 3.6, when transformed, could be different from the coordinates of the corresponding points, and thus do not appear in Figure 3.7. The transformation matrix ***R*** is specified by:

$$\boldsymbol{R} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where R_{ij} are the coefficients that rotate the configuration about an axis through the origin. The coefficients T_j are the translation factors. They translate the rotated configuration to its desired location. The transformation matrix transforms a point (x_i, y_i, z_i) on the 2-D arc to another point (x_o, y_o, z_o) on the 3-D arc by the following matrix multiplication

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} = \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix} \tag{1}$$

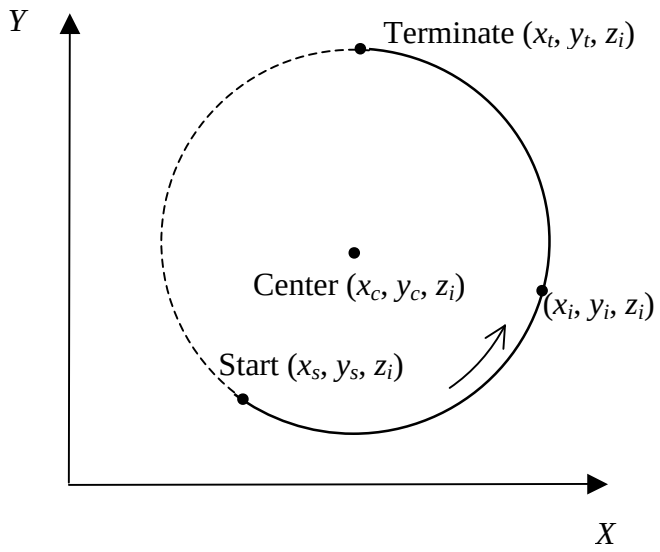


Figure 3.6. 2-D representation of an arc in IGES

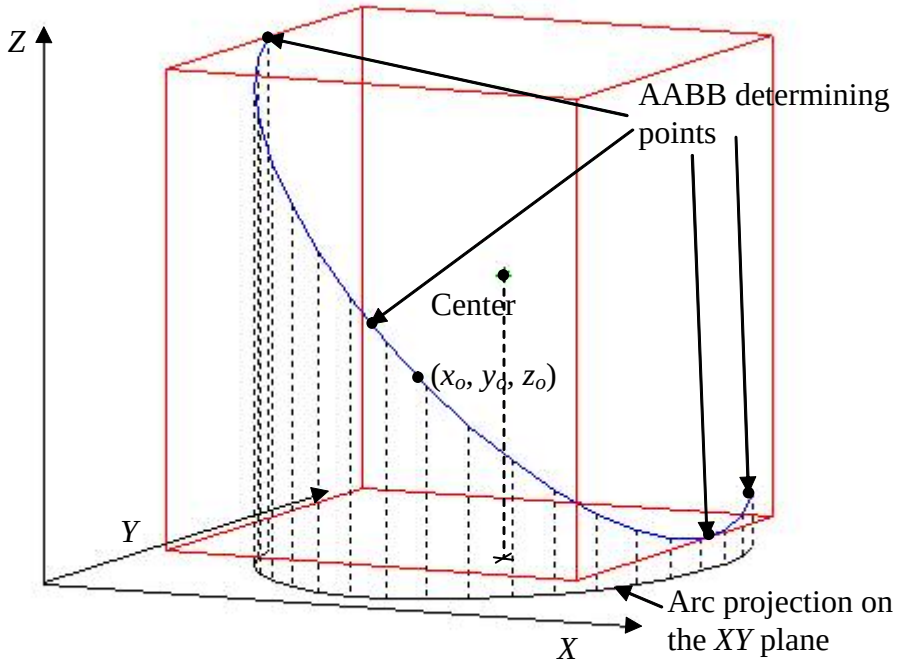


Figure 3.7. Arc (blue) in 3-dimensional space. Arc AABB is in red.

The minimum and maximum values of x_o, y_o, and z_o determine the bounds of the AABB of the arc in its oriented position in space. We first use the endpoints of this arc as initial values for the bounds, as they often do turn out to be AABB determination points. Thus, we use the start and terminate points as (x_i, y_i, z_i).

It is now necessary to compute any interior points on the arc that may require the bounds of the AABB to be expanded. We consider first the equation for x_o; the equations for y_o and z_o will be similar. Thus, from Equation (1)

$$x_o = R_{11}x_i + R_{12}y_i + R_{13}z_i + T_1 \tag{2}$$

The only values of this equation that will change along the arc are x_i and y_i. We require the minimum and maximum of

$$f(x_i, y_i) = x_o - R_{13}z_i - T_1 = R_{11}x_i + R_{12}y_i$$

If we let $x_i = x_c + r\cos\theta_i$ and $y_i = y_c + r\sin\theta_i$, where θ_i is the angle with respect to the positive *X*-axis and *r* is the radius of the arc, then this expression becomes

$$f(r_i, \theta_i) = R_{11}x_c + R_{12}y_c + r(R_{11}\cos\theta_i + R_{12}\sin\theta_i)$$

To find the minimum and maximum, we determine the values of θ_i that satisfy

$$\frac{df}{d\theta_i} = r(-R_{11}\sin\theta_i + R_{12}\cos\theta_i) = 0$$

Therefore,

$$\tan\theta_i = \frac{R_{12}}{R_{11}} \tag{3}$$

This gives two points on the circle of the arc with corresponding angles that have tangents equal to R_{12} / R_{11}. One will give the minimum and the other the maximum value for x_o on the circle. However, these points may not actually lie on the arc. To determine this, we calculate the angles that correspond to the start and end points of the arc. To find θ_s, we have the equations $x_s = x_c + r\cos\theta_s$ and $y_s = y_c + r\sin\theta_s$. This gives

$$\cos\theta_s = \frac{x_s - x_c}{r} \text{ and } \sin\theta_s = \frac{y_s - y_c}{r}$$

Thus,

$$\theta_s = \tan^{-1}\frac{y_s - y_c}{x_s - x_c}.$$

Similarly, we find θ_t from

$$\theta_t = \tan^{-1}\frac{y_t - y_c}{x_t - x_c}$$

Thus, we determine each value of θ_i that satisfies Equation (3) to see if $\theta_s < \theta_i < \theta_t$, which means that the point corresponding to θ_i lies on the arc. If one, or both, of these values does satisfy this requirement, then the corresponding point(s) causes the bounds of the AABB to be expanded past those determined by the endpoints. We thus use the relations $x_i = x_c + r\cos\theta_i$ and $y_i = y_c + r\sin\theta_i$ and Equation (2) to find the value of x_o and the bounds of the box in the X -direction are adjusted accordingly. Similarly, the minimum and maximum values for y_o and z_o are obtained when

$$\tan\theta_i = \frac{R_{22}}{R_{21}}$$

and

$$\tan\theta_i = \frac{R_{32}}{R_{31}}$$

respectively. Thus, we take the corresponding values of θ_i to check if these are on the arc, and if so, we adjust the bounds accordingly.

AABB of a Rational B-spline Curve

General curves are represented as rational B-spline curves. These curves are obtained by applying a smoothing function to sets of ordered control points as shown in Figure 3.8.

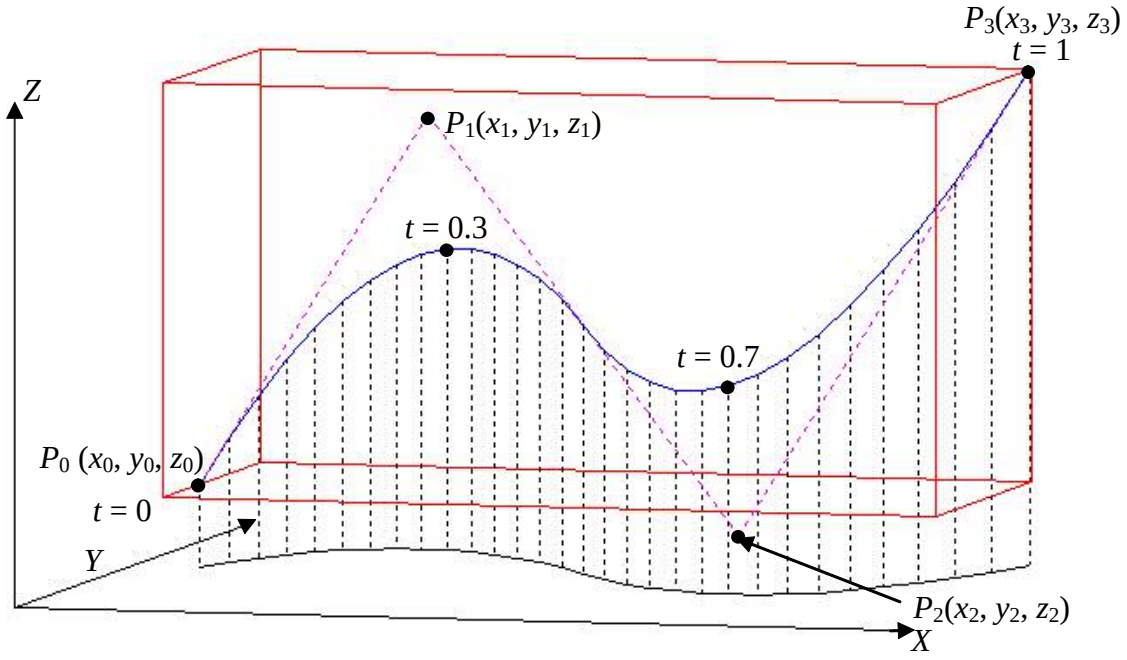


Figure 3.8. Rational B-spline curve (blue) and control points, P_i. Curve's AABB in red.

In determining the points on the curve, a parametric value of t is used. In most CAD systems, $0 \le t \le 1$. This parametric value is used to create a one-to-one mapping of t in the interval [0, 1] to the points on the curve, with $t = 0$ mapping to the first endpoint, and $t = 1$ mapping to the last endpoint. The curve is calculated as a function $G(t)$, which is given in the IGES 4.0 Specification (1988) by

$$G^{(M)}(t) = \frac{\sum_{i=0}^{K} W_i P_i b_i^{(M)}(t)}{\sum_{i=0}^{K} W_i b_i^{(M)}(t)} \tag{4}$$

where M is the order of the curve, K is the number of control points minus 1, W_i are the weights of each point, P_i are the control points (x_i, y_i, z_i), and $b_i^{(M)}(t)$ is an M^{th}-degree function determined by a knot sequence in t. The non-decreasing knot sequence in t is $t_{(-M)}, t_{(-M+1)}, \ldots, t_{(K+1)}$. In the knot sequence, $t_{(-M)} = t_{(-M+1)} = \ldots = t_0$ and

$t_{(K-M+1)} = t_{(K-M+2)} = \ldots = t_{(K+1)}$. In many commercial CAD systems, $W_i = 1$, $i = 0,1,\ldots,K$. The function $b_i^{(k)}(t)$ is a recursive function of degree *k* and is defined as follows:

$$b_i^{(0)}(t) = 1 \quad t_i \le t < t_{i+1} \qquad i = 0,1,\ldots,K \tag{5}$$
$$= 0 \quad \text{otherwise}$$

$$b_i^{(k)}(t) = \frac{t - t_{i-k}}{t_i - t_{i-k}} b_{i-1}^{(k-1)}(t) + \frac{t_{i+1} - t}{t_{i+1} - t_{i-k+1}} b_i^{(k-1)}(t) \qquad \begin{matrix} i = 0,1,\ldots,K \\ k = 1,2,3,\ldots \end{matrix} \tag{6}$$

These functions have the property that

$$\sum_{i=0}^{K} b_i^{(k)}(t) = 1 \qquad k = 0,1,2,\ldots$$

Thus, the curve function reduces to

$$G^{(M)}(t) = \sum_{i=0}^{K} P_i b_i{}^{(M)}(t)$$

Using these equations, algebraic expressions for the locations of the minima and maxima for first to third-order rational B-spline curves are determined.

AABB of a first-degree curve

First we determine the function $b_i^{(1)}$. From Equation (6), we have

$$b_i^{(1)}(t) = \frac{t - t_{i-1}}{t_i - t_{i-1}} b_{i-1}^{(0)}(t) + \frac{t_{i+1} - t}{t_{i+1} - t_i} b_i^{(0)}(t) \qquad i = 0,1,\ldots,K$$

or

$$
\begin{aligned}
b_i^{(1)}(t) &= \frac{t - t_{i-1}}{t_i - t_{i-1}} && t_{i-1} \le t < t_i \\
&= \frac{t_{i+1} - t}{t_{i+1} - t_i} && t_i \le t < t_{i+1} \qquad i = 0,1,\ldots,K \\
&= 0 && \text{otherwise}
\end{aligned}
$$

Thus, the B-spline curve function reduces to

$$
G^{(1)}(t) = P_i\left(\frac{t_{i+1} - t}{t_{i+1} - t_i}\right) + P_{i+1}\left(\frac{t - t_i}{t_{i+1} - t_i}\right) \qquad t_i \le t < t_{i+1}, \quad i = 0,1,\ldots,K-1
$$

This equation results in a straight line from one control point to another, and the minimum and maximum points will be control points. Therefore, it is only necessary to examine the control points to determine the AABB.

AABB of a second-degree curve

In this case, Equation (6) becomes

$$
b_i^{(2)}(t) = \frac{t - t_{i-2}}{t_i - t_{i-2}} b_{i-1}^{(1)}(t) + \frac{t_{i+1} - t}{t_{i+1} - t_{i-1}} b_i^{(1)}(t) \qquad i = 0,1,\ldots,K
$$

or

$$
\begin{aligned}
b_i^{(2)}(t) &= \left(\frac{t - t_{i-2}}{t_i - t_{i-2}}\right)\left(\frac{t - t_{i-2}}{t_{i-1} - t_{i-2}}\right) && t_{i-2} \le t < t_{i-1} \\
&= \left(\frac{t - t_{i-2}}{t_i - t_{i-2}}\right)\left(\frac{t_i - t}{t_i - t_{i-1}}\right) + \left(\frac{t_{i+1} - t}{t_{i+1} - t_{i-1}}\right)\left(\frac{t - t_{i-1}}{t_i - t_{i-1}}\right) && t_{i-1} \le t < t_i \qquad i = 0,1,\ldots,K-1 \\
&= \left(\frac{t_{i+1} - t}{t_{i+1} - t_{i-1}}\right)\left(\frac{t_{i+1} - t}{t_{i+1} - t_i}\right) && t_i \le t < t_{i+1} \\
&= 0 && \text{otherwise}
\end{aligned}
$$

The curve equation then reduces to

$$G^{(2)}(t) = P_i\left(\frac{t_{i+1}-t}{t_{i+1}-t_{i-1}}\right)\left(\frac{t_{i+1}-t}{t_{i+1}-t_i}\right) + P_{i+1}\left[\left(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}\right)\left(\frac{t_{i+1}-t}{t_{i+1}-t_i}\right)+\left(\frac{t_{i+2}-t}{t_{i+2}-t_i}\right)\left(\frac{t-t_i}{t_{i+1}-t_i}\right)\right] + P_{i+2}\left(\frac{t-t_i}{t_{i+2}-t_i}\right)\left(\frac{t-t_i}{t_{i+1}-t_i}\right) \qquad \begin{matrix} t_i \le t < t_{i+1} \\ i = 0,1,\ldots,K-2 \end{matrix}$$

To find the minimum and maximum, we set the derivative of $G^{(2)}(t)$ with respect to t to zero. Thus,

$$\frac{dG(t)}{dt} = \left(\frac{1}{t_{i+1}-t_i}\right)\left\{-2P_i\left(\frac{t_{i+1}-t}{t_{i+1}-t_{i-1}}\right) + P_{i+1}\left[\left(\frac{t_{i+1}+t_{i-1}-2t}{t_{i+1}-t_{i-1}}\right)+\left(\frac{t_{i+2}+t_i-2t}{t_{i+2}-t_i}\right)\right] + 2P_{i+2}\left(\frac{t-t_i}{t_{i+2}-t_i}\right)\right\} = 0$$

Upon solving for t, we obtain the following expression for the extrema, $t_{i,ext}$.

$$t_{i,ext} = \frac{2P_i t_{i+1}(t_{i+2}-t_i) - P_{i+1}\left[(t_{i+2}-t_i)(t_{i+1}+t_{i-1})+(t_{i+1}-t_{i-1})(t_{i+2}+t_i)\right] + 2P_{i+2}t_i(t_{i+1}-t_{i-1})}{2\left[(P_i-P_{i+1})(t_{i+2}-t_i)+(P_{i+2}-P_{i+1})(t_{i+1}-t_{i-1})\right]} \qquad i = 0,1,\ldots,K-2$$

Substituting in the values of the x_i, y_i, and z_i coordinates of the control points P_i, we obtain three values for t_{ext} for each value of i:

$$t_{xi,ext} = \frac{2x_i t_{i+1}(t_{i+2}-t_i) - x_{i+1}\left[(t_{i+2}-t_i)(t_{i+1}+t_{i-1})+(t_{i+1}-t_{i-1})(t_{i+2}+t_i)\right] + 2x_{i+2}t_i(t_{i+1}-t_{i-1})}{2\left[(x_i-x_{i+1})(t_{i+2}-t_i)+(x_{i+2}-x_{i+1})(t_{i+1}-t_{i-1})\right]} \qquad i = 0,1,\ldots,K-2$$

$$t_{yi,ext} = \frac{2y_i t_{i+1}(t_{i+2}-t_i) - y_{i+1}\left[(t_{i+2}-t_i)(t_{i+1}+t_{i-1})+(t_{i+1}-t_{i-1})(t_{i+2}+t_i)\right] + 2y_{i+2}t_i(t_{i+1}-t_{i-1})}{2\left[(y_i-y_{i+1})(t_{i+2}-t_i)+(y_{i+2}-y_{i+1})(t_{i+1}-t_{i-1})\right]} \qquad i = 0,1,\ldots,K-2$$

$$t_{zi,ext} = \frac{2z_i t_{i+1}(t_{i+2}-t_i) - z_{i+1}\left[(t_{i+2}-t_i)(t_{i+1}+t_{i-1})+(t_{i+1}-t_{i-1})(t_{i+2}+t_i)\right] + 2z_{i+2}t_i(t_{i+1}-t_{i-1})}{2\left[(z_i-z_{i+1})(t_{i+2}-t_i)+(z_{i+2}-z_{i+1})(t_{i+1}-t_{i-1})\right]} \qquad i = 0,1,\ldots,K-2$$

Since each value of i gives a different equation for its respective portion of the curve $G^{(2)}(t)$, the three values $t_{xi,ext}$, $t_{yi,ext}$, and $t_{zi,ext}$ correspond to the local minima or maxima of a curve that is represented by this equation for all values of t. However, the equation for $G^{(2)}(t)$ is only valid on the interval $t_i \le t < t_{i+1}$. Thus, we only consider those values of $t_{xi,ext}$, $t_{yi,ext}$, and $t_{zi,ext}$ that are within this interval, as only these correspond to actual minima and maxima of the spline curve. For all of these valid values of t, we evaluate the points that correspond to those values through the equation $G^{(2)}(t)$. The bounds of the AABB for the second-degree curve are determined as the minimum and maximum x, y, and z coordinates of all of these points and the curve endpoints.

AABB of a third-degree curve

For this case, Equation (6) gives

$$b_i^{(3)}(t) = \frac{t - t_{i-3}}{t_i - t_{i-3}} b_{i-1}^{(2)}(t) + \frac{t_{i+1} - t}{t_{i+1} - t_{i-2}} b_i^{(2)}(t) \qquad i = 0,1,\ldots,K$$

or

$$b_i^{(3)}(t) = \left(\frac{t-t_{i-3}}{t_i-t_{i-3}}\right)\left(\frac{t-t_{i-3}}{t_{i-1}-t_{i-3}}\right)\left(\frac{t-t_{i-3}}{t_{i-2}-t_{i-3}}\right) \qquad t_{i-3} \le t < t_{i-2}$$

$$= \left(\frac{t-t_{i-3}}{t_i-t_{i-3}}\right)\left[\left(\frac{t-t_{i-3}}{t_{i-1}-t_{i-3}}\right)\left(\frac{t_{i-1}-t}{t_{i-1}-t_{i-2}}\right)+\left(\frac{t_i-t}{t_i-t_{i-2}}\right)\left(\frac{t-t_{i-2}}{t_{i-1}-t_{i-2}}\right)\right]+$$
$$\left(\frac{t_{i+1}-t}{t_{i+1}-t_{i-2}}\right)\left(\frac{t-t_{i-2}}{t_i-t_{i-2}}\right)\left(\frac{t-t_{i-2}}{t_{i-1}-t_{i-2}}\right) \qquad t_{i-2} \le t < t_{i-1}$$

$$= \left(\frac{t-t_{i-3}}{t_i-t_{i-3}}\right)\left(\frac{t_i-t}{t_i-t_{i-2}}\right)\left(\frac{t_i-t}{t_i-t_{i-1}}\right)+$$
$$\left(\frac{t_{i+1}-t}{t_{i+1}-t_{i-2}}\right)\left[\left(\frac{t-t_{i-2}}{t_i-t_{i-2}}\right)\left(\frac{t_i-t}{t_i-t_{i-1}}\right)+\left(\frac{t_{i+1}-t}{t_{i+1}-t_{i-1}}\right)\left(\frac{t-t_{i-1}}{t_i-t_{i-1}}\right)\right] \qquad t_{i-1} \le t < t_i$$

$$= \left(\frac{t_{i+1}-t}{t_{i+1}-t_{i-2}}\right)\left(\frac{t_{i+1}-t}{t_{i+1}-t_{i-1}}\right)\left(\frac{t_{i+1}-t}{t_{i+1}-t_i}\right) \qquad t_i \le t < t_{i+1}$$

$$= 0 \qquad \text{otherwise}$$

$$i = 0,1,\ldots,K-2$$

The equation for $G^{(3)}(t)$ then reduces to

$$G^{(3)}(t) = P_i\left(\frac{t_{i+1}-t}{t_{i+1}-t_{i-2}}\right)\left(\frac{t_{i+1}-t}{t_{i+1}-t_{i-1}}\right)\left(\frac{t_{i+1}-t}{t_{i+1}-t_i}\right)+$$
$$P_{i+1}\left\{\left(\frac{t-t_{i-2}}{t_{i+1}-t_{i-2}}\right)\left(\frac{t_{i+1}-t}{t_{i+1}-t_{i-1}}\right)\left(\frac{t_{i+1}-t}{t_{i+1}-t_i}\right)+\left(\frac{t_{i+2}-t}{t_{i+2}-t_{i-1}}\right)\left[\left(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}\right)\left(\frac{t_{i+1}-t}{t_{i+1}-t_i}\right)+\left(\frac{t_{i+2}-t}{t_{i+2}-t_i}\right)\left(\frac{t-t_i}{t_{i+1}-t_i}\right)\right]\right\}+$$
$$P_{i+2}\left\{\left(\frac{t-t_{i-1}}{t_{i+2}-t_{i-1}}\right)\left[\left(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}\right)\left(\frac{t_{i+1}-t}{t_{i+1}-t_i}\right)+\left(\frac{t_{i+2}-t}{t_{i+2}-t_i}\right)\left(\frac{t-t_i}{t_{i+1}-t_i}\right)\right]+\left(\frac{t_{i+3}-t}{t_{i+3}-t_i}\right)\left(\frac{t-t_i}{t_{i+2}-t_i}\right)\left(\frac{t-t_i}{t_{i+1}-t_i}\right)\right\}+$$
$$P_{i+3}\left(\frac{t-t_i}{t_{i+3}-t_i}\right)\left(\frac{t-t_i}{t_{i+2}-t_i}\right)\left(\frac{t-t_i}{t_{i+1}-t_i}\right) \qquad t_i \le t < t_{i+1}, 0 \le i < K$$

Again we set the derivative of this equation to zero to determine the maxima and minima. Thus,

$$\frac{dG^{(3)}(t)}{dt} = \left(\frac{1}{t_{i+1}-t_i}\right)\Bigg\{-3P_i\left(\frac{t_{i+1}-t}{t_{i+1}-t_{i-2}}\right)\left(\frac{t_{i+1}-t}{t_{i+1}-t_{i-1}}\right)+$$
$$P_{i+1}\Bigg[\left(\frac{t_{i+1}-t}{t_{i+1}-t_{i-2}}\right)\left(\frac{t_{i+1}+2t_{i-2}-3t}{t_{i+1}-t_{i-1}}\right)+\left(\frac{t_{i+2}-t}{t_{i+2}-t_{i-1}}\right)\left(\frac{t_{i+2}+2t_i-3t}{t_{i+2}-t_i}\right)+$$
$$\frac{3t^2-2(t_{i+2}+t_{i+1}+t_{i-1})t+t_{i+2}t_{i+1}+t_{i+2}t_{i-1}+t_{i+1}t_{i-1}}{(t_{i+2}-t_{i-1})(t_{i+1}-t_{i-1})}\Bigg]+$$
$$P_{i+2}\Bigg[\left(\frac{t-t_{i-1}}{t_{i+2}-t_{i-1}}\right)\left(\frac{2t_{i+1}+t_{i-1}-3t}{t_{i+1}-t_{i-1}}\right)+\left(\frac{t-t_i}{t_{i+3}-t_i}\right)\left(\frac{2t_{i+3}+t_i-3t}{t_{i+2}-t_i}\right)+$$
$$\frac{-3t^2+2(t_{i+2}+t_i+t_{i-1})t-t_{i+2}t_{i+1}-t_{i+2}t_{i-1}-t_{i+1}t_{i-1}}{(t_{i+2}-t_{i-1})(t_{i+2}-t_i)}\Bigg]+$$
$$3P_{i+3}\left(\frac{t-t_i}{t_{i+3}-t_i}\right)\left(\frac{t-t_i}{t_{i+2}-t_i}\right)\Bigg\}=0 \qquad t_i \le t < t_{i+1},\ \ i=0,1,\ldots,K-2$$

which results in an equation of the form

$$A_i t^2 + B_i t + C_i = 0 \qquad i = 0,1,\ldots,K-2$$

where

$$A_i = 3(P_{i+1}-P_i)(t_{i+3}-t_i)(t_{i+2}-t_{i-1})(t_{i+2}-t_i)+$$
$$3(P_{i+1}-P_{i+2})[(t_{i+3}-t_i)(t_{i+1}-t_{i-2})(t_{i+2}-t_i)+(t_{i+3}-t_i)(t_{i+1}-t_{i-2})(t_{i+1}-t_{i-1})]+$$
$$3(P_{i+3}-P_{i+2})(t_{i+2}-t_{i-1})(t_{i+1}-t_{i-2})(t_{i+1}-t_{i-1})$$

$$B_i = [6P_i t_{i+1} - P_{i+1}(4t_{i+1}+2t_{i-2})](t_{i+3}-t_i)(t_{i+2}-t_{i-1})(t_{i+2}-t_i)+$$
$$[-P_{i+1}(4t_{i+2}+2t_i)+2P_{i+2}(t_{i+2}+t_i+t_{i-1})](t_{i+3}-t_i)(t_{i+1}-t_{i-2})(t_{i+1}-t_{i-1})+$$
$$[-2P_{i+1}(t_{i+2}+t_{i+1}+t_{i-1})+P_{i+2}(2t_{i+1}+4t_{i-1})](t_{i+3}-t_i)(t_{i+1}-t_{i-2})(t_{i+2}-t_i)+$$
$$[P_{i+2}(2t_{i+3}+4t_i)-6P_{i+3}t_i](t_{i+2}-t_{i-1})(t_{i+1}-t_{i-2})(t_{i+1}-t_{i-1})$$

$$C_i = [-3P_i t_{i+1}^2 + P_{i+1}t_{i+1}(t_{i+1}+2t_{i-2})](t_{i+3}-t_i)(t_{i+2}-t_{i-1})(t_{i+2}-t_i)+$$
$$[P_{i+1}t_{i+2}(t_{i+2}+2t_i)-P_{i+2}(t_{i+2}t_{i+1}+t_{i+2}t_{i-1}+t_{i+1}t_{i-1})](t_{i+3}-t_i)(t_{i+1}-t_{i-2})(t_{i+1}-t_{i-1})+$$
$$[P_{i+1}(t_{i+2}t_{i+1}+t_{i+2}t_{i-1}+t_{i+1}t_{i-1})-P_{i+2}t_{i-1}(2t_{i+1}+t_{i-1})](t_{i+3}-t_i)(t_{i+1}-t_{i-2})(t_{i+2}-t_i)+$$
$$[-P_{i+2}t_i(2t_{i+3}+t_i)+3P_{i+3}t_i^2]$$

Thus, two values are found:

$$t_{i1,ext} = \frac{-B_i + \sqrt{B_i^2 - 4A_iC_i}}{2A_i} \text{ and } t_{i2,ext} = \frac{-B_i - \sqrt{B_i^2 - 4A_iC_i}}{2A_i} \qquad i = 0,1,\ldots,K-2$$

These values are candidate values of *t* that correspond to maximum and minimum points of a curve given by the equation for the *i*th section of the spline curve. However, only those points within the interval, $t_i \le t_{i,ext} < t_{i+1}$ are actual local maximum and minimum points of the spline curve. Therefore, we evaluate the actual coordinates of the points that correspond to these values of $t_{i,ext}$ with the curve equation. We then take the minimum and maximum *x*, *y*, and *z* coordinates of all of these points and the endpoints of the curve as the bounds of the AABB.

3.2.2. Examination of a surface's interior

After the AABB of the trimming edge is found, it is necessary to examine the interior of the trimming surface for extreme points that may lie outside the bounds of the current AABB. There are three different types of surfaces that are usually exported in IGES format: tabulated cylinder, surface of revolution, and rational B-spline surface. The examination of the interiors of these surfaces is now described.

Tabulated cylinders

Tabulated cylinders are surfaces formed by sweeping a line segment called the generatrix parallel to itself along a curve. Such a surface is shown in Figure 3.9, along with a trimming edge on that surface. It will be shown that the AABB of this surface is the same as the AABB of the trimming edges.

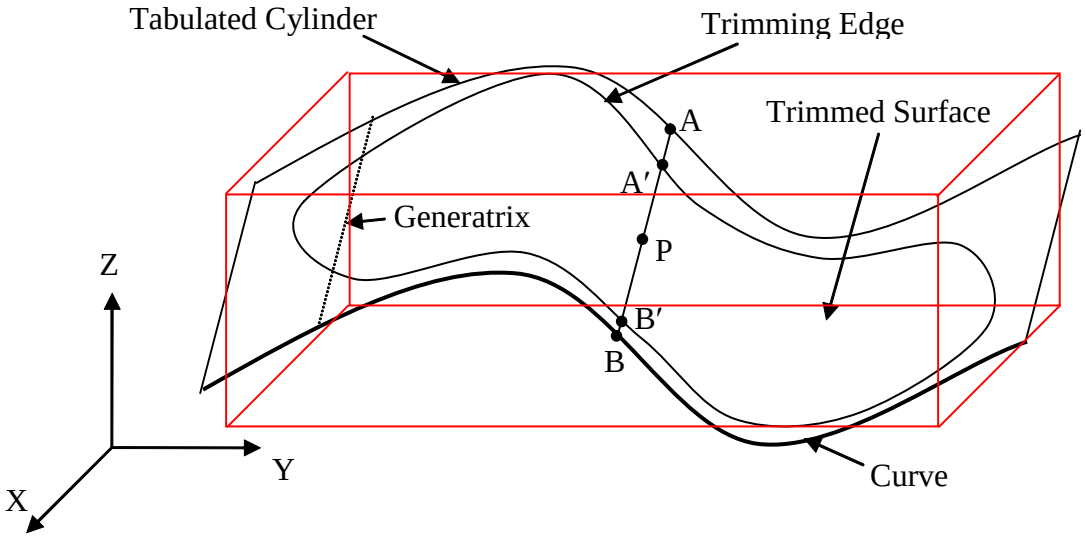


Figure 3.9. Trimmed surface of a tabulated cylinder. AABB is in red.

Referring to Figure 3.9, we take any point P in the interior of the trimmed surface. We then draw a line segment $\overline{AB}$ through P that is parallel to the generatrix of the tabulated cylinder. $\overline{AB}$ intersects the trimming edge at points A′ and B′. Thus, P is also on segment $\overline{A'B'}$. Since P is on this segment, it follows that it is contained in the AABB of this segment. Since the AABB of a line segment has the endpoints of the segment on opposite corners, we can deduce that one possible AABB containing P has A′ and B′ as its opposite corners. Because A′ and B′ are points on the trimming edge, this AABB is contained in the AABB of the trimming edge, which means that the point P is contained in the AABB of the trimming edge. Thus, it is unnecessary to examine the interior of a tabulated cylinder, as all points in the interior are contained in the AABB of the trimming edge.

Surfaces of Revolution

Surfaces of revolution are surfaces formed by revolving a line segment, arc, or curve around an axis line. Finding the interior of the surface is handled differently for each of these cases.

Revolution of a line: It will be shown that the AABB of a surface generated by a revolved line is the same as the AABB of the trimming edge.

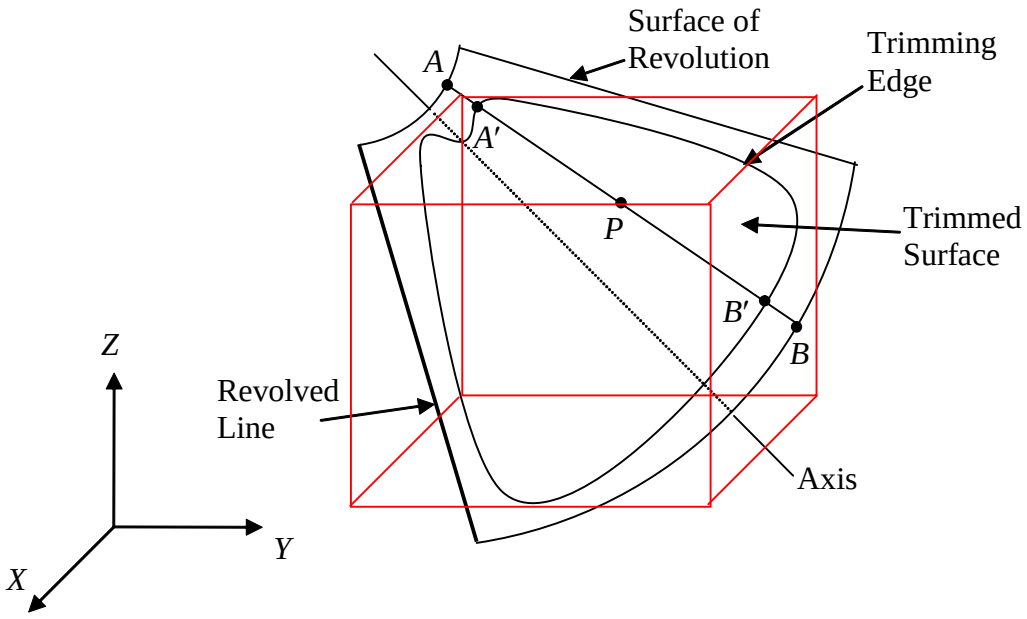


Figure 3.10. Trimmed Surface on a Surface of Revolution of a Line.

Referring to Figure 3.10 we take a point P in the interior of the trimmed surface. We then draw line segment $\overline{AB}$ through P that coincides with the revolved line as it passes through point P. $\overline{AB}$ intersects the trimming edge at points A' and B'. Thus, P is also on segment $\overline{A'B'}$. Since P is on this segment, it follows that it is contained in the AABB of this segment. Since the AABB of a line segment has the endpoints of the segment on opposite corners, we can deduce that one possible AABB containing P has

A' and B' as its opposite corners. Because A' and B' are points on the trimming edge, this AABB is contained in the AABB of the trimming edge, which means that the point P is contained in the AABB of the trimming edge. Therefore, it is unnecessary to examine the interior of the surface formed by the revolution of a line, as all points in the interior are contained in the AABB of the trimming edge.

Revolution of an arc: The revolution of an arc can create points that exceed the bounds of the trimming edge of a surface. It is, thus, necessary to find these points in order to calculate the AABB of the trimmed surface. The trimmed surface of revolution of an arc requires several entities. We start with the specification of a 2-D representation of an arc and its transformation matrix, given by Equation (1). The line that represents the revolution axis is given by its two endpoints. The trimming edges are specified in two forms. The first form is the actual representation of the edge in 3-dimensional space. However, a second representation of the edges is given in a plane in which one coordinate represents the angle of the position of a point on the arc, and the other represents the angles through which this point is revolved. The arc's angle represents the angles along the arc in its 2-D representation before it is transformed into 3-D space. The angle of revolution is measured by specifying the original arc as the point of 0° revolution. Positive revolution occurs counter-clockwise when looking along the axis from the axis's second endpoint to its first endpoint. An example of the two representations is shown in Figure 3.11. The figure gives the representations of two different trimmed surfaces that are generated from the revolution of the same arc. The first is represented by a bold trimming edge, the second by darker shading. In both arcs,

all of the points on the arc between the arc angles of –45° and 45° are revolved. In the 2-D representation, the first surface has a rectangular trimming edge; the second has a triangular trimming edge. Because the trimming edge for the first surface is rectangular, each of the points on the arc, including point *P*, is revolved from an angle of -60° to 0°. However, for the second surface, the angles of revolution vary according to the location of the point on the arc. For example, the point *P* is revolved from -60° to -45°. The 3-D representation shows what these surfaces look like in three-dimensional space. We use this 2-D representation to determine the domain in which to search to find extreme points for the bounds of the AABB.

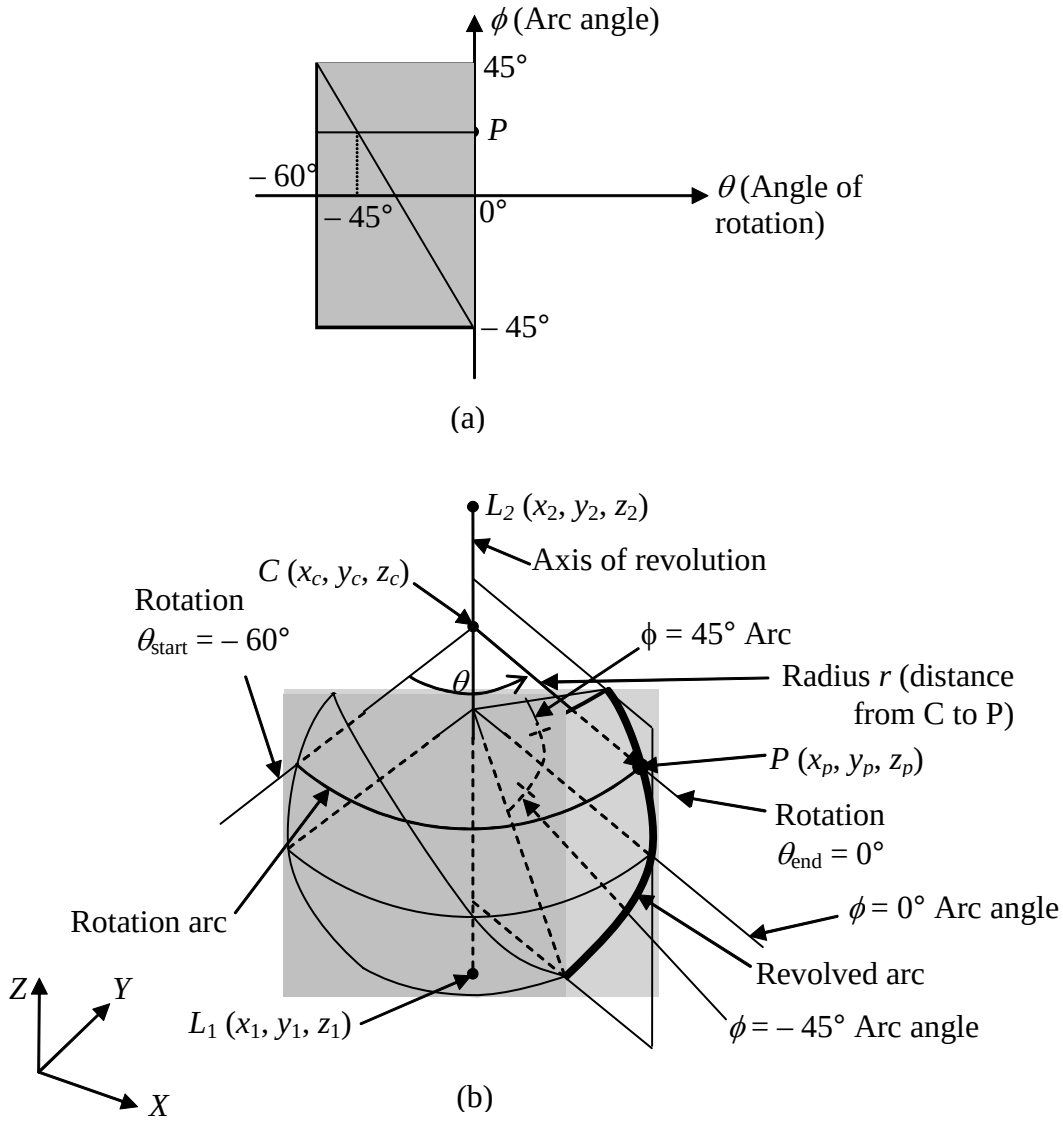


Figure 3.11. Rotated arc's (a) 2-D and (b) 3-D representations.

Consider a point on the revolved arc, which is associated with an angle measured on the revolved arc. The revolution of this point also forms an arc called the rotation arc. Consider this point as the 0° point on the circle containing the rotation arc. The arc's endpoints are given by the trimming edge. Determining the AABB of the rotation arc then gives an AABB based on the revolved arc angle. The extremes of this

function are then found using a binary search method to find the AABB of the entire trimmed surface.

To determine the AABB of the rotation arc, it is first necessary to find the center of the arc. IGES specifies the point on the revolved arc, P (x_p, y_p, z_p), and the endpoints L_1 (x_1, y_1, z_1), and L_2 (x_2, y_2, z_2) of the revolution axis. Using $C(x_c, y_c, z_c)$ as the center of rotation, the vector $\overrightarrow{PC}$ must be perpendicular to the vector $\overrightarrow{L_1L_2}$, or

$$(x_c - x_p)(x_2 - x_1) + (y_c - y_p)(y_2 - y_1) + (z_c - z_p)(z_2 - z_1) = 0 \tag{7}$$

In addition, C must be on the line $\overline{L_1L_2}$. Thus, C satisfies

$$\begin{aligned} x_c &= x_1 + e(x_2 - x_1) \\ y_c &= y_1 + e(y_2 - y_1) \\ z_c &= z_1 + e(z_2 - z_1) \end{aligned} \tag{8}$$

where e is a value to be determined. Let $d_x = x_2 - x_1$, $d_y = y_2 - y_1$, and $d_z = z_2 - z_1$. Then Equation (7) becomes

$$(x_1 - x_p + ed_x)d_x + (y_1 - y_p + ed_y)d_y + (z_1 - z_p + ed_z)d_z = 0$$

Solving for e, we obtain

$$e = \frac{(x_p - x_1)d_x + (y_p - y_1)d_y + (z_p - z_1)d_z}{d_x^2 + d_y^2 + d_z^2} \tag{9}$$

With this value of e, we can find the coordinates of C using Equation (8). The radius r of this arc is simply the distance between the points C and P, and is given by

$$r = \sqrt{(x_c - x_p)^2 + (y_c - y_p)^2 + (z_c - z_p)^2} \tag{10}$$

To find the AABB of the rotation arc, we formulate a 2-D representation of the arc and a transformation matrix to translate it into 3-Dimensional space. The 2-D

representation is constructed by setting the arc's *z*-value to 0 with the point (0, 0) its center. The beginning and end rotation angles from the 2-D representation of the trimming edge are used as the start and end angles of the rotation arc; θ_{start} and θ_{end}, respectively. The start and end points then have the coordinates

$$\left(r\cos\theta_{start}, r\sin\theta_{start}\right) \tag{11}$$

and

$$\left(r\cos\theta_{end}, r\sin\theta_{end}\right) \tag{12}$$

respectively, where *r* is the radius of the arc.

The transformation matrix transforms the point (0, 0, 0) to the point *C* (x_c, y_c, z_c) and the 0° point (*r*, 0, 0) to the point *P* (x_p, y_p, z_p). A third point out of plane is also needed to orient this coordinate system, so we also require the point (0, 0, 1) to translate to a point 1 unit away from *C* along the axis line toward the point L_2. We use Equation (1) as the way we translate these points. The translation of point (0, 0, 0) results in

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ 1 \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \tag{13}$$

Equation (13) gives the values of the translation terms of the transformation matrix in Equation (1). The transformation of the 0° point, (r,0,0), is

$$\begin{bmatrix} R_{11}r + x_c \\ R_{21}r + y_c \\ R_{31}r + z_c \\ 1 \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

Therefore,

$$R_{11} = \frac{x_p - x_c}{r}$$
$$R_{21} = \frac{y_p - y_c}{r}$$
$$R_{31} = \frac{z_p - z_c}{r}$$

To find the point a unit distance away from C toward L_2, we find the unit vector from L_1 to L_2. This is then added to the coordinates of C. We, thus, obtain the following equation that translates (0, 0, 1) to this point:

$$\begin{bmatrix} R_{13} + x_c \\ R_{23} + y_c \\ R_{33} + z_c \\ 1 \end{bmatrix} = \begin{bmatrix} x_c + \frac{d_x}{\sqrt{d_x^2 + d_y^2 + d_z^2}} \\ y_c + \frac{d_y}{\sqrt{d_x^2 + d_y^2 + d_z^2}} \\ z_c + \frac{d_z}{\sqrt{d_x^2 + d_y^2 + d_z^2}} \\ 1 \end{bmatrix}$$

Therefore,

$$R_{13} = \frac{d_x}{\sqrt{d_x^2 + d_y^2 + d_z^2}}$$
$$R_{23} = \frac{d_y}{\sqrt{d_x^2 + d_y^2 + d_z^2}}$$
$$R_{33} = \frac{d_z}{\sqrt{d_x^2 + d_y^2 + d_z^2}}$$

Now we must find the values of R_{12}, R_{22}, and R_{32}. In order to do so, we use the fact that the vector [R_{12} R_{22} R_{32}] is the cross-product of the vectors [R_{13} R_{23} R_{33}] and [R_{11} R_{21} R_{31}]. Thus, we get

$$R_{12} = R_{23}R_{31} - R_{21}R_{33} = \frac{d_y\left(z_p - z_c\right) - d_z\left(y_p - y_c\right)}{r\sqrt{d_x^2 + d_y^2 + d_z^2}}$$

$$R_{22} = R_{11}R_{33} - R_{13}R_{31} = \frac{d_z\left(x_p - x_c\right) - d_x\left(z_p - z_c\right)}{r\sqrt{d_x^2 + d_y^2 + d_z^2}}$$

$$R_{32} = R_{13}R_{21} - R_{11}R_{23} = \frac{d_x\left(y_p - y_c\right) - d_y\left(x_p - x_c\right)}{r\sqrt{d_x^2 + d_y^2 + d_z^2}}$$

Then, the transformation matrix becomes

$$[T] = \begin{bmatrix} \frac{x_p - x_c}{r} & \frac{d_y\left(z_p - z_c\right) - d_z\left(y_p - y_c\right)}{r\sqrt{d_x^2 + d_y^2 + d_z^2}} & \frac{d_x}{\sqrt{d_x^2 + d_y^2 + d_z^2}} & x_c \\ \frac{y_p - y_c}{r} & \frac{d_z\left(x_p - x_c\right) - d_x\left(z_p - z_c\right)}{r\sqrt{d_x^2 + d_y^2 + d_z^2}} & \frac{d_y}{\sqrt{d_x^2 + d_y^2 + d_z^2}} & y_c \\ \frac{z_p - z_c}{r} & \frac{d_x\left(y_p - y_c\right) - d_y\left(x_p - x_c\right)}{r\sqrt{d_x^2 + d_y^2 + d_z^2}} & \frac{d_z}{\sqrt{d_x^2 + d_y^2 + d_z^2}} & z_c \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (14)$$

With this transformation matrix, the AABB of the rotation arc can be found using the methods used to find the AABB of an arc, given in Section 3.2.1

Specifying a revolved arc angle results in a specific point being revolved. This revolution results in a specific rotation arc for which the bounds of its AABB can be found. Thus, the bounds of the rotation arc AABB are a function of the revolved arc angle. To find the AABB of the surface, the minimum and maximum bounds of all of these rotation arcs' AABBs must be found. This is done using a binary search algorithm, which is applied twice for each of the three orthogonal directions, once to find the minimum and once to find the maximum of the function over a certain interval.

To implement this algorithm, consider the function $y = f(x)$ where we wish to find the maximum $f(x)$ on the interval $x_{start} \le x \le x_{end}$. First, a fixed number n of

uniformly distributed subintervals is used to determine the intervals between the x_{start} and x_{end}: $\Delta x = (x_{end} - x_{start}) / n$. The following pseudocode describes the algorithm.

Table 3.2. Pseudocode for determining the maximum of $f(x)$ on the interval $x_{xtart} \le x \le x_{end}$

```
Δx = (x_end - x_start) / n
y_max = f(x_start)
x_c = x_start
if f(x_end) > y_max
        y_max = f(x_end)
        x_c = x_end
endif
for i = 1 to n – 1
        x = x_start + i * Δx
        if f(x) > y_max
                y_max = f(x)
                x_c = x
        endif
endfor
for i = 1 to N_iter
        Δx = Δx / 2
        if f(x_c + Δx) > y_max .and. f(x_c + Δx) > f(x_c - Δx)
                y_max = f(x_c + Δx)
                x_c = x_c + Δx
        elseif f(x_c - Δx) > y_max .and. f(x_c - Δx) > f(x_c + Δx)
                y_max = f(x_c - Δx)
                x_c = x_c - Δx
        endif
endfor
```

The resulting value y_{max} is approximately the maximum value of the function over the specified interval. The number of intervals and iterations ultimately determines its accuracy. The value of x_c is accurate within $[(x_{end} - x_{start}) / n] / 2^{N_{iter}}$. To find the minimum of a function, we modify the pseudocode given above as shown below.

Table 3.3. Pseudocode for determining the minimum of $f(x)$ on the interval $x_{xtart} \le x \le x_{end}$

```
Δx = (x_end - x_start) / n
y_min = f(x_min)
x_c = x_start
if f(x_end) < y_min
        y_min = f(x_end)
        x_c = x_end
endif
for i = 1 to n – 1
        x = x_start + i * Δx
        if f(x) < y_min
                y_min = f(x)
                x_c = x
        endif
endfor
for i = 1 to N_iter
        Δx = Δx / 2
        if f(x_c + Δx) < y_min .and.  f(x_c + Δx) < f(x_c - Δx)
                y_min = f(x_c + Δx)
                x_c = x_c + Δx
        elseif f(x_c - Δx) < y_min .and.  f(x_c - Δx) < f(x_c + Δx)
                y_min = f(x_c - Δx)
                x_c = x_c - Δx
        endif
endfor
```

To find the AABB of the surface, we apply the binary search algorithm to find the minimum and maximum bounds of the AABBs of all the rotation arcs. Because these AABBs are a function of the arc angle, we will be using θ to find the minimum or maximum bounds. This determination can be separated into six different functions: one for the minimum and one for the maximum of each of the *X*, *Y*, and *Z* directions. Each of these angles is found separately, using θ for *x*. The range of θ is found by taking the minimum and maximum values of the arc angle from the 2-D representation for the trimming edge. Thus, six angles will be found: θ_{xmin}, θ_{xmax}, θ_{ymin}, θ_{ymax}, θ_{zmin}, and θ_{zmax}. Each of these angles represents a point on the revolved arc that, when rotated, will yield

on its rotation arc the point on the surface with the maximum or minimum coordinate in the *x*, *y*, or *z* direction.

Revolution of a B-spline curve: The method used to find the AABB of the trimmed surface of revolution of a B-spline curve is similar to that of the revolution of an arc. As discussed previously, the rational B-spline curve is expressed as a function of the control points and a parameter *t*. The axis line is given by its two endpoints. The trimming edge is also given in 2-D and 3-D representations. The 3-dimensional representation is the actual trimming edge in 3-D space. The 2-D representation is very similar to the 2-D representation of the revolved arc. One coordinate represents the parameter *t* corresponding to the points on the curve, and the other represents the angles through which each point is revolved. An example of the two representations is shown in Figure 3.12. The 2-D representation shows that the part of the curve represented by *t* from 0.25 to 0.75 that are revolved with angles from -60° to 0°. The 3-D representation shows the entire B-spline curve and the portion corresponding to the *t* values from 0.25 to 0.75 that are revolved 60° clockwise around the axis line.

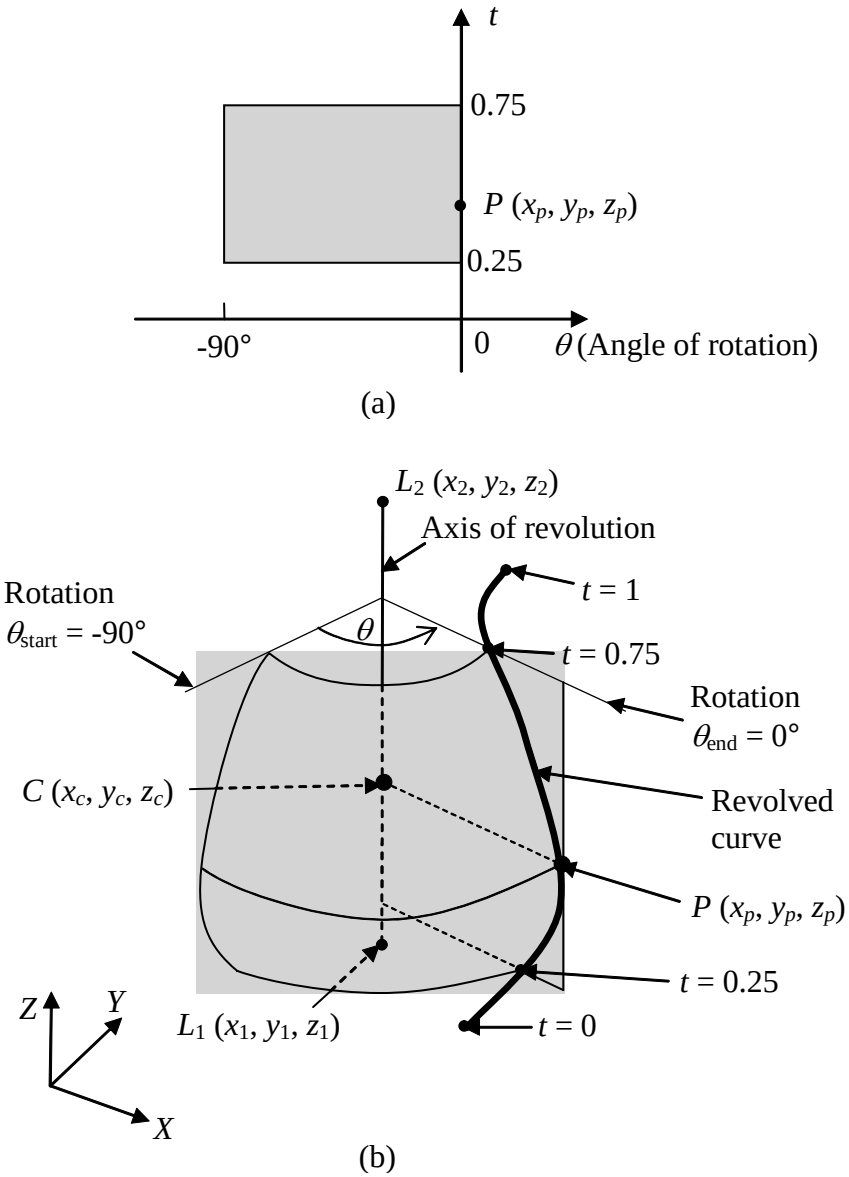


Figure 3.12. Rotated B-spline curve: (a) 2-D representation; and (b) 3-D representation.

The AABB of the trimmed surface is found in a manner similar to that of the revolved arc. Each point on the spline curve to be revolved corresponds to a value of t. Given a value of t, the corresponding point $P\ (x_p, y_p, z_p)$ on the curve can be found using the curve equation, Equation (4). With the point P and endpoints $L_1\ (x_1, y_1, z_1)$, and $L_2\ (x_2, y_2, z_2)$ of the revolution axis, the center of the rotation arc $C\ (x_c, y_c, z_c)$ is found using Equations (8) and (9). Equation (10) determines the radius r of the arc. We now

construct a 2-D representation of the arc, along with a transformation matrix. The starting and ending angles, θ_{start} and θ_{end}, are obtained from the 2-D representation of the trimming edge. Thus, for the 2-D representation, the center given by coordinates (0,0,0), and the start and end points are given by Equation (11) and Equation (12), respectively. The transformation matrix is determined by Equation (14). The AABB of the rotation arc can now be determined from the 2-D representation and the transformation matrix. A binary search, almost identical to that used to find the AABB of the revolved arc, is employed to determine the bounds for the trimmed surface. The AABB of the revolved arc is now a function of *t*, since *t* determines a point on the curve that, when revolved, gives a rotation arc. Thus, we perform six binary searches for the six values of *t* that result in the minimum or maximum values for bounds in the *X*, *Y*, and *Z* directions will be found. We then compare these bounds with those found for the AABB of the trimming edge of the surface. If the minimum bounds are lesser in value than those of the trimming edge's AABB, then we make the minimum bounds found in the binary search the new bound for the entire surface. The same process is used to determine the maximum bounds.

Rational B-Spline Surfaces

All surfaces not given as tabulated cylinders or surfaces of revolution are represented as rational B-spline surfaces, including planar surfaces. Planar surfaces are first degree surfaces and are dealt with separately. All higher-degree surfaces are determined by applying a smoothing function to arrays of control points. An example

of a rational B-spline surface is shown in Figure 3.13. We will now give the details for examining the interior of the trimmed surfaces of rational B-spline surfaces for extrema.

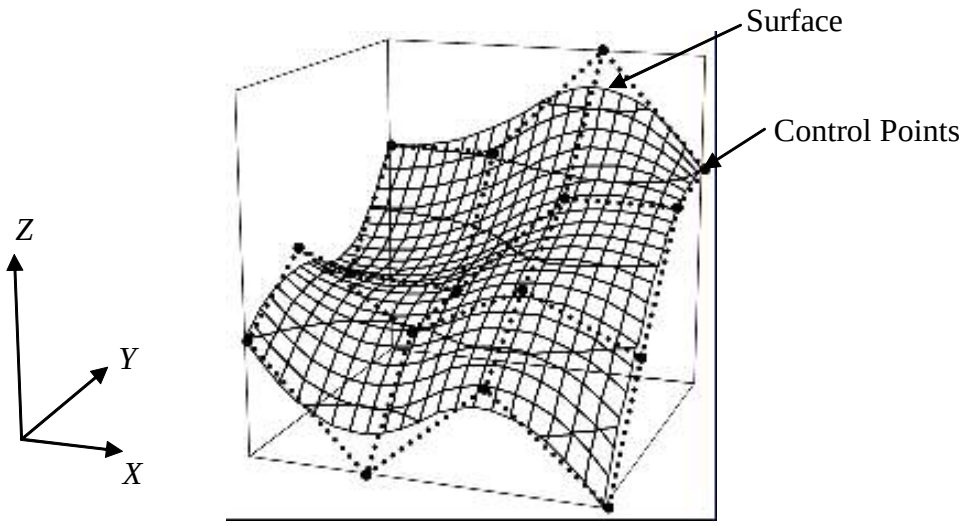


Figure 3.13. Rational B-Spline surface and its control points.

Planar Surfaces: We will show that all interior points of a plane are contained in the AABB of the edges of that plane and, therefore, there is no need to check for interior extrema points.

Take any point *P* in the surface as shown in Figure 3.14. Draw a line through *P* that lies on the plane. This line intersects the trimming edge, which is a closed curve, at points *A* and *B*. Since *P* is a point on the line segment $\overline{AB}$, it is located in the AABB that uses points *A* and *B* as its opposite corners. Because *A* and *B* appear on the trimming edge, this AABB is contained in the AABB of the entire trimming edge. Thus, point *P* is contained in the trimming edge AABB. Since *P* can be any point within the trimmed surface, it is unnecessary to check the interior for points that could change the trimmed surface's AABB.

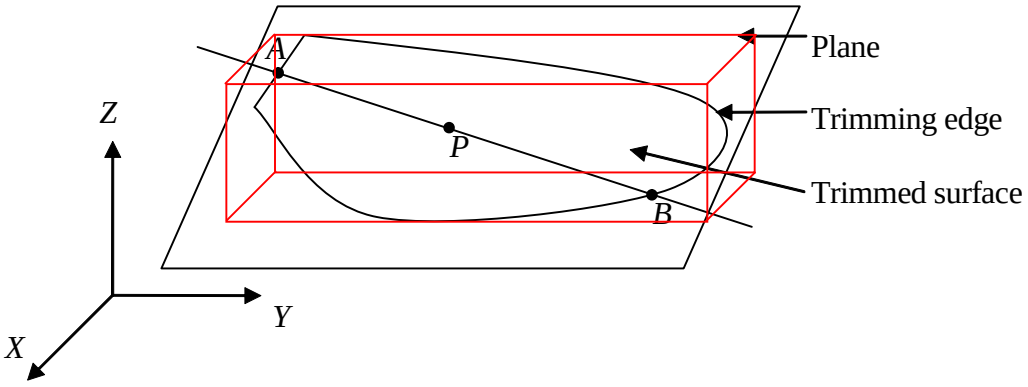


Figure 3.14. Trimmed surface of a plane.

Higher-degree surfaces: The points on the surface are calculated as functions with two parametric values, *s* and *t*. The function is given in the IGES 4.0 Specification (1988) as follows:

$$G^{(M_1)(M_2)}(s,t) = \frac{\sum_{i=0}^{K_1}\sum_{j=0}^{K_2} W_{i,j} P_{i,j} b_i^{(M_1)}(s) b_j^{(M_2)}(t)}{\sum_{i=0}^{K_1}\sum_{j=0}^{K_2} W_{i,j} b_i^{(M_1)}(s) b_j^{(M_2)}(t)}$$

where M_1 and M_2 are the order of the surface in *s* and *t*, respectively, K_1 and K_2 are the number of "rows" minus 1 and the number of "columns" minus 1, respectively, in the control point matrix, $W_{i,j}$ are the weights of each control point, $P_{i,j}$ are the control points, and $b_i^{(M_1)}(t)$ and $b_j^{(M_2)}(t)$ are functions determined by a knot sequences in *s* and *t*, respectively. The non-decreasing knot sequence in *s* is $s_{(-M_1)}, s_{(-M_1+1)}, \ldots, s_{(K_1+1)}$. In this sequence, $s_{(-M_1)} = s_{(-M_1+1)} = \ldots = s_0$ and $s_{(K_1-M_1+1)} = s_{(K_1-M_1+2)} = \ldots = s_{(K_1+1)}$. The non-decreasing knot sequence in *t* is $t_{(-M_2)}, t_{(-M_2+1)}, \ldots, t_{(K_2+1)}$. In this sequence, $t_{(-M_2)} = t_{(-M_2+1)} = \ldots = t_0$ and $t_{(K_2-M_2+1)} = t_{(K_2-M_2+2)} = \ldots = t_{(K_2+1)}$. In many commercial

CAD systems, $W_{i,j} = 1$ for $i = 0,\ldots,K_1$, $j = 0,\ldots,K_2$. The functions $b_i^{(M_1)}(s)$ and $b_j^{(M_2)}(t)$ are defined in Equations (4) and (5). The functions have the property that

$$. \sum_{i=0}^{K_1}\sum_{j=0}^{K_2} b_i^{(M_1)}(s) b_j^{(M_2)}(t) = 1$$

Thus, the equation reduces to

$$. G^{(M_1)(M_2)}(s,t) = \sum_{i=0}^{K_1}\sum_{j=0}^{K_2} P_{i,j} b_i^{(M_1)}(s) b_j^{(M_2)}(t)$$

This equation can be written as

$$G^{(M_1)(M_2)}(s,t) = \sum_{i=0}^{K_1} Q_i^{(M_2)}(t) b_i^{(M_1)}(s)$$

which is the equation for a spline curve of degree M_1 with control points $Q_i^{(M_2)}(t)$, where

$$Q_i^{(M_2)}(t) = \sum_{j=0}^{K_2} P_{i,j} b_j^{(M_2)}(t)$$

$Q_i^{(M_2)}(t)$ is also the equation for a spline curve of degree M_2 with control points $P_{i,j}$ at a fixed i. Thus, the points from $G^{(M_1)(M_2)}(s,t)$ can be calculated by taking, for each value of i, all control points of that i value and forming a spline curve of degree M_2 with these control points. The result is K_2 curves, each of which is a function of t. Those points on each curve corresponding to the desired value of t are then used as control points of another curve of degree M_1, which is a function of s. The given value of s will determine a point on this curve, which is the desired point on the surface. For example, in Figure 3.15, we want to determine the point corresponding to $s = 0.4$ and $t = 0.7$ on a surface with control points $P_{i,j}$, $i = 0, 1, 2, 3$, $j = 0, 1, 2, 3$, $M_1 = M_2 = 2$. First, spline

curves are created for the control points $P_{i,0}$, $P_{i,1}$, $P_{i,2}$, $P_{i,3}$, where i = 0, 1, 2, and 3. This results in four spline curves, $Q_i^{(2)}(t)$, i = 0, 1, 2, and 3. Then we find the point on each curve that corresponds to t = 0.7; namely $Q_i^{(2)}(0.7)$, i = 0, 1, 2, and 3. We then use these four points to construct another spline curve, which is a function of s. All of the points on this spline curve are points that occur on the surface. The curve is thus $G^{(2)(2)}(s,0.7)$. The point on this curve corresponding to s = 0.4 is the desired point on the surface.

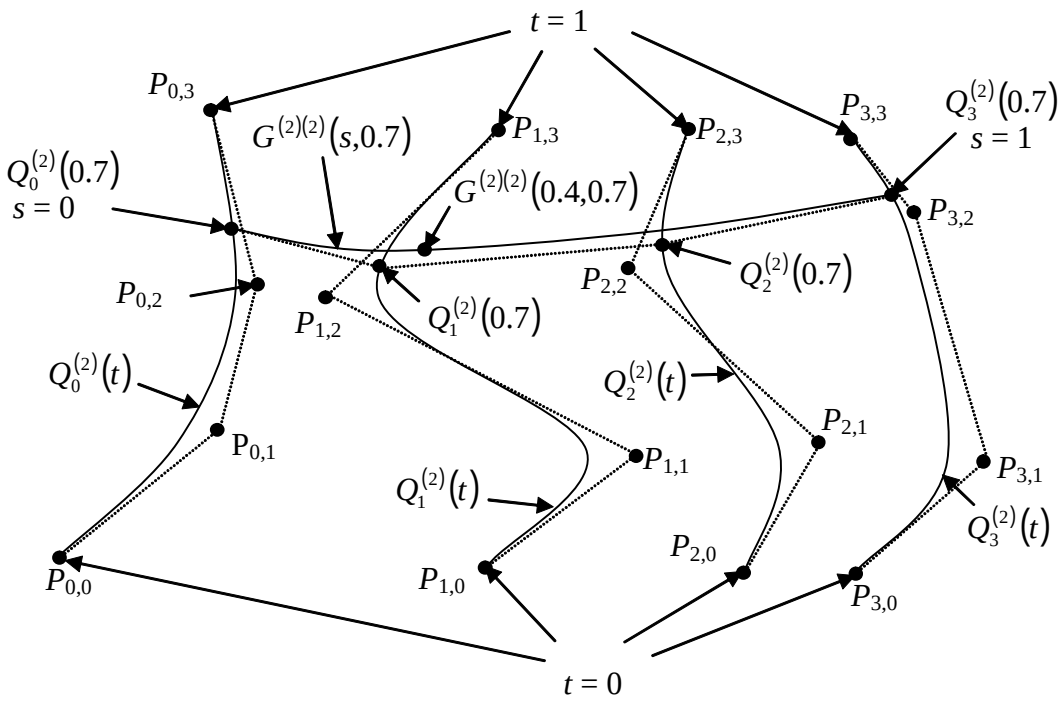


Figure 3.15. Determination of the point G(0.4, 0.7).

To find the AABB, an approach is used that is similar to those used to determine the surfaces of revolution of arcs and spline curves. For example, in Figure 3.15, setting t = 0.7 resulted in the curve $G^{(2)(2)}(s,0.7)$. This is analogous to a revolved surface, where fixing the arc angle or value of t gives the rotation arc of a point. The

AABB for this rational B-spline curve is found as described in Section 3.2.1, and is analogous to finding the AABB for the rotation arc for revolved surfaces. Thus, the result is a function that finds an AABB for a curve based on a value of *t*. It is then necessary to find the minimum and maximum bounds of the AABBs occurring for all values of *t* that correspond to the surface. Using, again, the binary search method mentioned previously, we find the six values of *t* that give the minimum and maximum bounds of AABBs in the three orthogonal directions. The AABB is defined by these six bounds.

3.3. Verification of AABB Determination

To verify the AABB determination, we applied the AABB determination algorithm to the part shown in Figure 3.16. In addition to plane surfaces, it includes a cylindrical surface, a hemispherical surface, and a B-spline surface. Notice that the part was designed so that the AABB determination must take into account these other types of surfaces. If the portions of the algorithm that deal with these surfaces are incorrect, then this will be reflected in an incorrect AABB for the part.

Th part was created in Pro/Engineer, converted into the IGES format, and then input to the AABB algorithm. The coordinates of the resulting AABB were then used to create a block part in Pro/Engineer to be placed on top of the current part. In this way, we are able to verify visually that the resulting AABB is correct. The results of this process are shown in Figure 3.17. In the orthographic views, a correct AABB will simply look like a rectangle drawn around the part, barely touching on the top, bottom, left, and right. The figure verifies that the calculated AABB is indeed correct. This

AABB was calculated with an average time over ten runs in 0.0155 ± 0.0005 seconds with a Pentium 4 2.4 GHz processor.

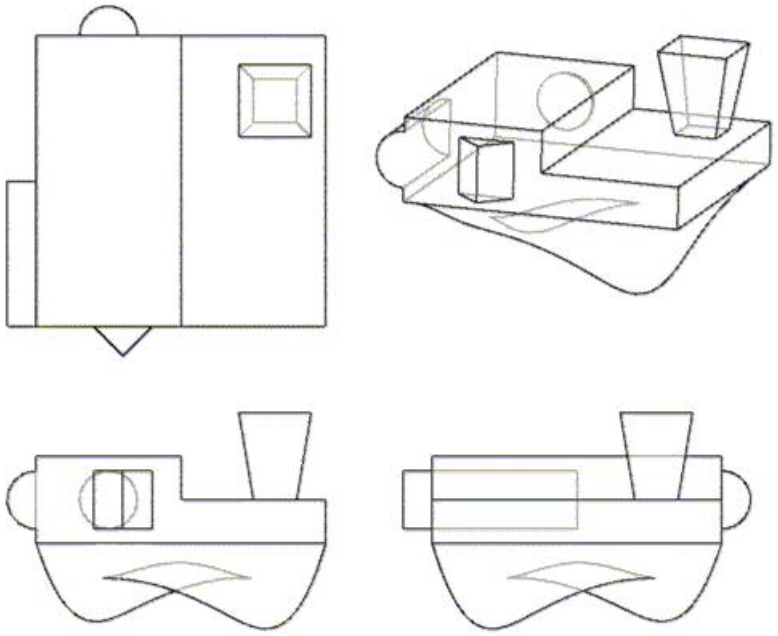


Figure 3.16. Orthographic and 3-D views of the part used to verify AABB determination in.

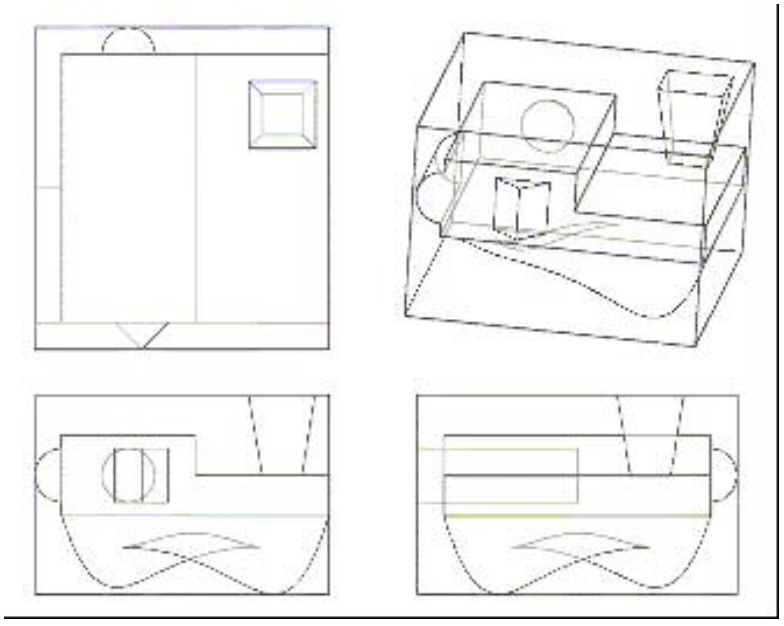


Figure 3.17. Part in Figure 3.16 enclosed by the AABB as calculated by the AABB determination algorithm.

3.4. Summary

In this chapter, various methods for finding the AABB for parts composed of different surfaces were discussed in detail. Pseudocode was given for the determination of the corners of the AABB of the part based on the AABBs of the part's surfaces. Then, the method for determining the surface AABB based on the trimming edge and surface interior was given. Next, the equations necessary to determine the AABB for the trimming edge were presented. Finally, equations were given that are used to adjust the AABB of the trimming edge so that the bounds of the AABB of the surface are obtained.

In this chapter, we have shown how to find the AABBs for each part in the assembly. In the next chapter, we present the determination of the relationships between these AABBs.

4. Finding contained AABBs

Now that we have the AABB of each part, we need to do some reasoning on the AABBs to find part visibility. The first step in this process is to determine the spatial relationships between these AABBs. The most important relationship between AABBs is for an AABB to be completely contained in another AABB, as shown in Figure 4.1. An AABB that is contained in another AABB is a candidate for being labeled invisible. Whether or not these candidates are actually invisible, however, is determined from the criteria discussed in Chapter 5.

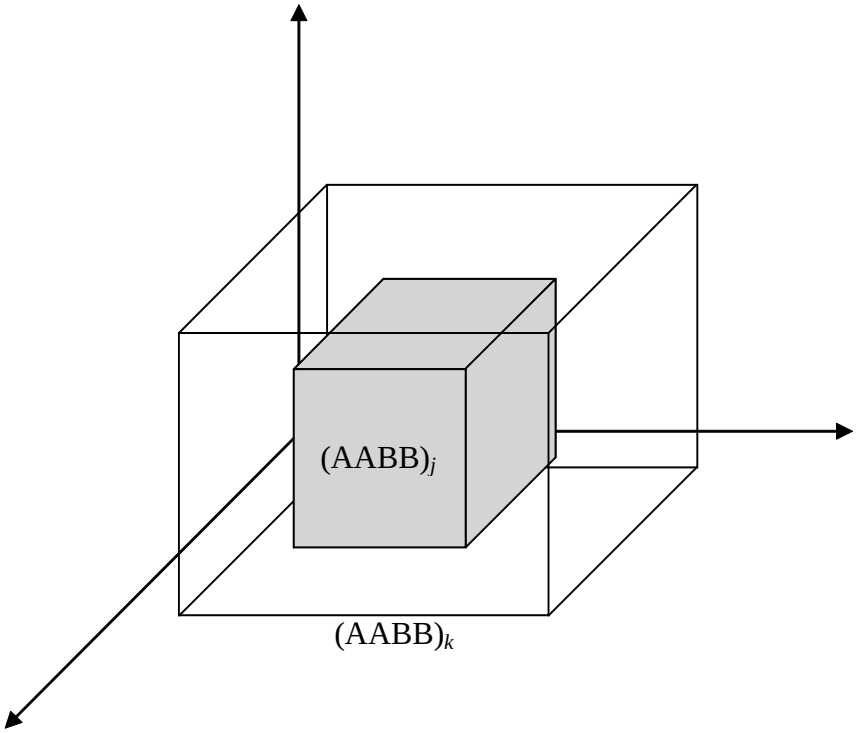


Figure 4.1. (AABB)$_j$ contained in (AABB)$_k$.

4.1. Definitions

In order to describe the algorithm that determines the visibility of AABBs, we first introduce several terms and symbols. These terms and symbols will then be used in this algorithm's description, which is provided in the subsequent sections.

The bounds of the AABB are uniquely determined by the coordinates of two diagonally opposite corners. Let the coordinates of two diagonally opposite corners of $(\text{AABB})_j$, shown in Figure 4.2, be $\left(X_j^{\min}, Y_j^{\min}, Z_j^{\min}\right)$ and $\left(X_j^{\max}, Y_j^{\max}, Z_j^{\max}\right)$, where $X_j^{\min} < X_j^{\max}$, $Y_j^{\min} < Y_j^{\max}$, and $Z_j^{\min} < Z_j^{\max}$.

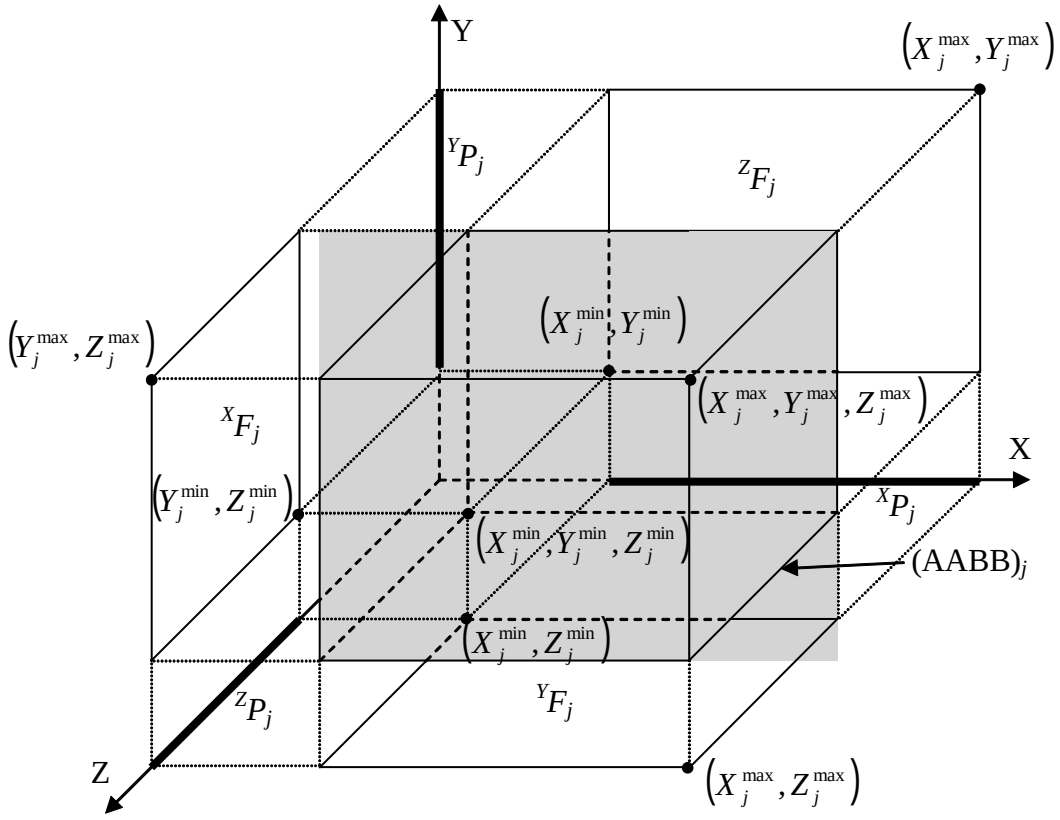


Figure 4.2. The three extents and projections of $(\text{AABB})_j$.

We first project each $(\text{AABB})_j$ onto the *XY*, *ZX*, and *YZ* planes. From these projections a minimum rectangle (extent) will be determined. As shown in Figure 4.2,

the projection of $(\text{AABB})_j$ onto their respective planes yields three rectangular extents: ${}^{X}F_j$, ${}^{Y}F_j$, and ${}^{Z}F_j$. Consider the extent ${}^{Z}F_j$. The coordinates of its diagonally opposed corners are $\left(X_j^{\min}, Y_j^{\min}\right)$ and $\left(X_j^{\max}, Y_j^{\max}\right)$ in the *XY* coordinate system. For the extent ${}^{Y}F_j$, $\left(X_j^{\min}, Z_j^{\min}\right)$ and $\left(X_j^{\max}, Z_j^{\max}\right)$ in the *XZ* coordinate system. Finally, for extent extent ${}^{X}F_j$, its diagonally opposed corners are $\left(Y_j^{\min}, Z_j^{\min}\right)$ and $\left(Y_j^{\max}, Z_j^{\max}\right)$ in the *YZ* coordinate system.

Projections of the extent ${}^{Z}F_j$ on the *X*- and *Y*-axes are ${}^{X}P_j$ and ${}^{Y}P_j$, respectively. Similarly, projections of the extent ${}^{Y}F_j$ on the *X*- and *Z*-axes are ${}^{X}P_j$ and ${}^{Z}P_j$, respectively. Finally, projections of the extent ${}^{X}F_j$ on the *Y*- and *Z*-axes are ${}^{Y}P_j$ and ${}^{Z}P_j$, respectively. These projections are shown in Figure 4.2.

4.2. Sorting coordinates

The relationships between AABBs can be determined based on the relationships between their projections. Thus, we will need to determine these projection relationships. In order to do this, we shall generate three lists of sorted coordinates, one in each of the coordinates *X*, *Y*, and *Z*. We will then use these lists to determine the relationships in a systematic and efficient manner. Afterwards, we determine from these lists those AABBs that are not completely contained within another AABB.

Consider the set of all the AABBs represented in the (X, Y, Z) coordinate system: $(\text{AABB})_j$, j = 0, 1, 2, ..., n – 1. First, we form three lists, one for each coordinate direction. In the X-direction, we form a list using the pairs $\left(X_j^{\min}, X_j^{\max}\right)$. This list looks like:

$$\left\{X_0^{\min}, X_0^{\max}, X_1^{\min}, X_1^{\max}, \ldots, X_{n-1}^{\min}, X_{n-1}^{\max}\right\}$$

In a similar manner, we form a second list in the Y- direction using the pairs $\left(Y_j^{\min}, Y_j^{\max}\right)$ and a third list in the Z- direction using $\left(Z_j^{\min}, Z_j^{\max}\right)$.

Consider the list in the X-direction. Determination of whether or not one AABB contains another requires that the list of the values $X_j^{\min}$, $X_j^{\max}$ be sorted in ascending order, starting with the numerically smallest coordinate value of $X_j^{\min}$ and $X_j^{\max}$, j = 0, 1, 2, ..., n – 1 and ending with the largest numerical value. If there are any groups of coordinate points where their values are equal, then the values within these groups must be arranged so that the relationships between their projections are determined correctly. To accomplish this, each group of values is separated into two subgroups, one with all $X_j^{\max}$ values and the other with all $X_j^{\min}$ values. Within each of these subgroups, the values are sorted from greatest value to the least value of the other coordinate in the projection. Thus, the $X_j^{\max}$ values are sorted in descending order of the corresponding $X_j^{\min}$ values and vice versa. If the values of the other coordinates between two projections are equal, then the relationship between the two corresponding projections is noted as “same” and it does not matter how these values are arranged with respect to each other. Finally, the subgroup containing all of the $X_j^{\max}$ is placed in the list before

the subgroup containing all the $X_j^{\min}$. This overall sort results in a sorted list we call L_X. In the next section, we explain why these groups of values are arranged in this manner. Note that there are exactly two coordinates from each projection ${}^{X}P_j$ in the ordered pair sorted list L_X. The first coordinate is the least coordinate $X_j^{\min}$ and the second one is its greatest coordinate $X_j^{\max}$. These two coordinates are not necessarily next to each other in the sorted list L_X, and may include between them other coordinate points. For example, $X_k^{\min}$ and/or $X_k^{\max}$ of projection ${}^{X}P_k$ may be between $X_j^{\min}$ and $X_j^{\max}$. This sorting procedure is also performed on the *Y*-coordinate list and the *Z*-coordinate list, resulting in lists L_Y and L_Z, respectively.

As an example, consider a collection of projections ${}^{X}P_j$, j = 0, 1, …, 5 shown in Figure 4.3. The projections have been arbitrarily offset from the axis to make their viewing easier. To start, we have the list for these projections as

$$L_x = \left\{X_0^{\min}, X_0^{\max}, X_1^{\min}, X_1^{\max}, X_2^{\min}, X_2^{\max}, X_3^{\min}, X_3^{\max}, X_4^{\min}, X_4^{\max}, X_5^{\min}, X_5^{\max}\right\}$$

This list must then be sorted numerically. For those values that are not equal, the sorting procedure is straightforward. However, we must determine the order of those values for which the coordinate values are equal. Thus, we have to determine the order of the groups { $X_1^{\min}, X_3^{\max}, X_4^{\min}$, and $X_5^{\min}$ } and { $X_1^{\max}$ and $X_4^{\max}$ }. Let us start with the first group. Because $X_3^{\max}$ is the only maximum coordinate, it is placed first. The remaining $X^{\min}$ values are sorted based on the corresponding values of $X_1^{\max}, X_4^{\max}$, and $X_5^{\max}$. In this case, we notice that $X_5^{\max} > X_1^{\max} = X_4^{\max}$. Since $X_5^{\max}$ is the largest, it is the next sorted value. It remains to be determined the order between $X_1^{\max}$ and $X_4^{\max}$.

Since these two values are equal, we find that ${}^{X}P_1$ and ${}^{X}P_4$ are identical in the values of their endpoints. Thus, we mark these two as "same", where we keep track of the relationships between projections. Since we have already determined the relationship between these two projections, the order between $X_1^{\min}$ and $X_4^{\min}$ does not matter. The same goes for the order between $X_1^{\max}$ and $X_4^{\max}$. We, therefore, assign their orders arbitrarily. Thus, the first subset has the order $X_3^{\max}, X_5^{\min}, X_1^{\min}$, and $X_4^{\min}$. The second subset will have the order $X_4^{\max}$ and $X_1^{\max}$. As a result, the sorted list for this set of projections is

$$L_x = \left\{X_3^{\min}, X_2^{\min}, X_3^{\max}, X_5^{\min}, X_1^{\min}, X_4^{\min}, X_5^{\max}, X_0^{\min}, X_4^{\max}, X_1^{\max}, X_2^{\max}, X_0^{\max}\right\}$$

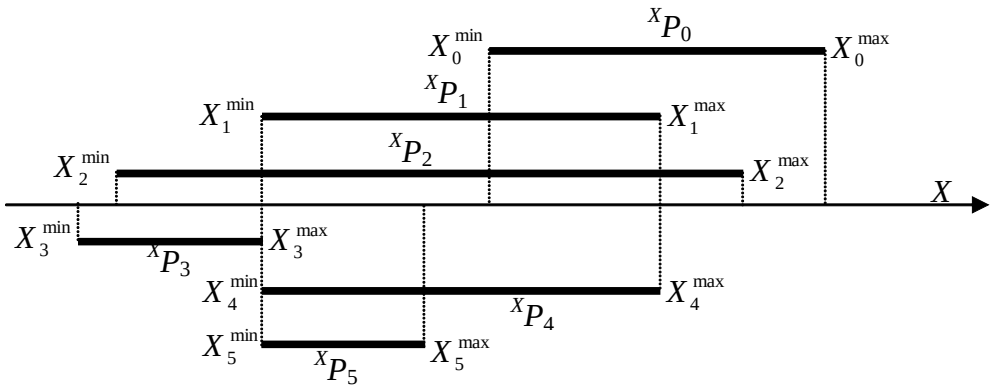


Figure 4.3. Projections and their coordinates.

4.3. Determining contained boxes

When one AABB contains another, we have a certain relationship between the two AABBs' projections. As can be seen in Figure 4.4, to have $(\text{AABB})_j$ contained in $(\text{AABB})_k$, ${}^{X}P_k$ must contain ${}^{X}P_j$, ${}^{Y}P_k$ must contain ${}^{Y}P_j$, and ${}^{Z}P_k$ must contain ${}^{Z}P_j$. This relationship holds true if one or two pairs of projections are the "same" instead of

having a containing relationship. When all three projections are identical, the AABBs are the "same." We use these rules to determine whether or not $(AABB)_j$ is contained in or is the "same" as $(AABB)_k$.

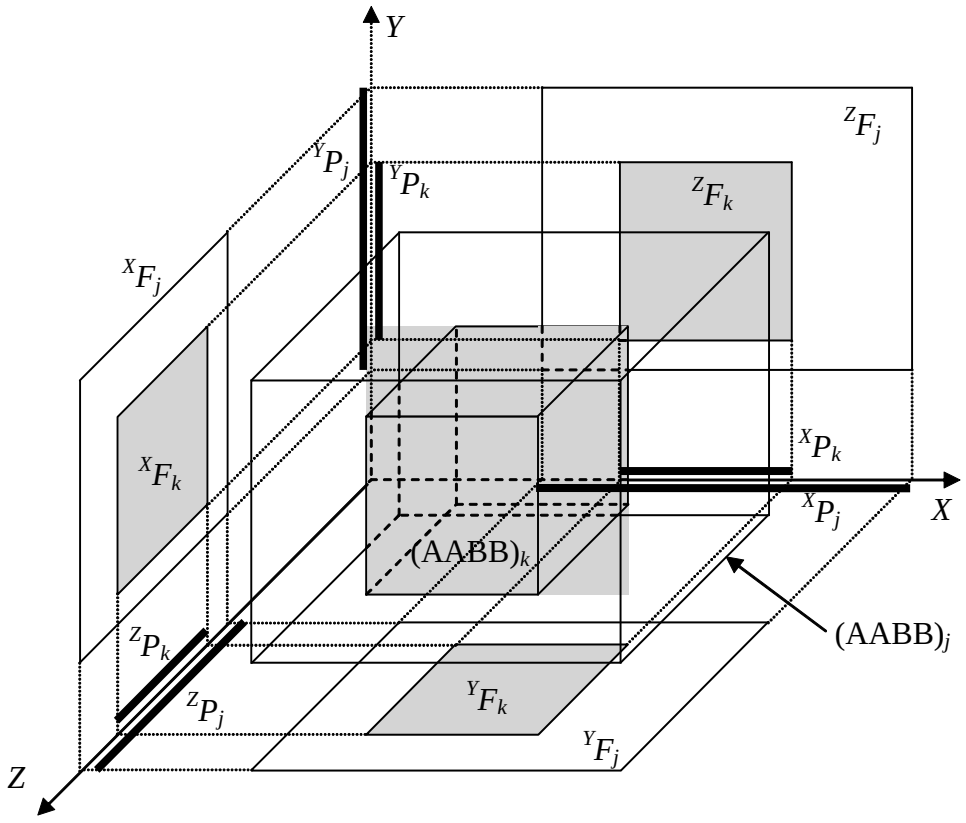


Figure 4.4. Extents and projections of $(AABB)_j$ and $(AABB)_k$. $(AABB)_j$ contains $(AABB)_k$.

To make these determinations, we determine which projections are contained in other projections. We arbitrarily start in the *X*-direction. Given two projections $^{X}P_j$ and $^{X}P_k$, the containing relationship is determined using the following rule:

> If both $X_j^{\min}$ and $X_j^{\max}$ of $^{X}P_j$ lie between $X_k^{\min}$ and $X_k^{\max}$ of $^{X}P_k$ in the list L_X, and $^{X}P_j$ is not the "same" as $^{X}P_k$, then $^{X}P_k$ contains $^{X}P_j$.

This is shown in Figure 4.5.

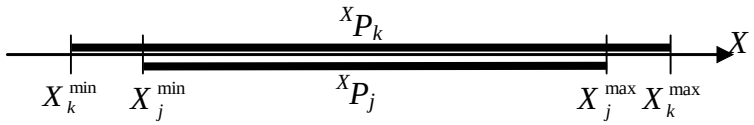


Figure 4.5. Projection $^{X}P_k$ containing projection $^{X}P_j$.

An efficient algorithm for parsing the list L_X to determine the containing relationships is needed, since individually checking each pair of projections is an inefficient process. Most pairs will be disjoint, so checking these pairs would be a waste of processing time. The current research uses a method that involves a one-time pass through of each list, which only makes necessary comparisons.

In this method, the list L_X is scanned to find projections that contain other projections. This is much like traveling along the axis and passing through each projection, as shown in Figure 4.6. This is sufficient because L_X is simply a list of the endpoints of all of the projections in the order in which they are encountered. For this method, each projection has two states, "open" and "closed". An "open" state occurs when we have traveled to a point in between the endpoints of a projection, or in terms of L_X , after we have encountered the minimum endpoint of a projection but before we reach its maximum endpoint. A projection is "closed" for all other cases. For example, in Figure 4.7, projection $^{X}P_k$ is "open" because the current traveled point lies between $X_k^{\min}$ and $X_k^{\max}$, but projection $^{X}P_j$ is "closed" because the point is beyond $X_j^{\max}$.

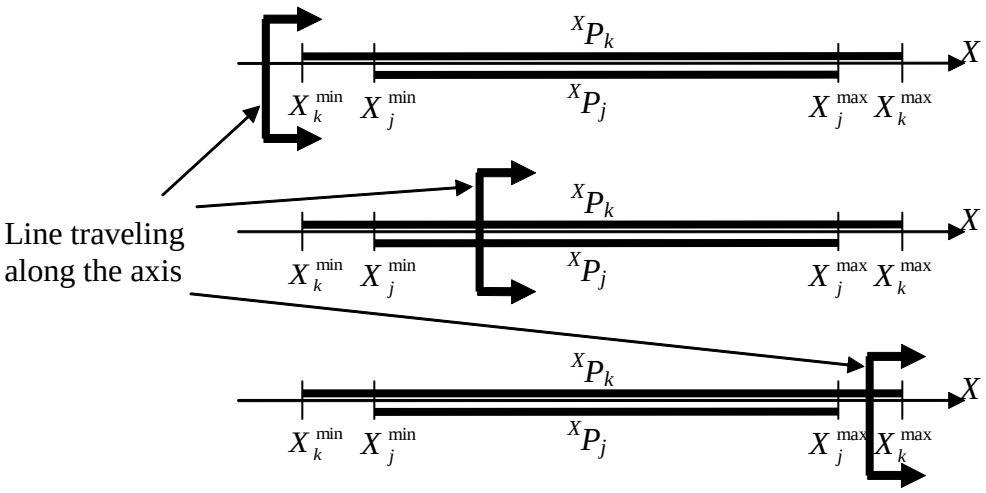


Figure 4.6. Traveling along an axis.

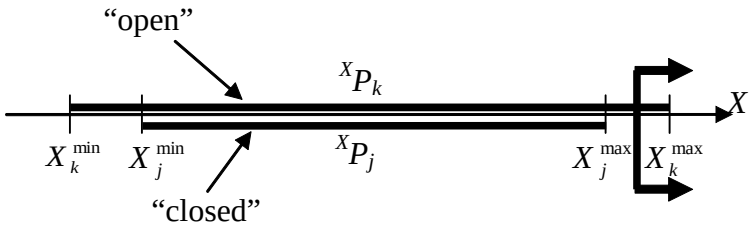


Figure 4.7. Example of an "open" and "closed" projection.

To start, all projections are given a "closed" status. We will be "opening" and "closing" them as we travel along the axis by going through L_X. Starting from the least value in L_X, whenever an $X_j^{\min}$ value is encountered, the corresponding projection $^{X}P_j$ is changed to an "open" status. The location in L_X of this endpoint is also recorded. When an $X_j^{\max}$ is encountered, the corresponding projection $^{X}P_j$ is changed to a "closed" status, since this is the end point of the projection. In addition, at this point, we check all those projections that are "open" to determine their relationships to $^{X}P_j$. In doing this, all that is necessary to check is if the "open" projections' start point locations precede the current projection's start point. Those "open" projections that satisfy this condition and are not the "same" as the current projection (determined previously), are noted as

containing the current projection. This is because the start points of the "open" projections are before the current projection's start point, and because, by the very nature of the projections being "open", the end points of those projections come after those of the current projection. An example of projections that satisfy this condition are shown in Figure 4.7. Projection ${}^{X}P_j$ has just been "closed", as we have just passed $X_j^{\max}$. Since ${}^{X}P_k$ is still "open", we check the order of $X_j^{\min}$ and $X_k^{\min}$. As $X_k^{\min}$ comes before $X_j^{\min}$, we have determined that ${}^{X}P_j$ is contained in ${}^{X}P_k$.

For an example of the scanning process, we use the projections in Figure 4.3, the list L_X for these projections was determined as

$$L_x = \left\{X_3^{\min}, X_2^{\min}, X_3^{\max}, X_5^{\min}, X_1^{\min}, X_4^{\min}, X_5^{\max}, X_0^{\min}, X_4^{\max}, X_1^{\max}, X_2^{\max}, X_0^{\max}\right\}$$

We start with all of the projections "closed". Then, in order, we open ${}^{X}P_3$ and ${}^{X}P_2$ and close ${}^{X}P_3$. When ${}^{X}P_3$ is closed, ${}^{X}P_2$ is the only open projection. However, $X_2^{\min}$ follows $X_3^{\min}$, so there is no containing relationship between ${}^{X}P_2$ and ${}^{X}P_3$. Since $X_3^{\max}$ is followed by $X_5^{\min}$, $X_1^{\min}$, and $X_4^{\min}$, we open ${}^{X}P_5$, ${}^{X}P_1$, and ${}^{X}P_4$. Next, we encounter $X_5^{\max}$; therefore, we close ${}^{X}P_5$. When we do this, we check all the open projections' minimum coordinates, $X_1^{\min}$, $X_2^{\min}$, and $X_4^{\min}$, and find that only $X_2^{\min}$ comes before $X_5^{\min}$. Thus, ${}^{X}P_5$ is contained in ${}^{X}P_2$. Proceeding along L_X, we next encounter $X_0^{\min}$ and $X_4^{\max}$, which leads us to open ${}^{X}P_0$ and close ${}^{X}P_4$. The coordinates of the projections ${}^{X}P_3$ and ${}^{X}P_2$ are such that $X_2^{\min}$ and $X_3^{\min}$ come before $X_5^{\min}$. However, ${}^{X}P_2$ and ${}^{X}P_5$ were previously determined as the "same" in the sorting process, so there can be no containing relationship between them. Thus, we determine that ${}^{X}P_5$ is contained in only

${}^{X}P_3$. Continuing through the rest of the list, we find that the only other containing relationship is that ${}^{X}P_2$ is contained in ${}^{X}P_3$. These are easily verified visually in Figure 4.3.

The pseudocode for this process is given in Table 4.1.

Table 4.1. Pseudocode for finding projections found in other projections.

```
for i = 0 to n - 1
        status(XP_i) = "closed"
endfor
for i = 0 to 2n - 1
        if L_X [i] is an X_j^min for any value of j (j is determined by the
        value of X_j^min found)
                status(XP_j) = "open"
                place[j] = i
        else if L_X [i] is an X_j^max for some value of j
                status(XP_j) = "closed"
                for k = 0 to n – 1
                        if status(XP_k) = "open"
                                if XP_k is not the "same" as XP_j
                                        if place[k] < place[j]
                                                XP_k contains XP_j
                                        endif
                                endif
                        endif
                endfor
        endif
endfor
```

The preceding process is also used to parse the lists L_Y and L_Z. Before doing this, it is possible to rule out AABBs that will not be able to contain or be contained in another AABB. In order for an AABB to have one of these relationships, its projections must contain, be contained in, or be the same as the other AABB's projections in each of the three directions. Thus, if we are parsing list L_Y after L_X, we can remove all those projections from L_Y that correspond to projections in L_X that do not have any of these

relationships with any other projection in L_X. Also, before parsing L_Z, we can remove all projections from L_Z that were removed from L_Y plus those that were determined not to have one of these relationships from parsing L_Y. However, this elimination process is not entirely necessary, as the list parsing process is already very fast. Averaging ten runs, the average time to parse the three lists without elimination for 490 parts is .030 ± .0046 seconds on a Pentium 4 2.4 GHz machine. Although there might be a speed advantage, it is easier not to do any removal and parse each list independently.

After the "containing" and "same" relationships between the projections have been found in the three directions, the "containing" and "same" relationships between the AABBs are found using the previously discussed requirement. This is the requirement that a "containing" relationship between AABBs has a "containing" or "same" relationship between each corresponding pair of the projections, with a "same" relationship resulting between the AABBs when the relationships in all three directions between the projections are the "same."

4.4. Other relationships

Although this research does not require other relationships, minor additions to the method presented can be made to find other relationships with a very small increase in computation time. These relationships include "disjoint," "adjacent," and "intersecting". Disjoint AABBs are those that do not share any points in common. Pairs of AABBs are adjacent if they have common points on their boundaries and share no interior points. AABBs intersect if they share some common interior points, but also have points that are not shared with each other.

To find these relationships, we must first find the disjoint, adjacent, and intersecting relationships among the projections. Adjacent projections are those that share a common endpoint, where one projection ends and the other starts, as shown in Figure 4.8. These adjacent points are found during the sort of the lists L_X, L_Y, and L_Z, where the values that are equal are found. When these equal values are sorted amongst each other based on the other coordinate values in the projections, the relationships between all projections that are starting and those that are ending can be marked "adjacent." This means that in the *X*-direction, for example, for all values *j* and *k* such that $X_j^{\max} = X_k^{\min}$, ${}^XP_j$ is adjacent to ${}^XP_k$.

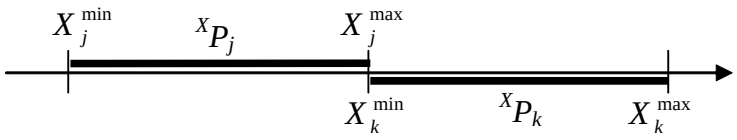


Figure 4.8. Projection ${}^XP_j$ adjacent to ${}^XP_k$.

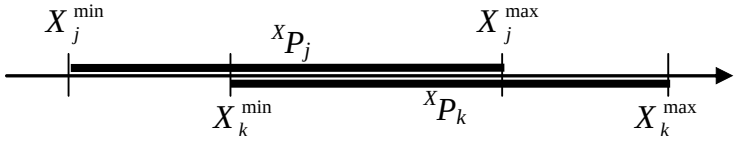


Figure 4.9. Projection ${}^XP_j$ intersecting ${}^XP_k$.

Intersecting projections are those that have one projection start after the other starts, but also end after the other ends, as shown in Figure 4.9. This is found by adding a conditional statement to the L_X parsing algorithm shown in Table 4.1. The additions are shown in bold in the modified pseudocode given in Table 4.2.

Table 4.2. Pseudocode modified to also find projections that intersect other projections.

```
for i = 0 to n - 1
        status(^X P_i) = "closed"
endfor
for i = 0 to 2n - 1
        if L_X [i] is an X_j^min for some value of j (j is determined by
                                                the value of X_j^min found)
                status(^X P_j) = "open"
                place[j] = i
        else if L_X [i] is an X_j^max for some value of j
                status(^X P_j) = "closed"
                for k = 0 to n – 1
                        if status(^X P_k) = "open"
                                if ^X P_k is not the "same" as ^X P_j
                                        if place[k] < place[j]
                                                ^X P_k contains ^X P_j
                                        else if place[k] > place[j]
                                                ^X P_k intersects ^X P_j
                                        endif
                                endif
                        endif
                endfor
        endif
endfor
```

When a projection "closes," the start points of all "open" projections are checked. All "open" projections will have their end coordinates greater than the endpoint of the projection that was just "closed." Those "open" projections that start earlier enclose the "closed" projection. Those that start later will end up intersecting the current projection.

The last relationship that can occur between projections is the "disjoint" relationship. This relationship occurs when two projections have no common points as shown in Figure 4.10. These relationships are determined by default, since any pair of

projections whose relationships have not yet been determined is categorized as "disjoint". Any pair with at least one point in common has already been categorized.

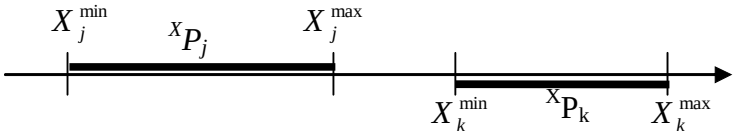


Figure 4.10. Projection ${}^{X}P_j$ is disjoint with ${}^{X}P_k$

Once all the relationships between all the projections have been determined, they are used to determine the AABB relationships according to the following rules:

1) If ${}^{X}P_j$ is disjoint with ${}^{X}P_k$, ${}^{Y}P_j$ is disjoint with ${}^{Y}P_k$, or ${}^{Z}P_j$ is disjoint with ${}^{Z}P_k$, then $(\text{AABB})_j$ is disjoint from $(\text{AABB})_k$.
2) If ${}^{X}P_j$ is not disjoint (meaning adjacent to, intersecting, containing, contained in) from ${}^{X}P_k$, ${}^{Y}P_j$ is not disjoint from ${}^{Y}P_k$, and ${}^{Z}P_j$ is not disjoint from ${}^{Z}P_k$, and at least one of these relationships is "adjacent," then $(\text{AABB})_j$ is adjacent to $(\text{AABB})_k$.
3) If ${}^{X}P_j$ intersects, contains, or is contained in ${}^{X}P_k$, ${}^{Y}P_j$ intersects, contains, or is contained in ${}^{Y}P_k$, ${}^{Z}P_j$ intersects, contains, or is contained in ${}^{Z}P_k$, and $(\text{AABB})_j$ is not containing or contained in $(\text{AABB})_k$, then $(\text{AABB})_j$ intersects $(\text{AABB})_k$.

For AABBs to be disjoint, it is only required that one pair of corresponding projections be disjoint. An example of this is shown in Figure 4.11. Only projections ${}^{X}P_j$ and ${}^{X}P_k$ are disjoint. Since each projection represents the *X*, *Y*, or *Z* coordinates of the points in the AABB, any pair of corresponding projections that are disjoint will correspond to two AABBs that have no points with the same coordinates; that is, they will have no common points.

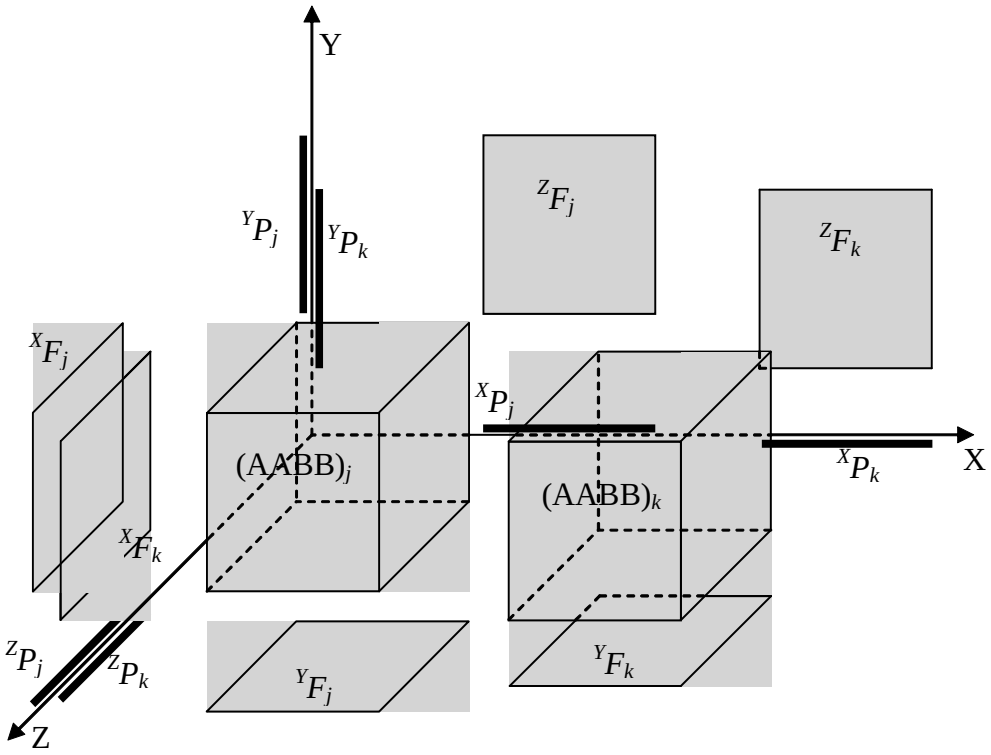


Figure 4.11. Two disjoint AABBs along with their extents and projections.

Two adjacent AABBs are shown in Figure 4.12, where it is seen that the projections, $^{X}P_j$ and $^{X}P_k$ are adjacent. Examining the non-disjoint AABBs, it is seen that a pair of corresponding adjacent projections means that all of the common points between the AABBs will have the same values for one of its coordinates. In other words, all of the common points will occur on the borders of the AABBs. Thus, the two AABBs are adjacent.

All other combinations of AABB projection relationships will result in both common and uncommon interior points. This means that these AABBs intersect.

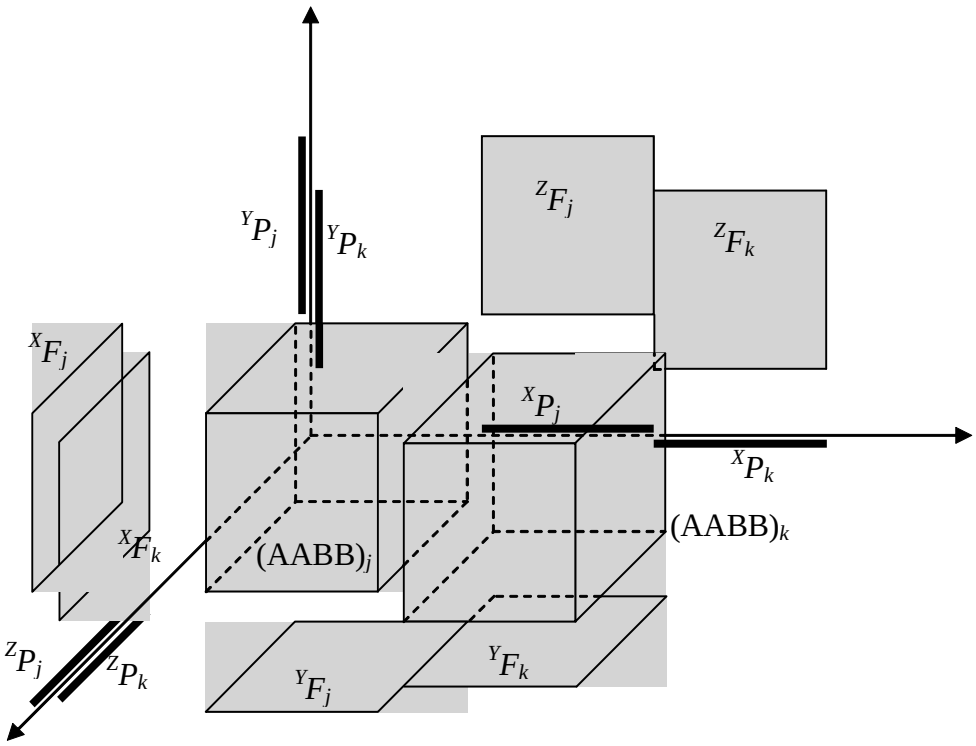


Figure 4.12. Two adjacent AABBs along with their extents and projections.

4.5. Summary

In this chapter, the procedure for finding which AABBs contain other AABBs was developed. First, definitions of extents and projections were given. Next, the manner in which lists of the coordinates of the projections were sorted was detailed. Then, based on these lists, the determination of AABB containment was performed. Finally, we presented extensions that can be made to the procedure to detect other relationships between AABBs, which may be helpful in other applications.

The next chapter develops the algorithm for the determination of the visibility of parts based on the results of this chapter.

5. Determining part visibility

Up to this point, we have determined the minimum AABB around a part and the relationships between AABBs. We will now use this information to determine part visibility.

Visibility can be defined in both static and dynamic terms. Static visibility means that the decision of whether or not to render a part never changes as soon as the parts are loaded into the computer's memory to display. In this situation, those parts that are invisible are those that will not be displayed under any circumstances unless the configuration of the parts, part geometry or relative positions, are changed. This cuts down on both memory requirements and processing time. Processing time does increase because it is necessary to determine which parts are visible when loading the parts into memory. However, this only happens once, and there can be a much more substantial savings in processing time during viewing because there are fewer parts to render.

Dynamic visibility pertains to those situations where the determination of what to render is determined by changing conditions during display, namely viewing angle. In this situation, all parts are loaded into memory. The savings occurs in viewing. When a certain angle for viewing is desired, the computer determines what is visible from this angle and then displays it. Processing time increases because of the visibility determination, but this is far outweighed by the savings that occurs from not having to render what is determined to be invisible.

A combination of these two approaches is what is most desired and can be achieved easily, as they are almost entirely independent of each other. The static

approach basically eliminates all those parts that are never visible during viewing. The subset that remains are all those parts that could be visible. At this point, we use the dynamic approach on the subset, which can greatly reduce processing time.

The following algorithm focuses on the static approach, as that is what will give the greatest contribution to the viewing of large assemblies. Dynamic (viewing angle) approaches have already been studied by other researchers (Bittner et al., 1998; Hudson et al., 1997; Kumar et al., 1996; Levi et al., 1999; Möller and Haines, 1999; Zhang and Hoff, 1997; Zhang et al., 1997).

The algorithm to be introduced determines the visibility of the AABB. Those parts that are determined to be visible will be displayed. It is noted that the method described below can determine whether or not a part is partially or completely surrounded by other parts. When completely surrounded by other parts, the part is invisible, as there are no views from which one can see the part. An example is shown in Figure 5.1, where the interior block is completely enclosed by the other blocks. Real world examples include the pistons in an engine, the picture tube in a television, and the girders in a skyscraper. These parts in their assemblies are completely hidden, not being able to be seen from any angle.

Invisibility is determined by examining the AABB of each part. Acknowledging that there are exceptions to this, we will assume that an AABB that is completely surrounded by multiple AABBs is invisible. This is not the same as an AABB being contained in another single AABB, as was determined in Chapter 4. The visibility determination of these contained AABBs is dealt with separately, as they are not necessarily invisible.

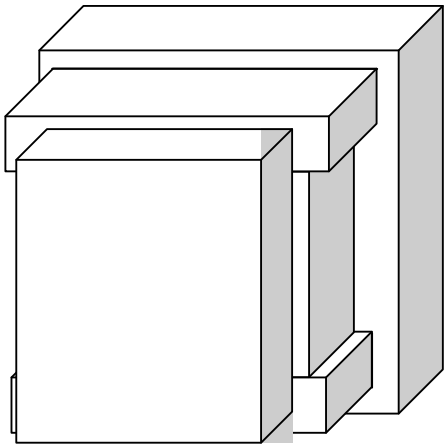


Figure 5.1. Example of a completely enclosed part.

5.1. Visibility determination process

Visibility is determined in two stages. First, the visibility of all AABBs not contained in another AABB is evaluated. That means that we basically take all those AABBs determined as contained by the method described in the previous chapter and remove them from consideration for the first stage. This first stage involves taking multiple cross sections and tracing around the exterior of each section. This is detailed in the Section 5.3. The second stage is then the visibility determination of these contained AABBs. This is based on the visibility determinations of the first stage and is detailed in Section 5.4. The flowchart for the process is given in Figure 5.2 and the process is described in the subsequent sections.

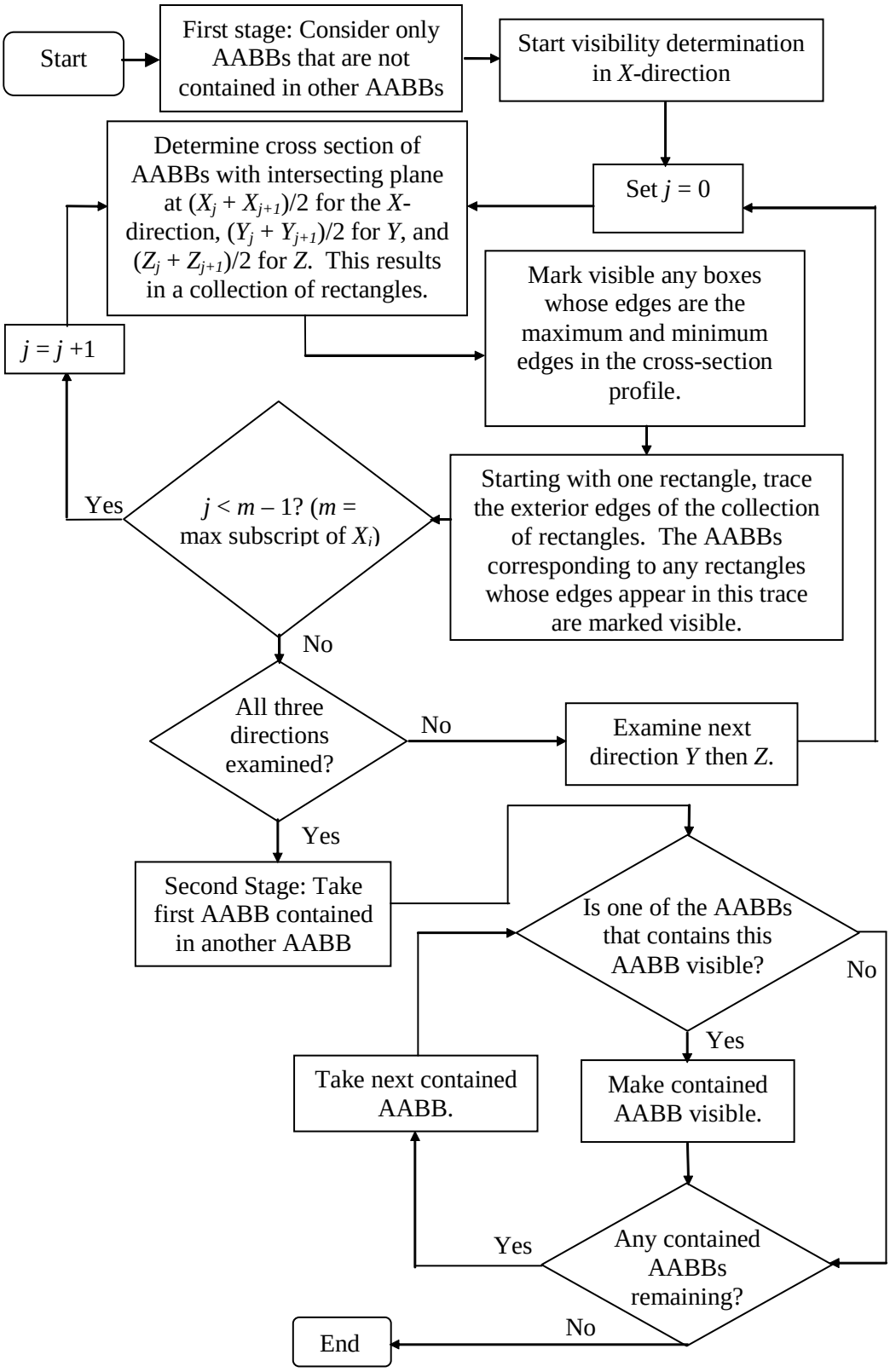


Figure 5.2. Flow chart for determining of the visibility of an AABB

5.2. Advantages of a cross-section trace for non-contained AABBs

The cross-section trace used in the algorithm to be presented has many advantages over other avenues that were explored during the research for determining visibility. First, methods were considered that would reason with the complex geometry of the mechanical parts to determine visibility. However, all of these methods involved the calculation of intersections of complex surfaces, which required large computation time. It was, therefore, concluded that determining the AABBs of parts first and then basing visibility calculations on them would be a much more computationally efficient approach.

One option for determining visibility is the ray trace. In this method, several rays are projected from various points in space. Each point represents a viewpoint, with the rays representing lines of sight. The first AABB encountered by each ray is determined to be visible. The main drawback with this method is that a more accurate result requires more computation time. To increase the accuracy of ray tracing, more viewpoints are needed and more rays per viewpoint are needed. If there is an AABB that is visible only from a small region of space at a certain angle, a large number of rays must be traced from a large number of viewpoints in order to have a good chance for ray tracing to detect it. However, there is always a possibility that an AABB that should be detected as visible is not. Thus, to get accurate results, ray tracing can be computationally costly. It also does not take advantage of the unique geometry that AABBs have when compared to more complex geometries.

To avoid the ray tracing approach, a cross section approach is considered. The cross section allows us to simplify the visibility calculations to visibility along the

outside of a “slice” of the AABBs, changing the problem to a two-dimensional one. Also, the greatest level of accuracy can be attained by a finite number of cross sections; that is, increasing the number of cross sections after a certain point will not increase accuracy. This is because of the geometry of the AABBs. There can be only a certain number of cross sections that are distinct from each other. Any more cross sections taken after this will result in duplicate information. Analyzing identical cross sections makes no contribution to visibility detection. This is an advantage over ray tracing, where it is unknown when the results become accurate. The only assurance of accuracy with ray tracing occurs when all of the AABBs have been detected as visible. In any other situation, it will be difficult to be sure that an AABB is invisible.

To analyze a cross section, it is also possible to use a ray tracing approach in a two dimensional case, tracing rays from exterior points to the configuration of rectangles that results from a cross section. The first rectangle encountered is marked visible. However, this approach has the same shortcomings as the three-dimensional approach, as greater accuracy requires more rays traced from more points. Thus, an edge trace approach was adopted to take advantage of the rectangular geometry that results from the cross sections. In addition, the exterior edge trace is an easy way to determine what boxes would be determined visible through a 2-D ray trace, as all rays in the ray trace would first intersect exterior edges, which would occur on the exterior edge trace. In this way, visibility of the rectangles in a section can be found with greater accuracy and in less time than with the ray trace.

5.3. Visibility of non-contained AABBs

We now describe the determination of the visibility of non-contained AABBs, which is the first stage. The visibility determination of contained AABBs will be dependent on the results of this first stage determination.

Consider the set of all AABBs that are not contained in another AABB and call this set *A*. Assume that there are *n* AABBs in this set, and let $(\text{AABB})_j$ be one of the members of the set ($1 \le j \le n$). As described in Section 4.1, (AABB)*j* has two coordinates associated with each direction: $X_j^{\min}$ and $X_j^{\max}$ for the *X*- direction, $Y_j^{\min}$ and $Y_j^{\max}$ for the *Y*-direction, and $Z_j^{\min}$ and $Z_j^{\max}$ for the *Z*-direction. The visibility algorithm uses the sorted lists L_X, L_Y, and L_Z that have been determined as described in Section 4.2.

We start with the *X*-direction. Consider the list L_X, which is composed of the sorted values of $X_j^{\min}$ and $X_j^{\max}$ and may contain some groups with equal values. A new list ${}^{S}L_X$ is created that contains only the distinct values of *X*. Thus, only one value from each group of equal values in L_X is included in ${}^{S}L_X$. Therefore, the new list ${}^{S}L_X$ consists of only the *X* values such that ${}^{S}L_X = \{X_0, X_1 \ldots X_m\}$, where $X_0 < X_1 < \ldots < X_m$, $m \le 2n-1$, and *m* is the number of distinct values.

The method requires that, for the *X*-direction, we take cross sections of the set *A* by sequentially generating a series of *YZ*-planes. The location of these planes will be discussed subsequently. A cross-section is the resulting intersection between each of these planes and the set *A*. Taking a cross-section of the AABBs results in a profile of rectangles. In Figure 5.3, two configurations of AABBs are each intersected by one plane. The resulting rectangular profiles ${}^{X}R(c)$ are shown in Figure 5.4. The numbering

of the rectangles in the figure is arbitrary and only for the purposes of explanation. At each of the *YZ* planes, the cross-section profiles are ${}^{X}S(c)$, where c is the X-coordinate where the cross-section is generated. The profile of each intersected $(\text{AABB})_j$ is a rectangle ${}^{X}R_j(c)$, which has the same coordinates as its extent, ${}^{X}F_j$, as defined in Section 4.1.

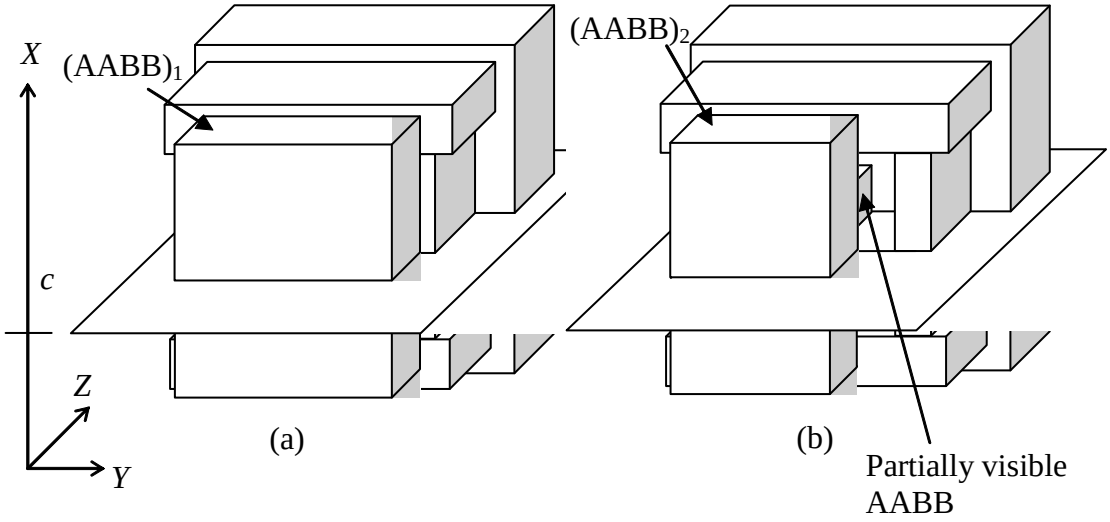


Figure 5.3 Plane intersecting collections of AABBs. Both configurations are the same except that $(\text{AABB})_1$ in (a) is replaced with $(\text{AABB})_2$ in (b).

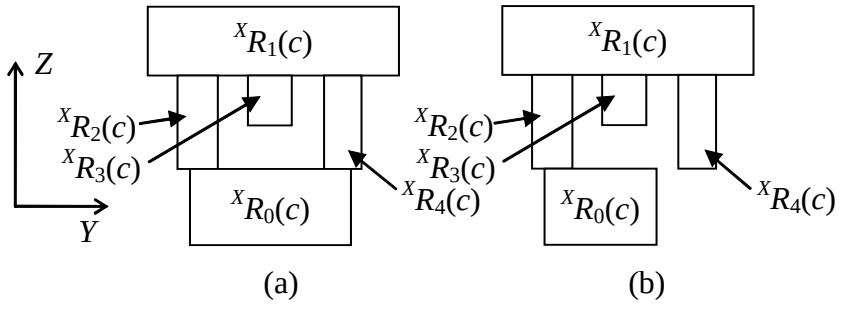


Figure 5.4. Cross section profiles, ${}^{X}S(c)$, from (a) Figure 5.3a and (b) Figure 5.3b.

We will now discuss the determination of the values of c at which to take cross sections. To simplify this discussion, we will assume we are taking cross sections perpendicular to X, but the determination of where to take cross sections in Y and Z will be the same. The list ${}^{S}L_X$ we formed represents the list of all of the X coordinates where

two cross-section profiles, each immediately taken in opposite directions on the X-axis, will be different from each other. (Profiles are different when one of the profiles contains a rectangle from one AABB that is not present in the other one.) This leads to the property that two cross-section profiles that both occur between the same two successive entities in ${}^{S}L_X$ will have identical profiles, meaning that the profiles contain rectangles from the same AABBs. In other words:

I. Any two cross-section profiles ${}^{X}S(c)$ and ${}^{X}S(d)$ are different when $X_{k-1} < c < X_k$, $X_k < d < X_{k+1}$, and X_{k-1}, X_k, X_{k+1} are all successive members of ${}^{S}L_X$.

II. Any two cross-section profiles ${}^{X}S(c)$ and ${}^{X}S(d)$ are identical when $X_k < c$, $d < X_{k+1}$, where X_k and X_{k+1} are successive members of ${}^{S}L_X$.

To demonstrate this, we take the configuration of boxes shown in Figure 5.3a as an example. From the front view (the positive Z viewing angle), the AABB configuration looks as shown in Figure 5.5. The list ${}^{S}L_X$ for this configuration consists of the values X_0, X_1, X_2, …, X_9. Taking a cross section in X results in one of the nine cross sections in Figure 5.6. Notice that it does not matter what the value of c is between the two values at which a cross-section is taken, since in this range of c the cross-section is the same. Thus, we will take only one cross-section for each interval between successive values of ${}^{S}L_X$. This was the purpose of forming the list ${}^{S}L_X$, which tabulates the locations of only the cross-sections that need to be examined. For convenience, cross sections are taken at the mid-distance of each interval, that is at $d_k = (X_k + X_{k+1}) / 2$, where $k = 0, 1, 2, ..., m - 1$ and m is the number of entries in ${}^{S}L_X$.

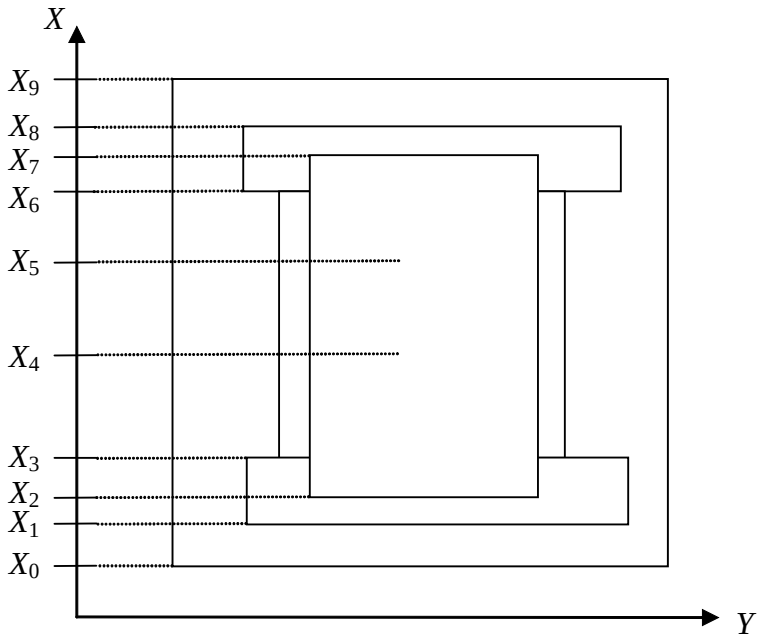


Figure 5.5. View of front (positive Z viewing angle) of AABBs in Figure 5.3a.

The algorithm determines visibility of the AABBs using the profiles ${}^{X}S(d_k)$. Since each rectangle is associated with an AABB, we associate the visibility of the AABB with the rectangle's visibility. If the rectangle ${}^{X}R_j(d_k)$ is visible, then its associated $(\text{AABB})_j$ and corresponding part are also visible. We now give a description of how to determine the visibility of the rectangles in a profile.

A cumulative approach is taken to determine visibility. Initially, all AABBs are marked invisible. Then, during the analysis, if an AABB is marked visible at any point, it is ultimately visible. At no time can an AABB that was marked visible be subsequently marked invisible.

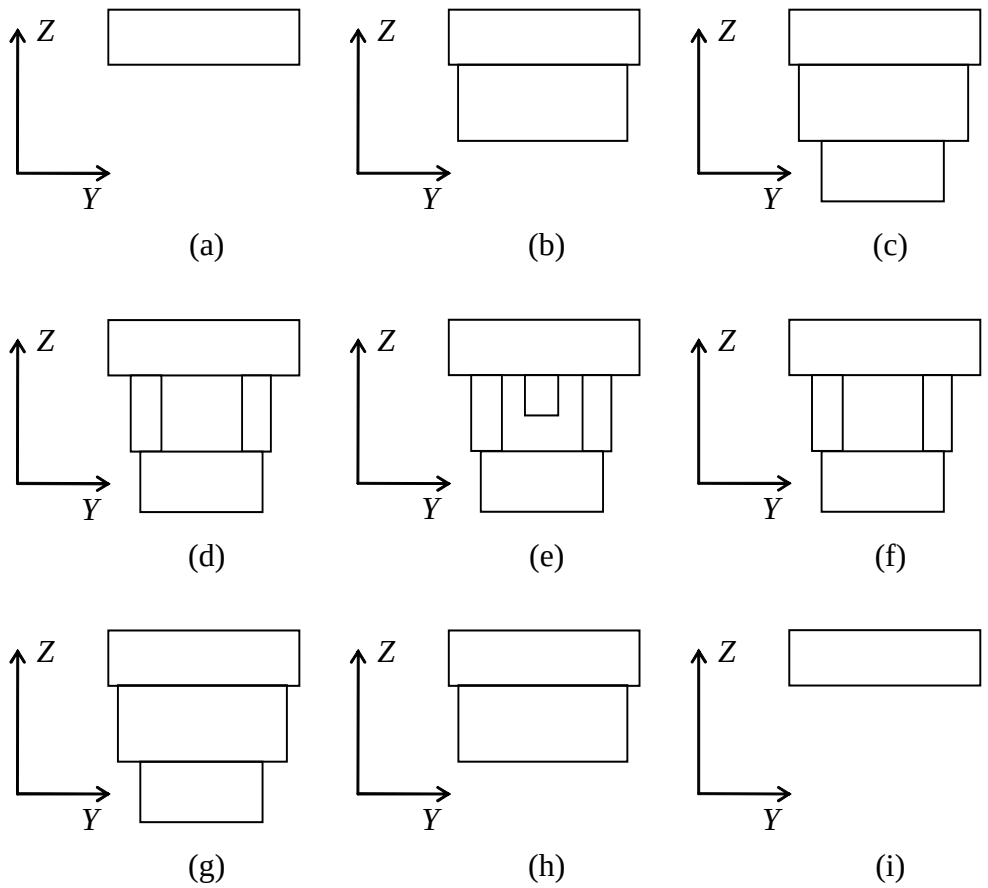


Figure 5.6. The various cross-sections of the AABBs in Figure 5.3a, taken between (a) X_0 and X_1, (b) X_1 and X_2, (c) X_2 and X_3, (d) X_3 and X_4, (e) X_4 and X_5, (f) X_5 and X_6, (g) X_6 and X_7, (h) X_7 and X_8, and (i) X_8 and X_9.

In examining a profile, if all of the AABBs of the rectangles in that profile were previously marked visible, no further analysis is needed in that profile. This is because the analysis can only determine if a previously invisible AABB should be marked visible. Analyzing profiles with AABBs that are all visible is not necessary, and the analysis moves to the next cross section profile. In addition, in analyzing a profile, it is necessary to keep track of which rectangles were marked visible. Thus, we create a separate record of visibility in this profile. In this separate record, we use the same approach. We start with all the rectangles invisible, and then mark them visible as the analysis proceeds. When analysis of the profile has finished, we transfer the visibility

of the rectangles in the profile to the visibility of the AABBs. This simply involves making all those AABBs visible whose corresponding rectangles are visible.

A profile is analyzed as follows. Those rectangles with the minimum and maximum coordinates in Y and Z are marked visible. For example, in both Figure 5.4a and Figure 5.4b, ${}^{X}R_0(d_k)$ and ${}^{X}R_1(d_k)$ are marked visible. ${}^{X}R_0(d_k)$ is visible because it has the minimum value in the Z-direction. ${}^{X}R_1(d_k)$ is visible because it has the minimum and maximum values in the Y-direction and the maximum value in the Z-direction.

The algorithm determines which rectangles have these properties as follows. Consider a profile ${}^{X}S(d_k)$. A rectangle ${}^{X}R_j(d_k)$, one of the m rectangles in this profile, has diagonal corners of $\left(Y_j^{\min}, Z_j^{\min}\right)$ and $\left(Y_j^{\max}, Z_j^{\max}\right)$. The rectangle ${}^{X}R_j$ is visible if one of the following four conditions is met:

Condition I: $Y_j^{\min} = \min\left(Y_q^{\min}\right)$,

Condition II: $Z_j^{\min} = \min\left(Z_q^{\min}\right)$,

Condition III: $Y_j^{\max} = \max\left(Y_q^{\max}\right)$,

Condition IV: $Z_j^{\max} = \max\left(Z_q^{\max}\right)$, $q = 0, 1, 2, \ldots, m - 1$

To find which rectangles to make visible, we use the lists L_Y and L_Z (recall Section 4.2). In each of these lists, the minimum and maximum values in the profile will occur as the first and last terms of the list. The first value, or group of equal values, pertains to the minimum value. Similarly, the last value, or group of equal values, pertains to the maximum value. Thus, all of the rectangles corresponding to these values are marked visible. For example, Figure 5.7a has the sorted list L_Y

$$L_Y = \left\{Y_5^{\min}, Y_1^{\min}, Y_2^{\min}, Y_0^{\min}, Y_2^{\max}, Y_3^{\min}, Y_6^{\min}, Y_3^{\max}, Y_5^{\max}, Y_4^{\min}, Y_0^{\max}, Y_4^{\max}, Y_6^{\max}, Y_1^{\max}\right\}$$

Only rectangles 0 through 4 occur in this profile. The order of those values pertaining to rectangles 5 and 6 are only placed for explanation purposes. Thus, to find the visible rectangles, we find the first value from the profile. This is $Y_1^{\min}$ and, therefore, $(AABB)_1$ is visible. If $Y_2^{\min}$, the next value, were equal to $Y_1^{\min}$, then $(AABB)_2$ would also be visible. The last value from the profile is $Y_1^{\max}$. This value also determines that $(AABB)_1$ is visible. Since $Y_6^{\max}$ is not in the profile, if $Y_4^{\max}$ were equal to $Y_1^{\max}$, it would indicate that $(AABB)_4$ were visible as well.

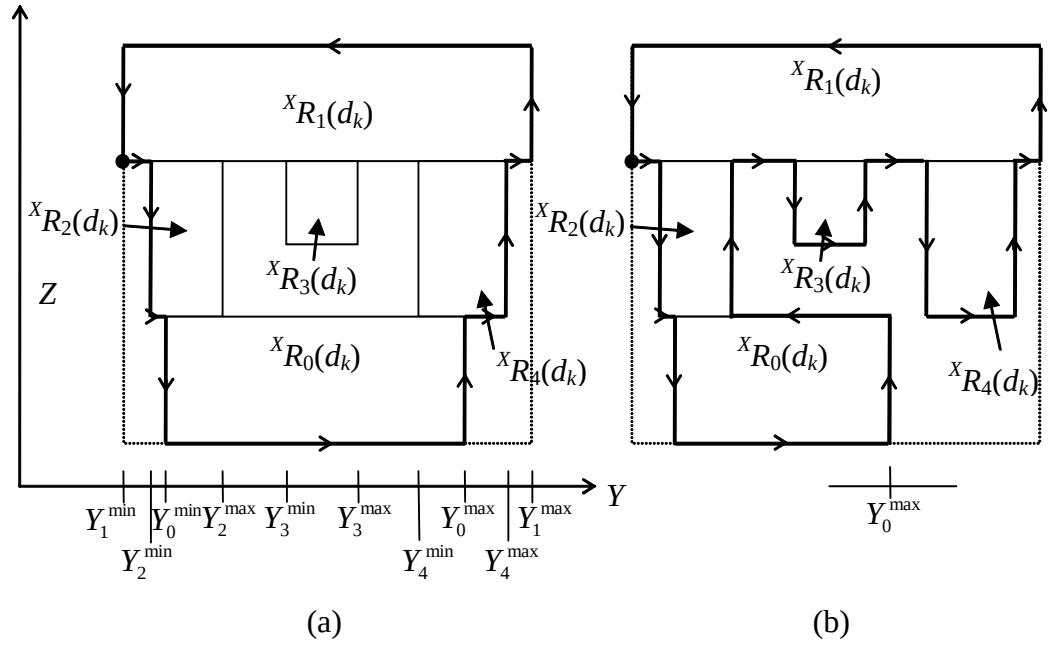


Figure 5.7. Boundary traces of the profiles in (a) Figure 5.4a, and (b) Figure 5.4b.

For the next stage of the algorithm, an edge trace is performed around the exterior of the profile $^{X}S(d_k)$, and any rectangle whose edges coincide with this edge trace are marked visible. Edge traces are shown in bold in Figure 5.7 for both profiles in Figure 5.4. The trace is performed in a counterclockwise (CCW) manner on the profile starting with one of the visible rectangles matching Condition I. In the case shown in

the figures, this is ${}^{X}R_1(d_k)$. Those rectangles found visible through Conditions I to IV always occur on this exterior edge trace. Thus, we must always start with an appropriate corner of one of them.

A trace consists of a series of points with directed lines that connect them. To designate the trace direction, let Y^- and Y^+ denote the decreasing and increasing Y directions, respectively. Similarly, we let Z^- and Z^+ denote the decreasing and increasing Z directions, respectively. We will choose the starting point of the trace as follows. We take all the rectangles that meet Condition I and we let ${}^{X}R_q(d_k)$ represent each of these rectangles. Of these rectangles, we find the one rectangle ${}^{X}R_j(d_k)$ that satisfies $Z_j^{\min} = \min\left(Z_q^{\min}\right)$, meaning it will have the minimum $Z_j^{\min}$ of all the rectangles ${}^{X}R_q(d_k)$. We then use $\left(Y_j^{\min}, Z_j^{\min}\right)$ of the rectangle ${}^{X}R_j(d_k)$ as the starting point. This point is chosen because it is guaranteed to occur on an exterior edge trace, no matter how complex the configuration of rectangles is. The trace then proceeds in the Y^+ direction and continues around the rectangles. This trace stops when the starting point is reached. For example, in Figure 5.7 only ${}^{X}R_1(d_k)$ satisfies Condition I. Thus, we choose its bottom left corner as the starting point. If other rectangles also satisfied Condition I, we would find among these rectangles the one whose bottom edge is furthest beneath the others and use its lower left hand corner as the starting point. The trace then proceeds to the right and ends when the starting point is reached.

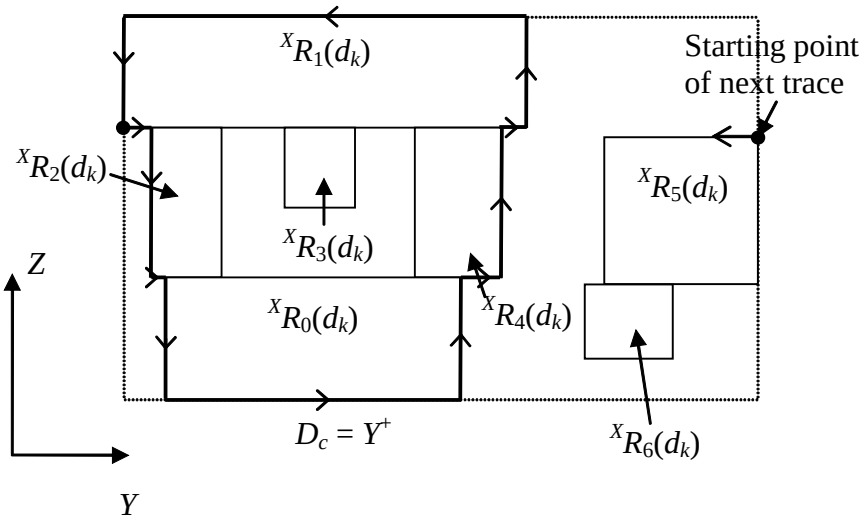


Figure 5.8. A profile in which rectangle 5 does not appear on the first edge trace. (D_c=Y^+ indicates the current direction)

We will discuss the details of the trace subsequently. For now, let us assume that the trace has been performed. From the procedure just described, we could have missed some rectangles in the profile that are visible. An example of this is shown in Figure 5.8. Regions $^{X}R_5(d_k)$ and $^{X}R_6(d_k)$ should be marked visible, but in the current scheme, they are not. To remedy this, we perform additional traces as needed if there is a rectangle that meets Conditions I to IV that has not been marked visible in the profile. For example, in Figure 5.8, $^{X}R_0(d_k)$, $^{X}R_1(d_k)$, and $^{X}R_5(d_k)$ satisfy one or more of the conditions. However, the initial trace does not include $^{X}R_5(d_k)$, which meets Condition III. Thus, another trace must be performed starting on $^{X}R_5$, whose starting point is based on one of the following conditions:

Condition I-A: $\left(X_j^{\min}, Y_j^{\min}\right)$, where $^{X}R_j(d_k)$ is the rectangle where $Z_j^{\min} = \min\left(Z_q^{\min}\right)$. The rectangles $^{X}R_q(d_k)$ are those that do not appear in the trace that meet Condition I. The trace proceeds in the Y^+ direction.

Condition II-A: $\left(Y_j^{\max}, Z_j^{\min}\right)$, where ${}^{X}R_j(d_k)$ is the rectangle where $Y_j^{\max} = \max\left(Y_q^{\max}\right)$. The rectangles ${}^{X}R_q(d_k)$ are those that do not appear in the trace that meet Condition II. The trace proceeds in the Z^+ direction.

Condition III-A: $\left(Y_j^{\max}, Z_j^{\max}\right)$, where ${}^{X}R_j(d_k)$ is the rectangle where $Z_j^{\max} = \max\left(Z_q^{\max}\right)$. The rectangles ${}^{X}R_q(d_k)$ are those that do not appear in the trace that meet Condition III. The trace proceeds in the Y^- direction.

Condition IV-A: $\left(Y_j^{\min}, Z_j^{\max}\right)$, where ${}^{X}R_j(d_k)$ is the rectangle where $Y_j^{\min} = \min\left(Y_q^{\min}\right)$. The rectangles ${}^{X}R_q(d_k)$ are the remaining rectangles that do not appear in the trace that meet Condition IV. The trace proceeds in the Z^- direction.

For example, in Figure 5.8, ${}^{X}R_5(d_k)$ does not appear on the first trace, but meets Condition III. The next trace would start with the top right corner of ${}^{X}R_5(d_k)$ and proceed left. As long as there is a rectangle that meets one of Conditions I through IV that is not part of a trace, we continue performing traces using these rectangles as starting points.

We now discuss the details of how to perform the trace. Let ${}^{X}R_c(d_k)$ denote the current rectangle, D_c denote the current direction, and (Y_p, Z_p) denote the current point in the trace. Since this trace occurs in a counter-clockwise manner, the value of either Y_p or Z_p corresponds with the current rectangle and the direction traveled. For example, a trace with $D_c = Y^+$ can only occur on the minimum Z edge of ${}^{X}R_c(d_k)$. An example of

this is shown in Figure 5.8. Therefore, Z_p is the minimum coordinate of ${}^{X}R_c(d_k)$. One of the following three events can occur as the algorithm traverses in the current direction:

(1) A rectangle will be encountered that will change the direction of the trace,

(2) The end of the rectangle is reached, but the trace can be continued in the same direction on a different rectangle, or

(3) The end of the rectangle is reached and the trace is continued on the next edge of the same rectangle.

We use these events to determine the direction the trace should go. Examples of these three events are shown in Figure 5.9.

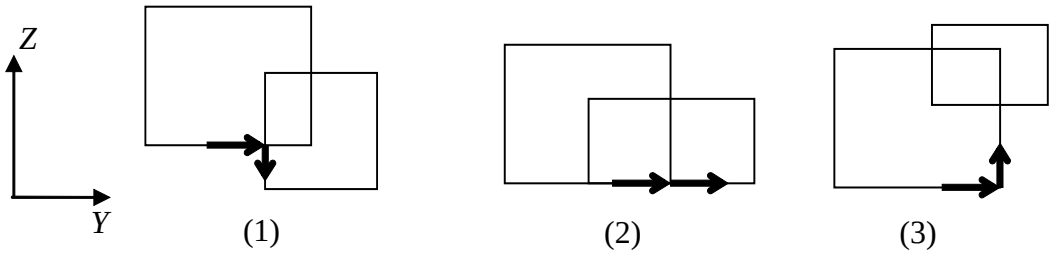
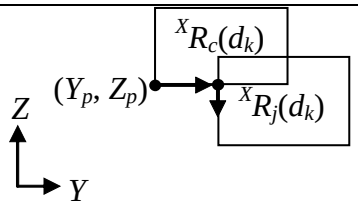
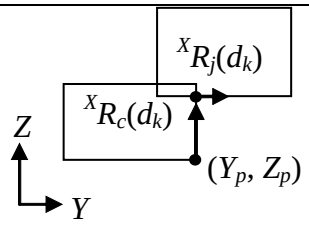
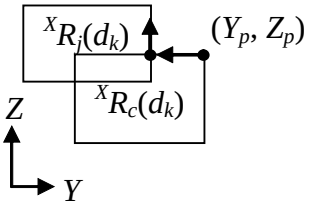
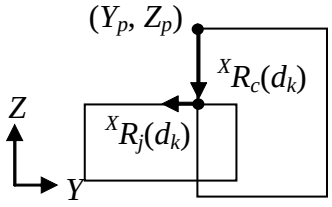


Figure 5.9. Examples of the occurrence of the three rectangle traversal events. (1) direction changed by rectangle encountered. (2) trace continued in same direction on different rectangle, (3) trace continued on next edge of same rectangle.

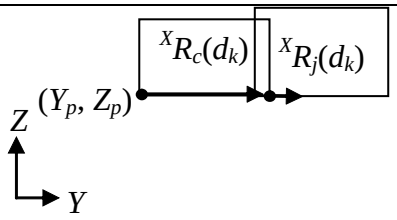
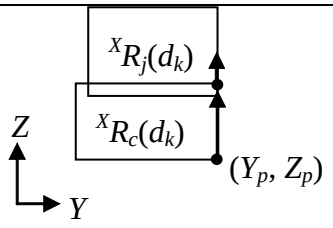
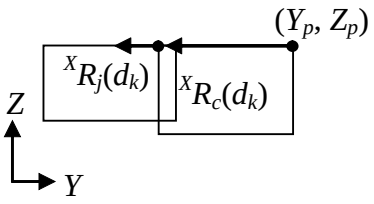
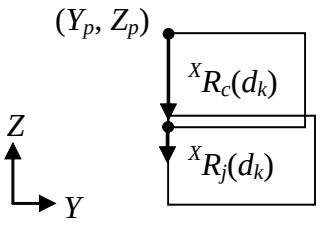
We determine the next step of the trace by first searching for a ${}^{X}R_j(d_k)$ that will change the trace direction. The conditions used for finding this rectangle and the actions taken if found are shown in Table 5.1. The lists L_Y and L_Z are used to find the boxes that satisfy the above conditions, so that not all rectangles are checked. For example, if we are searching in the Y^+ direction, we start with the current point in L_Y and advance through the list to find the first rectangle that meets the conditions. If no such rectangle is found when the last bound of the current rectangle is reached in the list, the search is ended.

Table 5.1. Finding a rectangle that changes the trace direction			
D_c	Conditions for the rectangle ${}^{X}R_j(d_k)$	Figure for finding of ${}^{X}R_j(d_k)$	Actions taken if found
Y^+	Of the rectangles that satisfy $Y_p < Y_j^{\min} \le Y_c^{\max}$ and $Z_j^{\min} < Z_p \le Z_j^{\max}$, ${}^{X}R_j(d_k)$ has the minimum $Y_j^{\min}$		$Y_p = Y_j^{\min}$, ${}^{X}R_c(d_k) = {}^{X}R_j(d_k)$, and $D_c = Z^-$
Z^+	Of the rectangles that satisfy $Z_p < Z_j^{\min} \le Z_c^{\max}$ and $Y_j^{\min} \le Y_p < Y_j^{\max}$, ${}^{X}R_j(d_k)$ has the minimum $Z_j^{\min}$		$Z_p = Z_j^{\min}$, ${}^{X}R_c(d_k) = {}^{X}R_j(d_k)$, and $D_c = Y^+$
Y^-	Of the rectangles that satisfy $Y_c^{\min} \le Y_j^{\max} < Y_p$ and $Z_j^{\min} \le Z_p < Z_j^{\max}$, ${}^{X}R_j(d_k)$ has the maximum $Y_j^{\max}$		$Y_p = Y_j^{\max}$, ${}^{X}R_c(d_k) = {}^{X}R_j(d_k)$, and $D_c = Z^+$
Z^-	Of the rectangles that satisfy $Z_c^{\min} \le Z_j^{\max} < Z_p$ and $Y_j^{\min} < Y_p \le Y_j^{\max}$, ${}^{X}R_j(d_k)$ has the maximum $Z_j^{\max}$		$Z_p = Z_j^{\max}$, ${}^{X}R_c(d_k) = {}^{X}R_j(d_k)$, and $D_c = Y^-$

If a rectangle is not found that will change the direction of the trace, we search for a rectangle ${}^{X}R_j(d_k)$ that continues the trace in the same direction. The conditions for finding this rectangle and the actions taken if found are shown in Table 5.2. Again, the lists L_Y and L_Z are used to search for this rectangle. For example, if we are searching in the Y^+ direction, we start with the current point in L_Y and advance through the list to the first rectangle to meet the conditions stated in Table 5.2 for the Y^+ direction. If no such

rectangle is found when the first $Y_j^{\min} > Y_c^{\max}$ is encountered in the current list, then the search is ended.

Table 5.2. Finding a rectangle that continues the trace direction

D_c	Conditions for the rectangle ${}^{X}R_j(d_k)$	Figure for finding of ${}^{X}R_j(d_k)$	Actions taken
Y^+	$Y_j^{\min} \le Y_c^{\max} < Y_j^{\max}$ and $Z_j^{\min} = Z_p$		$Y_p = Y_c^{\max}$, then ${}^{X}R_c(d_k) = {}^{X}R_j(d_k)$
Z^+	$Z_j^{\min} \le Z_c^{\max} < Z_j^{\max}$ and $Y_j^{\max} = Y_p$		$Z_p = Z_c^{\max}$, then ${}^{X}R_c(d_k) = {}^{X}R_j(d_k)$
Y^-	$Y_j^{\min} < Y_c^{\min} \le Y_j^{\max}$ and $Z_j^{\max} = Z_p$		$Y_p = Y_c^{\min}$, then ${}^{X}R_c(d_k) = {}^{X}R_j(d_k)$
Z^-	$Z_j^{\min} < Z_c^{\min} \le Z_j^{\max}$ and $Y_j^{\min} = Y_p$		$Z_p = Z_c^{\min}$, then ${}^{X}R_c(d_k) = {}^{X}R_j(d_k)$

Finally, if this rectangle is not found, the trace continues on the same rectangle on its next edge. The direction and current position are changed to reflect this. The actions taken are shown inTable 5.3.

Table 5.3. Continuing the trace on the current rectangle, but in a different direction.	
D_c	Actions taken
Y^+	$Y_p = Y_c^{\max}$ and $D_c = Z^+$
Z^+	$Z_p = Z_c^{\max}$ and $D_c = Y^-$
Y^-	$Y_p = Y_c^{\min}$ and $D_c = Z^-$
Z^-	$Z_p = Z_c^{\min}$ and $D_c = Y^+$

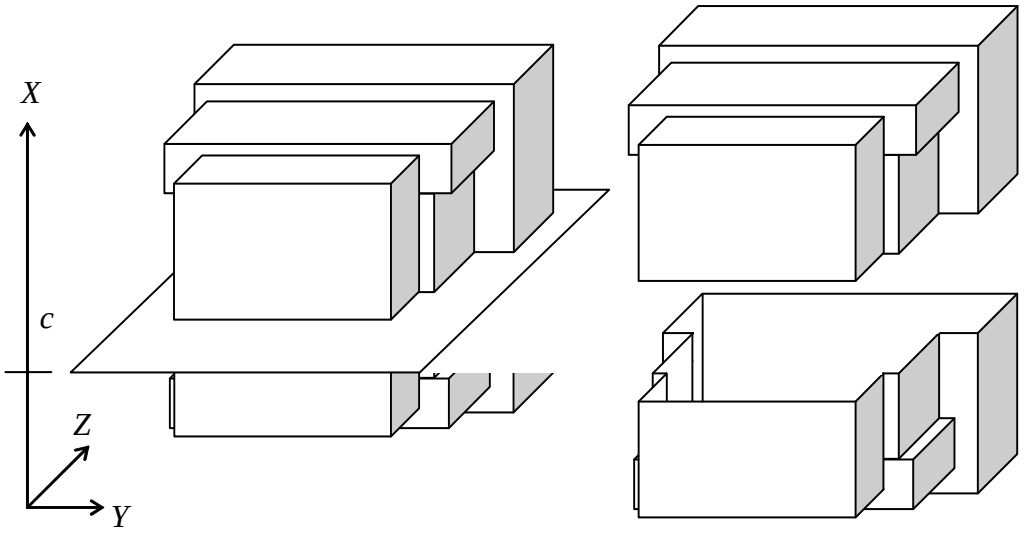
As the trace is performed, we mark ${}^{Z}R_c(d_k)$ visible for each section of the trace. Once the trace is completed, if needed, other traces are performed as described previously if there are rectangles satisfying any of the Conditions I through IV that are not visible. After all traces are performed on the *k*th profile, for each rectangle that was marked visible in this profile, the corresponding AABB is marked visible. Then, a trace is performed on the profile for d_{k+1}.

After the traces have been performed on all the profiles in the *X*-direction, this procedure is performed, if necessary, in the *Y*-direction, and then the *Z*-direction. The analysis in the *Y*-direction is necessary only if there are AABBs that are still invisible after all of the profiles in the *X*-direction have been analyzed. Then an analysis for the *Z*-direction is performed only if there are still invisible AABBs after the analysis in the *Y*-direction. In the *Y*-direction, an analysis is performed in the same manner as the *X*-direction, only using *X* and *Z* values for the profiles instead of *Y* and *Z*. Similarly, for

the *Z*-direction, profiles in *X* and *Y* are used. Those non-contained AABBs that were not marked visible after an analysis in the three directions are marked invisible.

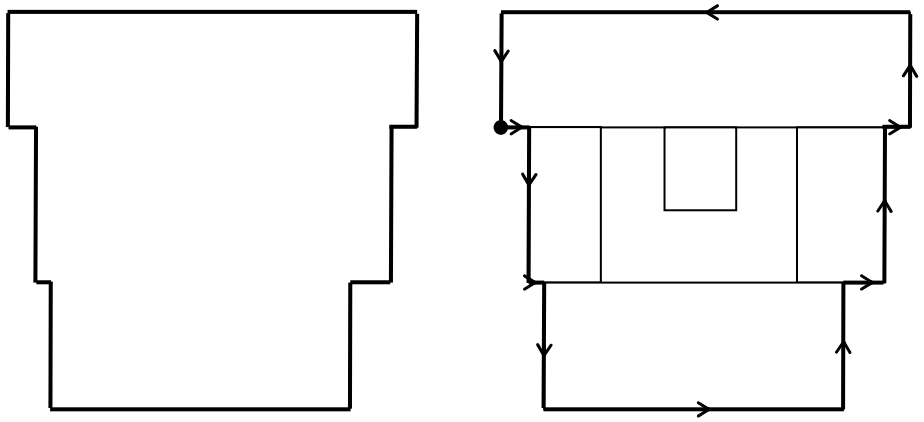
The reason we use this approach is because it guarantees that AABBs that are wholly enclosed within a group of AABBs will be detected as invisible. To prove this, consider the following. For the enclosed AABBs to be invisible, any ray originating from outside the AABB configuration that intersects one of the enclosed AABBs will intersect another AABB first. Because the enclosed AABBs will never be the first AABB intersected by such a ray, they will never be visible. Now, because the AABBs are enclosed, one can also create a shell from the faces of the non-enclosed AABBs whose interior volume includes the volume of all the AABBs plus any empty volume that occurs in the enclosure. For example, the shell for the AABBs in Figure 5.3a is shown in Figure 5.10. Because the points where the rays first intersect the AABB configuration are the points where they first enter any volume from the AABBs, these points are all located on this shell. As such, the shell incorporates all the exterior points of the AABB configuration. Now, let us take any cross-section that goes through an enclosed AABB, including a cross-section of the shell. In examining, the cross-section of the shell, it must contain all the exterior points of the cross-section, because the shell contained all the exterior points of the AABB configuration. However, in our method, the exterior edge trace also consists of all the exterior points of a cross-section. Thus, they are essentially the same. For example, a cross-section of the shell in Figure 5.10 is shown in Figure 5.11a. It can be seen to have all the same points as the exterior edge trace shown in Figure 5.11b. Because the enclosed AABB was not a part of the shell, its rectangle does not have any exterior edge points, and thus will not appear in an

exterior edge trace. As such, it will not be marked visible for this cross-section. However, since this is true of any cross-section through an enclosed AABB, it is not possible for the enclosed AABB to be marked visible in our algorithm and therefore, will be correctly marked as invisible.



(a) (b)

Figure 5.10. Shell for AABBs in Figure 5.3a. (a) Shell and cross-section plane. (b) Exploded shell where cross-section is taken.



(a) (b)

Figure 5.11. Cross-section profiles of (a) shell and (b) AABBs with exterior edge trace.

5.4. Visibility of contained AABBs

After all of the cross-sections have been analyzed, the visibility of the AABBs that were not analyzed previously is determined. These are the AABBs that are contained in other AABBs. This visibility determination is made using the fact that contained AABBs do not necessarily belong to objects that are invisible.

In the process of surrounding each object with an AABB, there is empty space where another object could be located. Thus, this other object's AABB would be contained in the AABB of the first object, but the object itself could be visible. For example, consider the objects shown in Figure 5.12. A portion of the screw is contained in the empty volume of the AABBs for the wedge shaped object and the plate. However, the screw is visible. This case illustrates that by assuming that a part would be invisible if its AABB is contained in another AABB would yield an incorrect result.

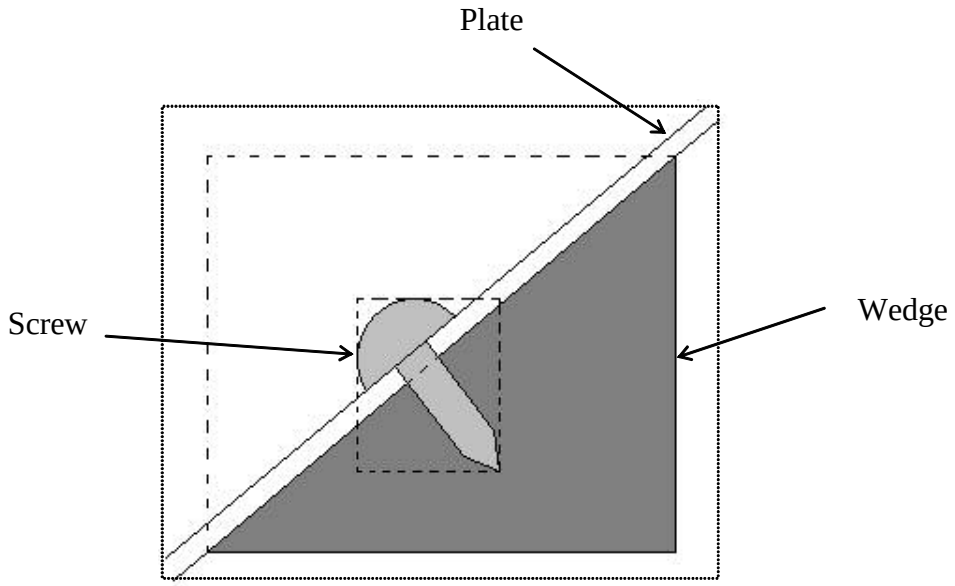


Figure 5.12. Visible part whose AABB is contained in another AABB.

In order to deal with this contingency, the following rule is applied:

Rule #1: If an AABB is contained in another AABB that is visible, then the contained AABB is visible.

With this rule, a conservative visibility approach is used. This means that the set of objects that are visible to the human eye will be a subset of the objects determined visible through the entire visibility determination process of the computer program. The fact remains that a few parts whose AABBs are contained in visible AABBs are invisible to the human eye. Thus, this rule makes these parts visible when they should not be. However, this small sacrifice is made so that those AABBs that are actually visible can be detected rapidly. A more complex rule could be employed to make these determinations correctly, but that would require a more intensive computation. In addition, the only difference would be the visibility of a small number of parts, meaning only a small increase in performance by marking these parts invisible. It would be inefficient to have to use a very large calculation time to determine the visibility of these parts, when it will yield the same results in display with only a small increase in display performance. Thus, this rule offers a fast, albeit conservative way to determine the visibility of parts whose AABBs are contained in another AABB. For example, the screw in Figure 5.12 is detected as visible, as its AABB is contained in the AABB of the plate, which is visible from a previous determination. Once the visibility determination of the contained AABBs is finished, we have completed the visibility determination of all the parts. The basic steps of the algorithm are summarized in Table 5.4.

Table 5.4. Basic steps of the visibility determination algorithm

1. Determine AABB for each surface.
2. Determine AABB for each part based on the AABBs of its surfaces.
3. Sort the coordinates of the part AABBs in the *X*, *Y*, and *Z* directions.
4. Determine which part AABBs are contained in another single part AABB.
5. Perform cross-section traces to determine visibility of non-contained AABBs.
6. Determine visibility of contained AABBs based on the visibility of the AABBs that contain them.

5.5. Validation of the Visibility Algorithm

In order to test the visibility algorithm, a test assembly was created. Several parts were created with the general shape shown in Figure 5.13. A group of four congruent parts and a rectangular plate are used to form a box assembly as shown in Figure 5.14. This box may or may not be closed by adding another plate to the front, depending on what configuration is to be tested. These boxes are then created in different sizes and nested within each other, as shown in Figure 5.15. We will use this to both test the correctness and speed of the algorithms based on different numbers of boxes and different configurations of open and closed boxes.

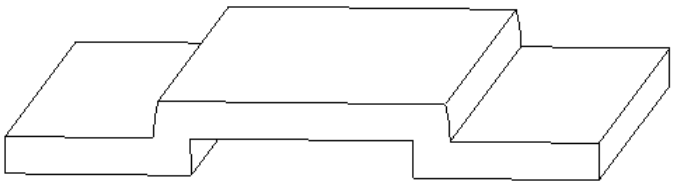


Figure 5.13. General shape of parts used in test assembly.

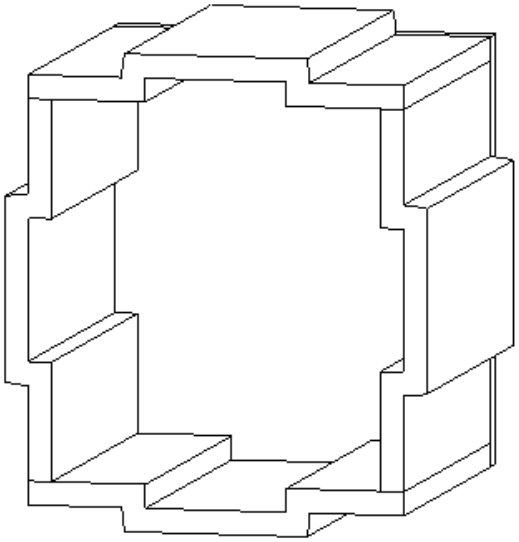


Figure 5.14. Five-sided box assembly created from part shown in Figure 5.13.

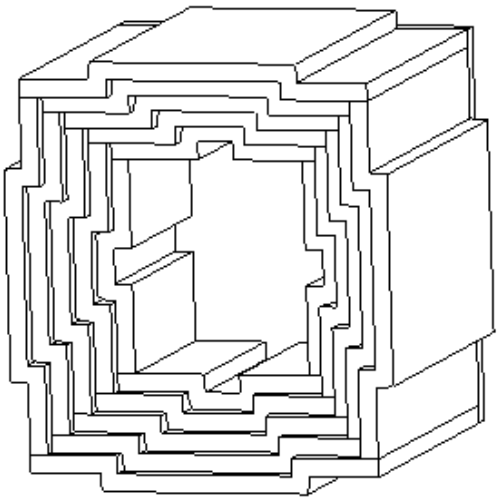


Figure 5.15. A group of nested boxes.

In performing the algorithms on any set of open boxes, we find that all of the parts are determined as visible, as expected. Also, we find that when the outermost box is closed, only the parts that comprise the outer box are visible. All the rest of the parts are marked invisible, no matter whether they are closed or open. Also, when mixing configurations of open and closed boxes, we find that any parts from boxes that are interior to a closed box are all marked as invisible. Similarly, all the parts that are exterior to the outermost closed box are marked visible. In addition, the part that closes the outermost box is also visible. Based on the results of these runs, it is concluded that the algorithms are working correctly.

The purpose of these runs is also to establish the speed of the algorithms. When doing these runs on a machine with a Pentium 4 2.4 GHz processor, it was found that almost the entire CPU time that the computer used to establish the visibility of the parts was employed in the determination of the AABBs. In fact, using the maximum of 30 parts for five nested closed boxes, the portion of the run time used to determine visibility after AABB determination was less than 0.008 seconds. As such, the determining factor for run time of the algorithms is the AABB determination. The resulting processor time versus the number of parts is shown in Figure 5.16. The times measured are the average value of ten runs each, with standard deviation shown with error bars. The relationship between processor time and number of parts is fairly linear, mainly due to the similar shape of all the parts and the fact that the determination of the AABBs for the parts is independent from each other. Therefore, the run time for each part is approximately 0.005 seconds per part. In actuality, the AABB determination time for each part is heavily dependent on the types of surfaces that make up the part.

The AABB for a part made entirely of plane surfaces will be detected quickly, whereas a part composed of spline surfaces or revolved spline curves will require more time, as they require a lot more processing in order to find the limits of their AABBs. However, for relatively simple parts, we know that the visibility determination on a 2.4 GHz processor is approximately 0.005 seconds per part. The processing time for the determination of the AABB for the part in Section 3.3, which is a more complex part, is approximately 0.016 seconds, and is probably a more typical processing time.

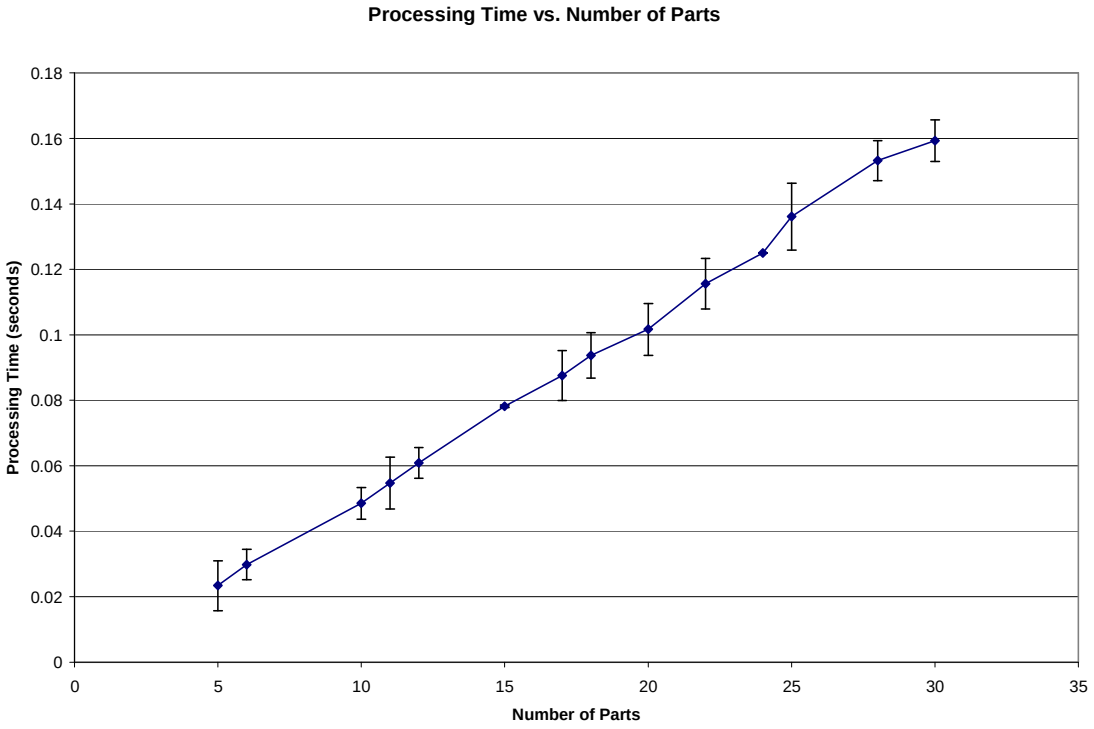


Figure 5.16. Plot of processor time versus number of parts for nested box assemblies.

5.6. Limitations of the Current Algorithm

Certain combinations of geometries will be incorrectly marked invisible when they may be visible. This is caused by the empty volume that results from the use of bounding boxes, which is the volume that is contained in the bounding box that is not

occupied by the part. For example, Figure 5.17 shows a simple assembly in which Part 2, which rests on the shelf created by Parts 1 and 3, is detected incorrectly. We now discuss the reasons for this. Drawing the AABBs around the parts, we get the results shown in Figure 5.18. Since Part 2 is a block, its AABB is identical to the block itself. It is seen that $(AABB)_2$ is not contained in either $(AABB)_1$ or $(AABB)_3$. Thus, Rule #1 in Section 5.4 does not apply. Taking cross sections in all three directions, the resulting profiles that are obtained when the intersecting plane passes through $(AABB)_2$ are shown in Figure 5.19. From these profiles, it is seen that any of the rectangles pertaining to $(AABB)_2$ will not appear in an exterior edge trace in any of the cross sections. Thus, the visibility algorithm, as it currently exists, will not detect that Part 2 is visible.

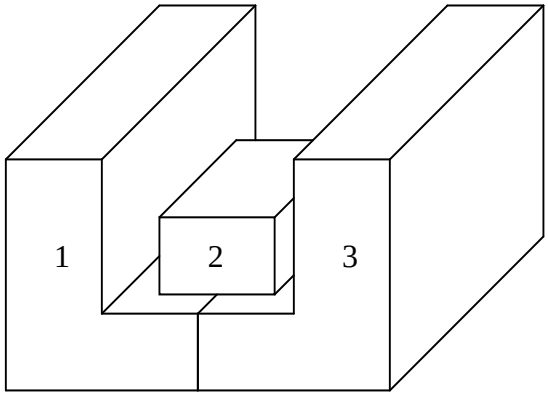


Figure 5.17. Example of parts that will yield incorrect results using the current algorithms. Part 2 will be marked invisible.

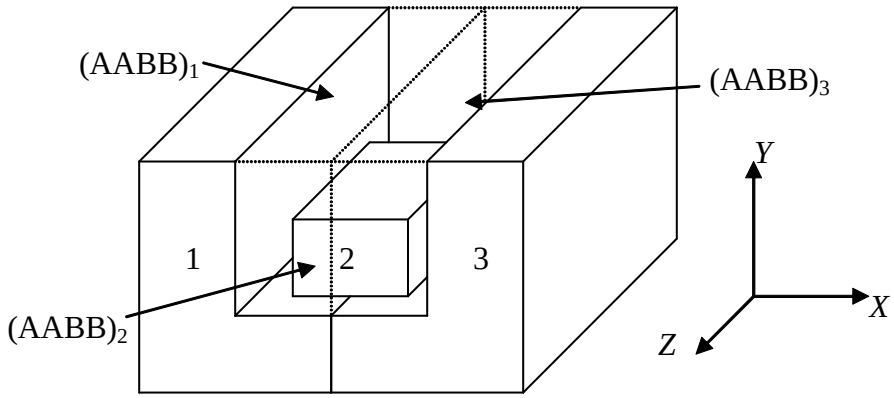


Figure 5.18. Parts in Figure 5.17 and their AABBs.

Upon examining this configuration, we see that Part 2 is determined to be invisible because $(AABB)_2$ is contained in the volume created by the union of $(AABB)_1$ and $(AABB)_3$. In fact, any combination of AABBs whose union contains another AABB will cause that AABB to be marked invisible, whether or not it should be. The rectangular cross sections of the AABBs creating the union will always have one or more of its edges marked as an exterior edge during the edge-tracing portion of the algorithm. This will not always be true for those AABBs appearing in the volume created by the union.

The reason why AABBs of parts contained in the volume created by the union of other AABBs are visible is that when parts are assembled, the empty volumes from several AABBs can intersect. The nature of some of these intersecting volumes is such that anything that is contained in that volume may be visible. Thus, parts whose AABBs lie within this volume are not contained within a single AABB, where Rule #1 in Section 5.4 would apply. This will cause these parts contained in the intersecting volumes to be marked invisible. This incorrect invisibility assignment is a function of the geometry of the parts and how they are assembled. Thus, for more complex

assemblies there is an increased possibility of parts being marked invisible when they should be marked visible.

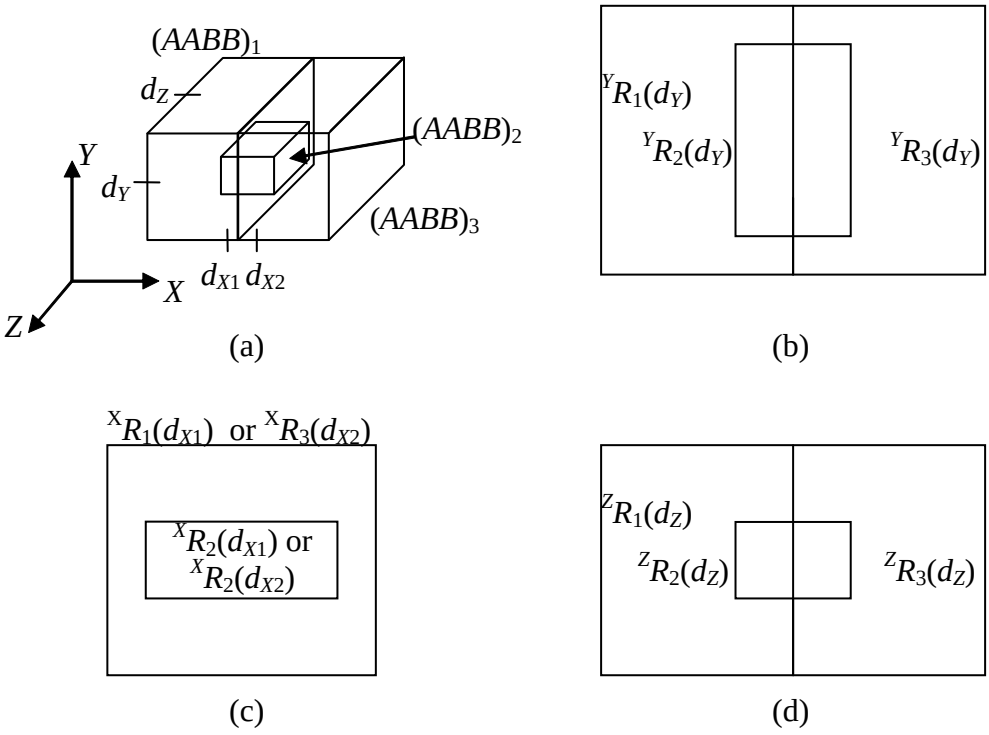


Figure 5.19. Cross-section profiles of the AABBs in Figure 5.18. (a) AABBs in 3-D space and the locations of the cross-sections taken, (b) cross-section at d_Y, (c) cross-section at d_{X1} or d_{X2}, (d) cross-section at d_Z.

5.7. Removal of a Limitation of the Algorithm

In this section, we propose a scheme that can decrease the amount of empty volume, without greatly lengthening the amount of computational time needed to determine the visibility of parts. The basic scheme is to allow more than one AABB per part, where each AABB only contains a section of the part. We only require that the part will be contained in the union of these AABBs. This will decrease the amount of empty space. The basic steps that would be taken with this modification are shown in Table 5.5.

Table 5.5. Basic steps of the visibility determination algorithm when allowing multiple AABBs per part.

1. Determine the multiple AABBs for each part based on some kind of geometric determinations (e. g. surface AABBs)
2. Sort the coordinates of the part AABBs in the *X*, *Y*, and *Z* directions.
3. Determine which part AABBs are contained in another single AABB.
4. Perform cross-section traces to determine visibility of non-contained AABBs.
5. Determine visibility of contained AABBs based on the visibility of the AABBs that contain them.

Consider the part shown in Figure 5.20. The solid-gray figure represents an arrow-shaped part. Figure 5.20a shows the part with only one AABB. Figure 5.20b shows the part with two AABBs. The shaded sections are the portions of empty volume included in (a) that are eliminated by (b). By doing this, we decrease the amount of empty volume in the AABBs, and thus, increase the visibility/invisibility detection capability of the algorithm.

How the multiple AABBs are determined is not a part of this research. Research on this subject can be found, for example, in the work of Sanna and Montuschi (1995). Their research includes the determination of a predetermined number of bounding boxes for an object given a larger number of bounding boxes. This technique lends itself well to the bounding box technique described in Section 3, since bounding boxes are determined for each surface in a part. This collection of AABBs can then be combined into a predetermined number of AABBs using their technique.

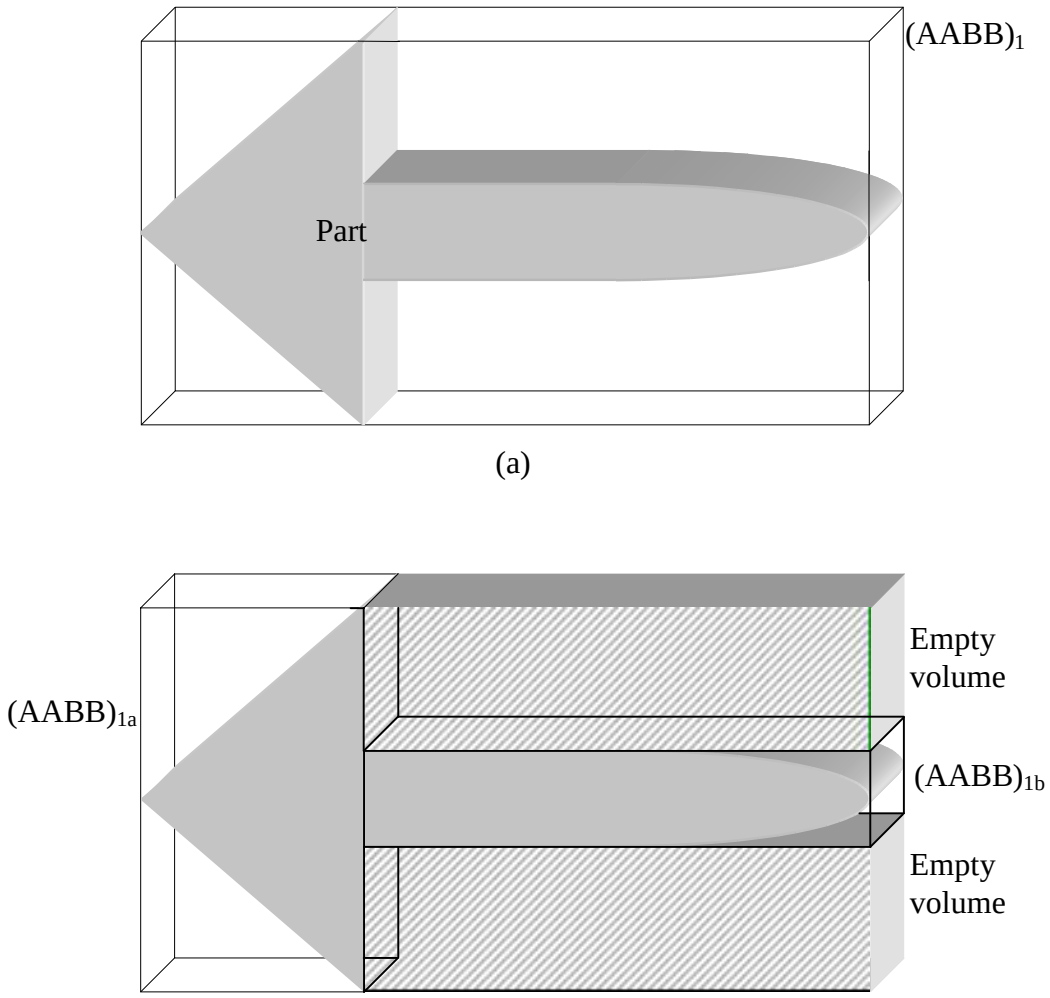


Figure 5.20. Part with multiple AABBs. (a) one AABB (b) two AABBs

Once the AABBs for each part are determined, the algorithms previously described can be used without making any major modifications to them. The only change that needs to be made is to allow more than one AABB to correspond to the same part. No modifications need to be made because the algorithms determine only which parts are visible, assuming that those not determined visible are invisible. With this scheme, when an AABB is encountered in any of the cross-section traces, the

corresponding part is automatically determined as visible. There is no algorithm that makes the part invisible afterwards.

To illustrate how this technique will work, once again consider the parts in Figure 5.17. Allowing more than one AABB per part, one possible configuration of AABBs would be to have two AABBs each for parts 1 and 3, as shown in Figure 5.21. The two AABBs for Part 1 are labeled 1-1 and 1-2. Similarly, the AABBs for part 3 are labeled 3-1 and 3-2. The cross sections through $(AABB)_2$ in all three directions are shown in Figure 5.22. With these cross sections, the exterior edge traces will detect part 2 as being visible.

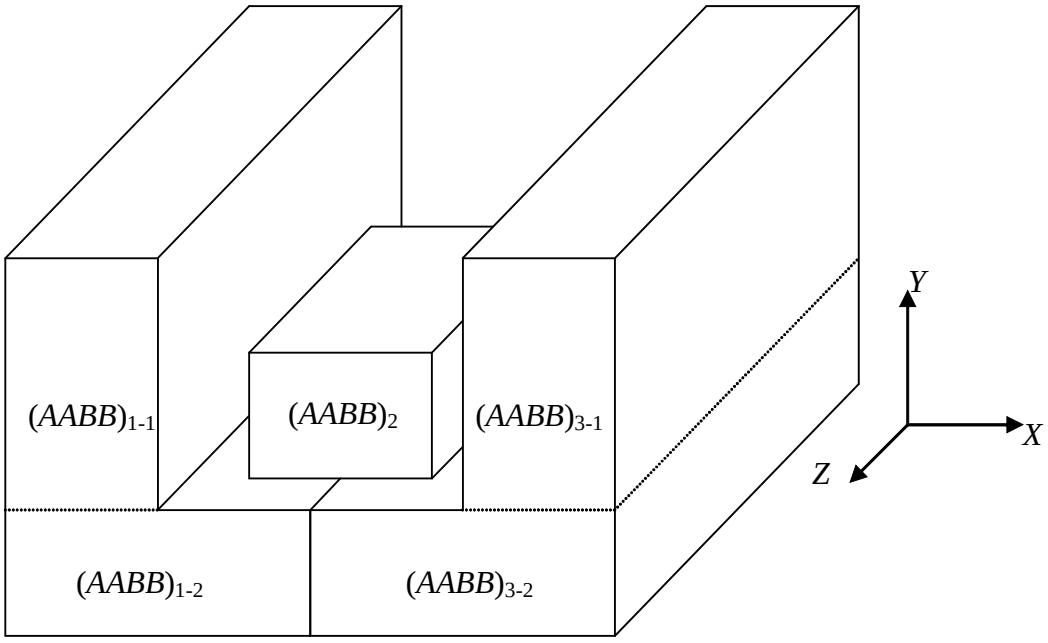


Figure 5.21. Parts in Figure 5.17 with multiple AABBs allowed.

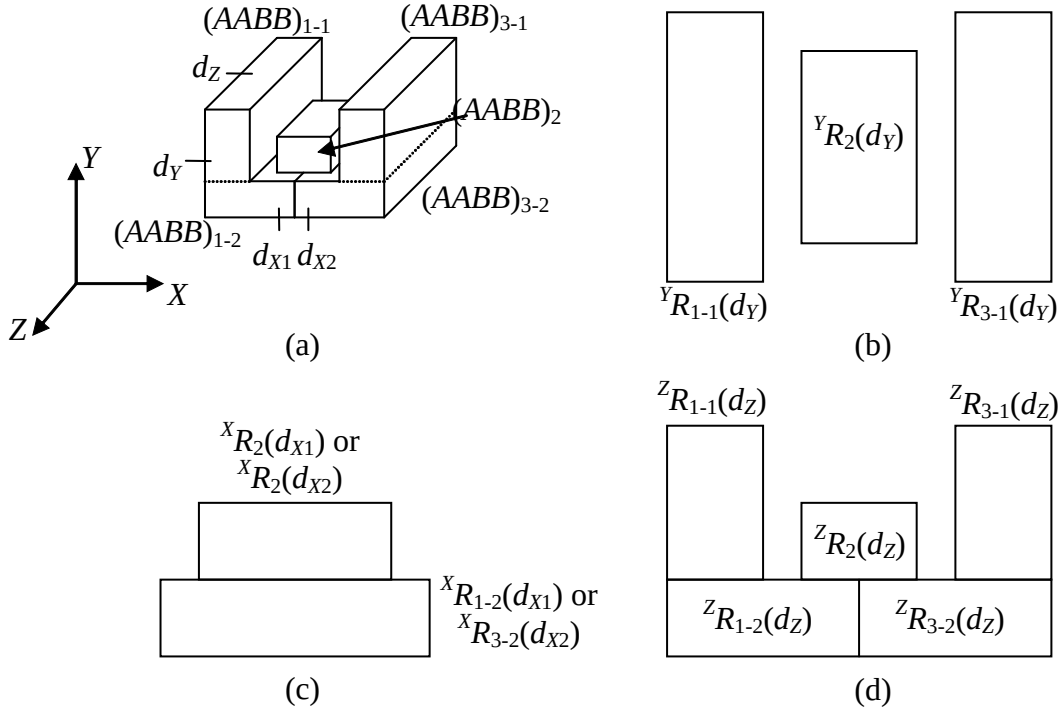


Figure 5.22. Cross-section profiles of the AABBs in Figure 5.21. (a) AABBs in 3-D space and the locations of the cross-sections taken, (b) cross-section at d_Y, (c) cross-section at d_{X1} or d_{X2}, (d) cross-section at d_Z.

5.8. Corollaries to the algorithms

5.8.1. Using the cross-section trace algorithms on surface AABBs

To increase display speed, one can reduce the number of surfaces that are rendered. One way to accomplish this is to use the cross-section visibility trace algorithms on the AABBs of each individual part surface to calculate surface visibility before using them on the parts themselves. An assembly is a collection of parts placed together in a certain configuration. A part is a collection of surfaces placed together in a certain configuration. As such, a part can be seen as an "assembly" of surfaces. To find the AABB for each part we needed the AABB for each surface. Using the algorithms on these surface AABBs for each part, the visibility of the surfaces for the

part can be determined. This process results in the collection of surfaces that are to be rendered if this part is found visible and eliminates from rendering any surfaces that are not visible in a part. Surfaces not visible in a part will not be visible in an assembly that includes the part. The process is summarized in Table 5.6. However, this process will likely not produce much savings as far as reducing the number of surfaces to be rendered, as parts are not typically modeled with surfaces that will not be visible. An example is shown in Figure 5.23, which is a sphere. Spheres are output as two surfaces. As such, there are two AABBs that each correspond with a surface. Obviously, the two surfaces are visible for this part. Performing the cross-section trace algorithm on the two AABBs will also result in both surfaces as visible. But suppose that performing the trace algorithm causes the top surface to be marked invisible. The part's AABB would still be the same. This AABB is then used as normal in determining the visibility of the part as normal. However, if the part is determined as visible, we would then only display the bottom half of the sphere, as the top half was determined invisible earlier.

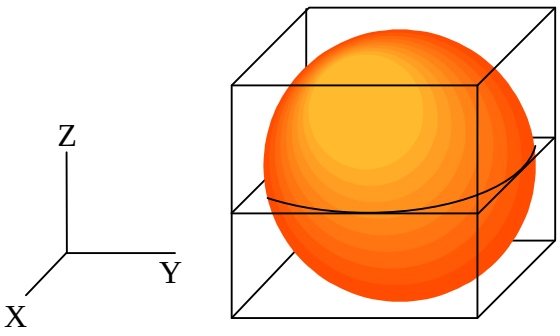


Figure 5.23. A sphere with its two surface AABBs.

Another possibility is to use surface AABBs when analyzing the assembly instead of part AABBs. The process is given in Table 5.7. This requires a great deal more processing, but it will reduce the amount of geometry that needs to be rendered

even more than simply applying the algorithms on the parts of an assembly. With the current algorithm, when a part is marked visible, all of its surfaces are rendered. However, this does not mean that all of the surfaces rendered are visible. Many of these surfaces may not be visible because of the geometry of the assembly. Those assemblies that result in some enclosed parts have the most to gain. Those parts that form the exterior will be visible. However, many of the surfaces of these parts are inside the assembly, and not visible. These surfaces could be very complicated, as interior parts could be mounted on them. As such, removing these surfaces could reduce display time dramatically. A simple example is shown in Figure 5.24, which shows an assembly of two parts that form an enclosed box. The "lid" of the box has a square protrusion that goes inside the box when the two parts are assembled. Both parts are in fact visible and should be rendered. However, as an assembly, there are surfaces that make up these parts that are not visible. These surfaces, shown in grey in Figure 5.24a, are interior to the assembled box, and thus are invisible.

Table 5.6. Basic steps using the visibility determination algorithm on the surfaces of each part before performing the algorithm on the parts of the assembly

1. For each part:
    a. Determine the AABB for each surface.
    b. Sort the coordinates of the surface AABBs in the X, Y, and Z directions.
    c. Determine which surface AABBs are contained in another single surface AABB.
    d. Perform cross-section traces to determine visibility of non-contained AABBs.
    e. Determine visibility of contained AABBs based on the visibility of the AABBs that contain them. Those surfaces determined visible will be rendered if its corresponding part is marked visible
2. Determine AABB for each part based on the AABBs of its surfaces.
3. Sort the coordinates of the part AABBs in the X, Y, and Z directions.
4. Determine which part AABBs are contained in another single part AABB.
5. Perform cross-section traces to determine visibility of non-contained AABBs.
6. Determine visibility of contained AABBs based on the visibility of the AABBs that contain them.

Table 5.7. Basic steps of the visibility determination algorithm using surface AABBs instead of part AABBs

1. Determine AABB for each surface.
2. Sort the coordinates of the surface AABBs in the *X*, *Y*, and *Z* directions.
3. Determine which surface AABBs are contained in another single surface AABB.
4. Perform cross-section traces to determine visibility of non-contained AABBs.
5. Determine visibility of contained AABBs based on the visibility of the AABBs that contain them.

In the algorithms' current state, these surfaces would be rendered when they do not need to be rendered. In order to deal with this, performing the algorithm on AABBs of surfaces instead of part AABBs will allow those surfaces that are hidden to be determined as invisible and, thus, not to be rendered. This basically means that we will find the AABB of each surface, but will not be using them to find the AABB of each part. We simply feed the surface AABBs of all the parts in the assemblies into the visibility determination algorithm, and let the visibility of each AABB pertain to the visibility of its corresponding surface, not its part. We are, in effect, viewing assemblies as collections of surfaces instead of parts.

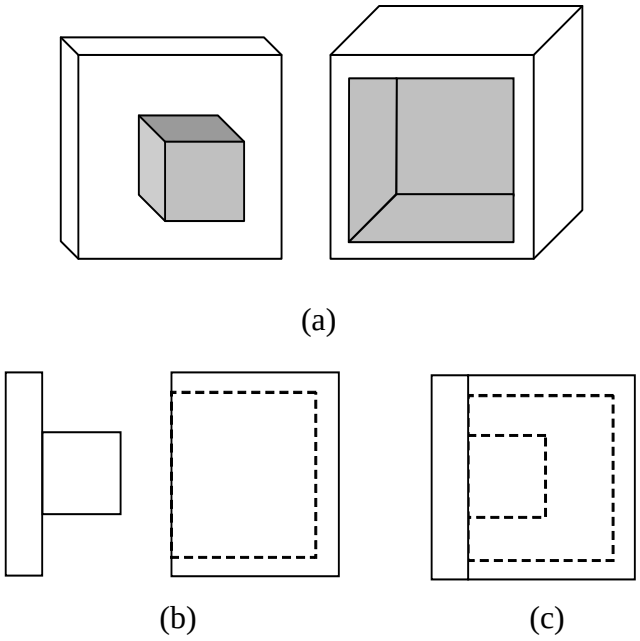
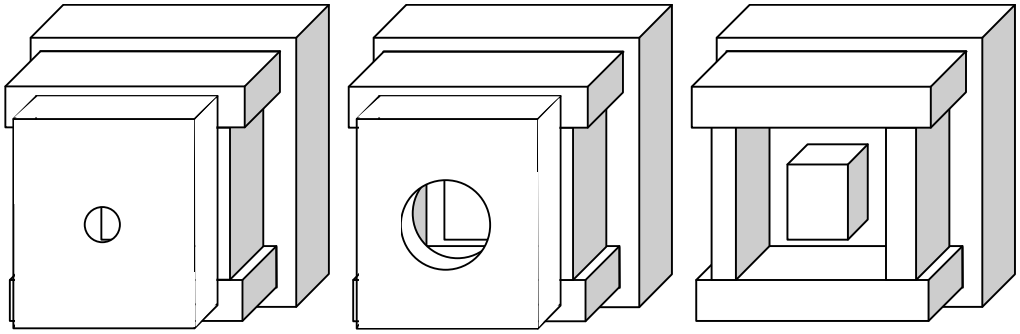


Figure 5.24. Invisible surfaces of two visible parts. (a) view of each individual part. Invisible surfaces in grey. (b) Exploded side view. (c) Side view of parts assembled

5.8.2. Through Holes in Surfaces

Depending on what the models are being used for, holes in surfaces may or may not be important. Models are often used for collision detection and maintainability analyses to determine whether parts can be physically removed and replaced. In these kinds of analyses, visual accuracy is not as important. It does not matter whether parts can be seen through holes. In fact, small holes can be removed entirely for these types of analyses, with the results of these analyses being the same as those results that would be obtained if the analyses were performed with the holes still in place. But, if visual accuracy is important with small holes, the algorithms as they currently stand are not equipped to deal with them. Parts that are only visible through holes will be marked invisible by our algorithms. In addition, if there exist any parts with large holes, such as the assembly in Figure 5.25b, the hole can be a big factor in any type of analysis and

unfortunately cannot be ignored. There are two ways to deal with this, but both are dependent on the development of the algorithms on surface AABBs mentioned in Section 5.8.1.



(a) (b) (c)

Figure 5.25. Assembly where one part has a hole in it. (a) Original assembly (b) Assembly with hole enlarged. (c) Assembly without the part with the hole.

The first proposed method involves treating each surface with a hole in it as if the surface was not there. Essentially all the surfaces that are visible through the holes will still be visible if the holes were larger. Therefore, we make the holes the size of the surface. Thus, those surfaces that should be visible through the holes would be visible if the surface containing the hole were not there. An example is shown in Figure 5.25a, where the front part has a hole through it. Through this hole, the interior part of the configuration is visible. Figure 5.25b shows that enlarging the hole does not change the visibility of any parts. What is visible in Figure 5.25a remains visible in Figure 5.25b. Thus, we enlarge the hole until it encompasses the entire front part, which results in an assembly where the part no longer exists, as shown in Figure 5.25c. Notice that all visible parts are still marked visible despite removal of the front part.

Removing the surfaces with holes before performing the cross-section trace algorithms on the remaining configuration of AABBs is not sufficient, since the

visibility of the surfaces with the holes is unknown. Thus, we modify the cross-section trace algorithm as follows. The traces around the perimeter of the cross-sections are performed on each cross-section profile until the rectangle for an AABB with a hole is encountered in the trace. This surface is marked visible. However, we will need the results of a trace without this rectangle. We could obtain this information by performing the trace again, but the section of the trace that occurs before the rectangle in question occurs will be the same. Instead, we backtrack slightly to the previous rectangle, and then remove the surface with the hole's rectangle from the profile. Then we resume the trace is as if the rectangle were never there. An example is shown in Figure 5.26. The part corresponding to ${}^{X}R_0(d_k)$ has a hole through it. Thus, when the edge trace encounters it in Figure 5.26a, we mark it as visible. Then we return the trace to its previous segment before ${}^{X}R_0(d_k)$ was encountered and remove ${}^{X}R_0(d_k)$ from the profile, as shown in Figure 5.26b. Then we finally continue the trace without ${}^{X}R_0(d_k)$, as in Figure 5.26c. In the end, we have determined visibility for both the surface with the hole and those surfaces that are visible if the surface were not there. Holes in surfaces that are invisible are not considered. The approach is summarized in Table 5.8.

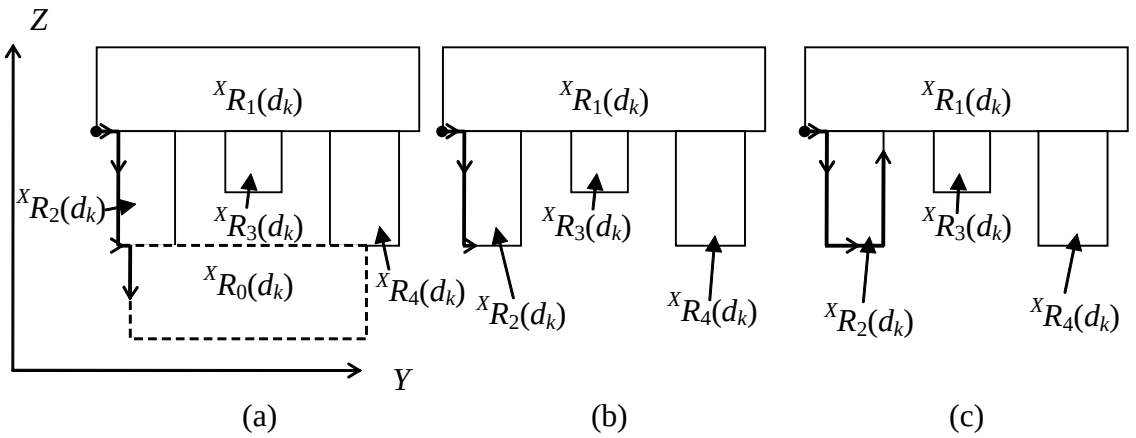


Figure 5.26. Example of edge trace on a profile when the part corresponding to $^{X}R_0(d_k)$ has a through hole. (a) Edge trace encounters rectangle of part with hole. (b) Rectangle is removed and trace backtracked to previous portion. (c) Trace is continued on modified profile.

Table 5.8. Basic steps of the visibility determination algorithm (using surface AABBs instead of part AABBs) accounting for surfaces with holes by treating them as if they are not present.

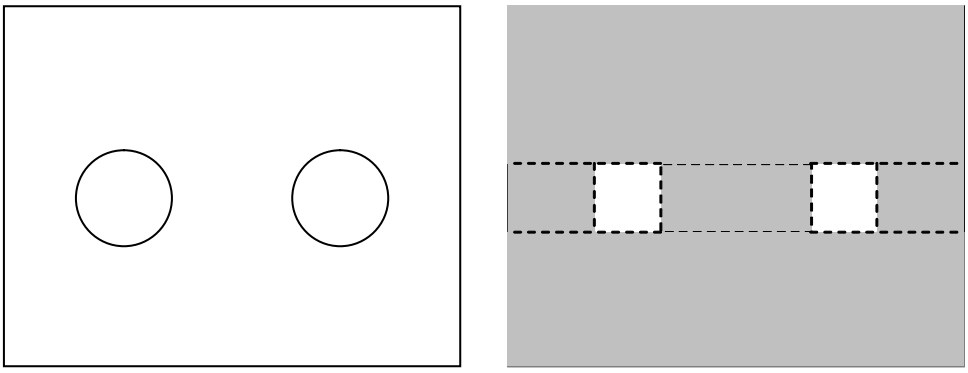
1. Determine AABB for each surface.
2. Sort the coordinates of the surface AABBs in the *X*, *Y*, and *Z* directions.
3. Determine which surface AABBs are contained in another single surface AABB.
4. Perform cross-section traces to determine visibility of non-contained AABBs. If the AABB of a surface that has a hole is encountered, mark the surface visible and then perform the trace without the AABB.
5. Determine visibility of contained AABBs based on the visibility of the AABBs that contain them.

The second proposed method will provide more accuracy than the first one, as far as part visibility through holes is concerned. However, it may require more processing time and may not be worth the small increase in performance that comes from not rendering those parts that are not visible in the second method, but visible in the first method. The second method uses multiple AABBs to surround a surface instead of one AABB, as proposed in Section 5.7 for parts. The difference is that we will be leaving an empty volume where the hole occurs. This process is summarized in Table 5.9. For example, take the surface shown in Figure 5.27 for a 2½ -D example. If we were to use a single AABB, it would be the 2½ -D rectangle that is the border of the surface. However, using multiple AABBs, we are able to cover the surface in a manner that allows open volume to represent the holes, but still have the surface covered by AABBs. With this method, the volume from the holes is left open, allowing the visibility trace to pass through the holes and include those parts that can be seen through the holes. At the same time, the solid sections will still be accounted for by the AABBs. This will allow those parts that are truly invisible despite the holes to still be marked invisible.

Table 5.9. Basic steps of the visibility determination algorithm (using surface AABBs instead of part AABBs) using multiple AABBs for surfaces with holes

1. Determine AABB for each surface. If the surface has a hole, calculate multiple AABBs that can be used to allow empty volume for the hole while still containing the part.
2. Sort the coordinates of the surface AABBs in the *X*, *Y*, and *Z* directions.
3. Determine which surface AABBs are contained in another single surface AABB.
4. Perform cross-section traces to determine visibility of non-contained AABBs.
5. Determine visibility of contained AABBs based on the visibility of the AABBs that contain them.

The second method should be more accurate than the first method because it more closely approximates the geometry of the parts. As a result, this method could cause more invisible parts to be marked visible than the second method. However, the second method also requires more processing time than the previous method in both calculating the AABBs to use for the surface and performing the algorithms on more AABBs. If there are many surfaces with holes, there could be a considerable increase in computation time. In addition, a surface with larger holes will generally result in more visible surfaces behind it than a surface with smaller holes. Thus, the benefit of using the second option over the first is diminished on surfaces with larger holes, as the results of the second option will be much closer to those of the first option while requiring more processing time.



(a) (b)

Figure 5.27. Example of (a) a surface with holes and (b) how it would be covered by multiple AABBs.

In dealing with the problem of surface through holes, these two options are good ways to deal with the problem of visibility through holes. If interior surfaces are simple and easy to render, the first option would be a better choice, as those surfaces that would be determined visible by the first option but invisible by the second one do not require much time to render. The additional processing time used to determine their visibility would not be worth the savings in rendering. In addition, surfaces with large holes would have a lot of interior surfaces determined visible by the second option anyway, so it is a good idea to use the first option there as well. It would seem that the only situation where the second option is better is when there are intricate surfaces behind the holes where their removal would save a lot of rendering time.

5.8.3. Visibility from a specific viewing angle

The algorithms we have described so far determine the visibility of parts and surfaces of an assembly from any angle so those that are invisible are removed from the assembly, reducing the total number of parts to be displayed. This is the purpose of this

study. However, since we are already using AABBs in our method, then possibilities for their use in other aspects could be explored to see if they lend an advantage. One such possibility is visibility determination from a specific viewing angle. This would be used in the actual display method to determine which surfaces of those determined visible by the main algorithm are to be rendered from the specific viewpoint determined by the user. Much research has already been done on viewing angle visibility, but because the AABBs are already calculated for the algorithms in this research, it may be advantageous to use them in viewing angle visibility if they result in fast and accurate algorithms.

This approach is analogous to the use of Z-buffers to determine what is to be rendered on the display. Z-buffers take the triangles that are tessellated from surfaces and, going from those triangles furthest from the viewpoint to those that are closest, determine what should be displayed on the screen. In effect, it keeps track of the visible parts of each triangle from the chosen viewing angle. In essence, this approach will be doing the same thing, going from the furthest AABB to the viewpoint to the closest and keeping track of the visible parts of the AABBs. The only difference being that the AABBs only determine which parts will be rendered, not what each pixel should be displaying. The advantage of this is that the calculation of which parts will be rendered will save in the rendering process itself, which requires much more processing.

We are only exploring the use of AABBs in a viewing angle that is in a direction parallel to one of the global axes. We make this restriction because parallel viewing directions result in the AABBs becoming rectangles. Non-parallel viewing directions cause the AABBs to appear as hexagons, whose intersections are much more complex,

and would require more research to determine how to compute their intersections. Example shapes are shown in Figure 5.28. Generalization to the following procedure could be used for this determination from the non-parallel angles.

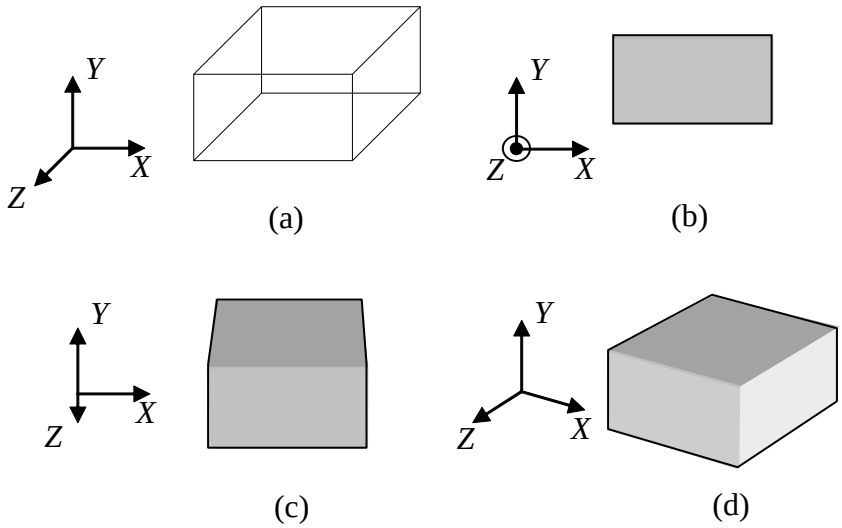


Figure 5.28. Example of the shapes that result from different viewing angles. (a) AABB of a part. (b) View from a direction parallel to an axis. (c) View from a direction orthogonal to an axis, but not parallel to an axis. (d) View from a direction not orthogonal to any axis

From a viewpoint orthogonal to one of the global axes, the AABBs that surround parts or surfaces appear as rectangles. We will use these rectangles to represent their corresponding surfaces or parts. As an example, let us assume that the viewing direction is parallel to the *Z*-axis. Each rectangle can be specified by two coordinates, (X_{min}, Y_{min}) and (X_{max}, Y_{max}) as shown in Figure 5.29. Thus, each AABB has two rectangles associated with it, as there are two faces orthogonal to the viewing direction. In addition, each rectangle has a *z*-coordinate associated with the plane of the face of the AABB from which the rectangle is obtained. The viewpoint also has a *z*-coordinate as well. These values will be used to determine the distance between the point of view and the rectangles. Only those rectangles that have a *z*-value in the

direction of the viewpoint are considered for visibility, since those that have a *z*-value in the other direction are behind the viewpoint and thus, invisible. For those AABBs that have two rectangles in the viewing direction, we only consider the one that is closer to the viewpoint, as it is the face of the AABB that is visible. Compiling all these rectangles, we get a list specified by the coordinates of two opposite corners, and their corresponding *z*-coordinates.

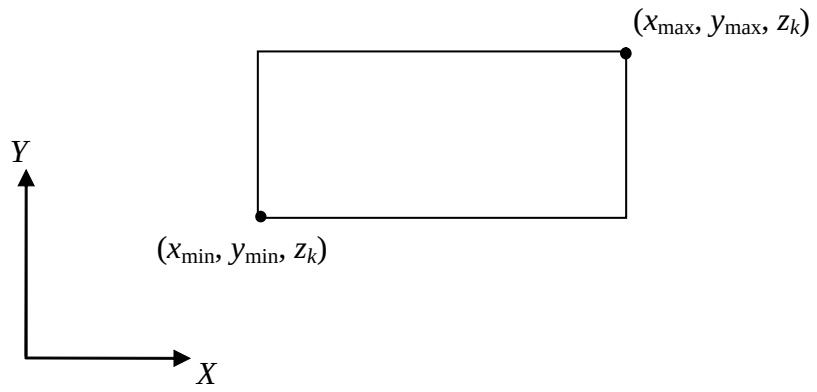


Figure 5.29. A rectangle and the two points that define it.

To determine the visibility of these rectangles, we determine all portions of the rectangles that are visible, assuming that portions of rectangles are hidden by those rectangles closer to the viewpoint that share those portions. To do this, we determine the visible portions of all rectangles by starting with the farthest rectangle from the viewpoint and moving forward towards the viewpoint. Beginning with the first rectangle, we add the next closest rectangle to it. This rectangle may or may not overlap with the previous rectangle. We keep track of only the portion of the first rectangle that does not share the same area as the second, the uncovered portion. Then, we consider the next rectangle, and keep track of the uncovered portions of those previously uncovered areas of the first two rectangles that do not overlap. We continue

until we reach the closest rectangle to the viewpoint. In this way, we will have determined the visible areas of all the rectangles.

We start by sorting the list of rectangles by their distances from the viewpoint, with the farthest rectangle(s) occupying the first elements of the list. We will call this list L_r. We then create a separate list L_v of all the visible portions of the rectangles. First, we place the first rectangle from L_r into L_v. Then, we take the next rectangle in L_r. We call R_r the current rectangle in L_r and R_v the current rectangle in L_v, which is the single rectangle that was the first rectangle in L_r. R_r is closer to the viewpoint than R_v, and may or may not overlap. If the area of R_v is obscured by R_r, then R_v's entry in L_v is modified to keep track of the area that doesn't overlap R_r. This area may not be rectangular. If it is not rectangular, it can always be split into several rectangles based on its shape, resulting in multiple entries in L_v. This is described in further detail below. It is also possible that R_v is completely overlapped by R_r, meaning that R_v is no longer visible. In this case R_v is removed from L_v. If there is no overlap, then R_v's entry is unchanged. Next, we add R_r to L_v and set R_r as the next rectangle in L_r. We repeat the process by cycling through all the rectangles in L_v instead of performing the process on a single rectangle as R_v. We are thus calculating the visible portions of all the rectangles in L_v when R_r is placed in front of them. This process continues until we have considered the last rectangle in L_r. The final result is a list L_v of visible rectangles. Finally, we make all the parts or surfaces (depending on what the AABBs pertain to) whose rectangles appear in list L_v visible as well.

We now describe the method used to determine the portion of R_v that is uncovered. The coordinates that specify R_r are $(x_{r,min}, y_{r,min})$ and $(x_{r,max}, y_{r,max})$ and those

that specify R_v are ($x_{v,min}$, $y_{v,min}$) and ($x_{v,max}$, $y_{v,max}$). Each point (x, y) in R_r satisfies the conditions $x_{r,min} < x < x_{r,max}$ and $y_{r,min} < y < y_{r,max}$. Similarly, each point (x, y) in R_v satisfies the conditions $x_{v,min} < x < x_{v,max}$ and $y_{v,min} < y < y_{v,max}$. The four possibilities for the x-intervals of the two rectangles are shown in Table 5.10. The same possibilities apply for the y-intervals of the rectangles as for the x-intervals by simply replacing the x values in Table 5.10 with y values. The relationships between the coordinates of the rectangles in x and y determine the areas of the rectangles that overlap.

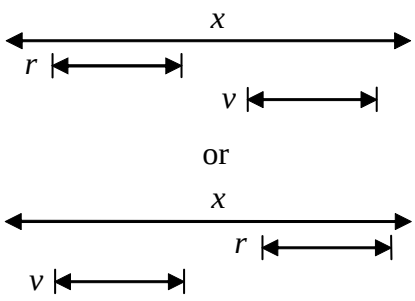
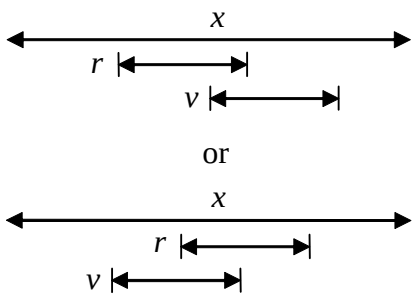
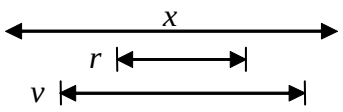
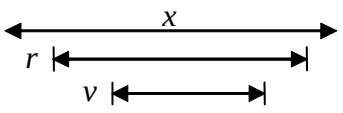
Using the relationships in Table 5.10 for x and y, we can determine the area of R_v not shared by R_r. If R_v and R_r are disjoint in either x or y, then they do not overlap. When R_v is disjoint with every entry in L_v, L_v is not modified. If the rectangles are not disjoint, then the remaining possibilities are that R_r intersects, is contained in, or contains R_v in x and/or y. We will explore each case separately.

The first case is that R_r intersects R_v in both x and y. Referring to Figure 5.30, there are two possibilities for the x intervals. Possibility 1, which we will call $X1$, is satisfied by $x_{r,min} < x_{v,min} < x_{r,max} < x_{v,max}$. Possibility 2, called $X2$, has the condition $x_{v,min} < x_{r,min} < x_{v,max} < x_{r,max}$. Similarly, the two possibilities for y intervals are $Y1$: $y_{r,min} < y_{v,min} < y_{r,max} < y_{v,max}$; and $Y2$: $y_{v,min} < y_{r,min} < y_{v,max} < y_{r,max}$. These four combinations leave an L-shaped area of R_v uncovered. This shape can be broken into two rectangles; therefore, we change R_v's entry in L_v to one of these pieces and also add the other piece to L_v.

Another relationship that could occur is for R_r to be intersecting R_v in either x or y, and for R_r to be contained in R_v in the other direction. There are also four possibilities for this occurrence shown in Figure 5.31. Each of the four possibilities is

specified by which relationship from the previous paragraph is the intersecting one: $X1$, $X2$, $Y1$, or $Y2$. In all four possibilities, the non-overlapping portion of R_v can be split into three rectangles.

Table 5.10. Possible cases for the x-intervals of two rectangles

Case	x-interval properties	x-interval coordinate properties
Disjoint in x		$x_{r,max} < x_{v,min}$ or $x_{v,max} < x_{r,min}$
Intersecting in x		$x_{r,min} < x_{v,min} < x_{r,max} < x_{v,max}$ or $x_{v,min} < x_{r,min} < x_{v,max} < x_{r,max}$
R_r "contained in" R_v in x		$x_{v,min} < x_{r,min} < x_{r,max} < x_{v,max}$
R_r "contains" R_v in x		$x_{r,min} < x_{v,min} < x_{v,max} < x_{r,max}$

Another possible relationship is again to have R_r intersect R_v in either x or y, but have R_r contain R_v in the other direction. This again has four possibilities, as shown in Figure 5.32, with each possibility again specified by intersecting in either $X1$, $X2$, $Y1$, or $Y2$. The region of R_v that is not shared thus turns out to simply be a single rectangle.

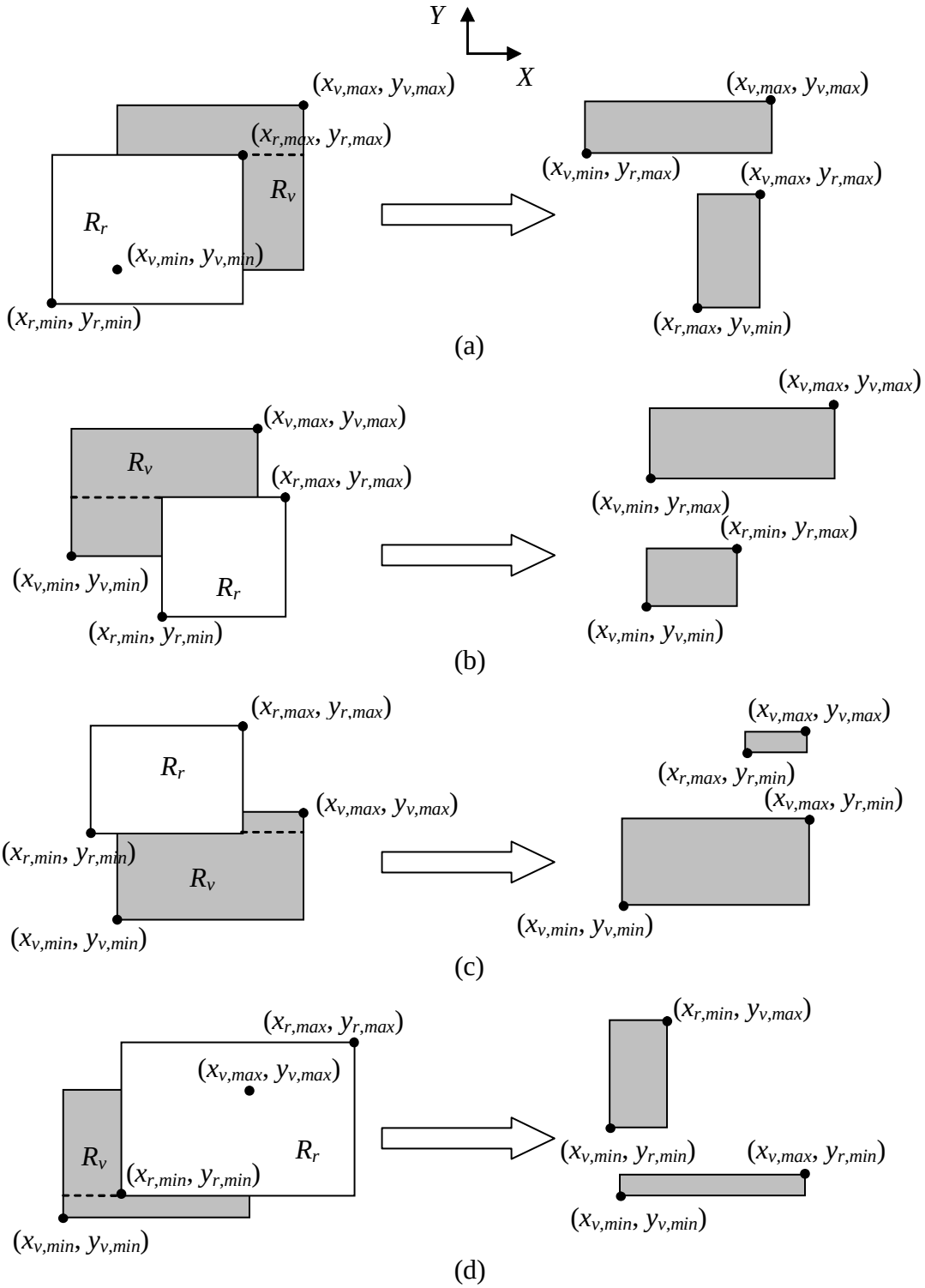


Figure 5.30. The four possibilities for R_r intersecting R_v in both x and y. (a) $X1$ and $Y1$, (b) $X2$ and $Y1$, (c) $X1$ and $Y2$, (d) $X2$ and $Y2$. Dark grey area is kept in L_v. Dotted line splits dark area into two rectangles.

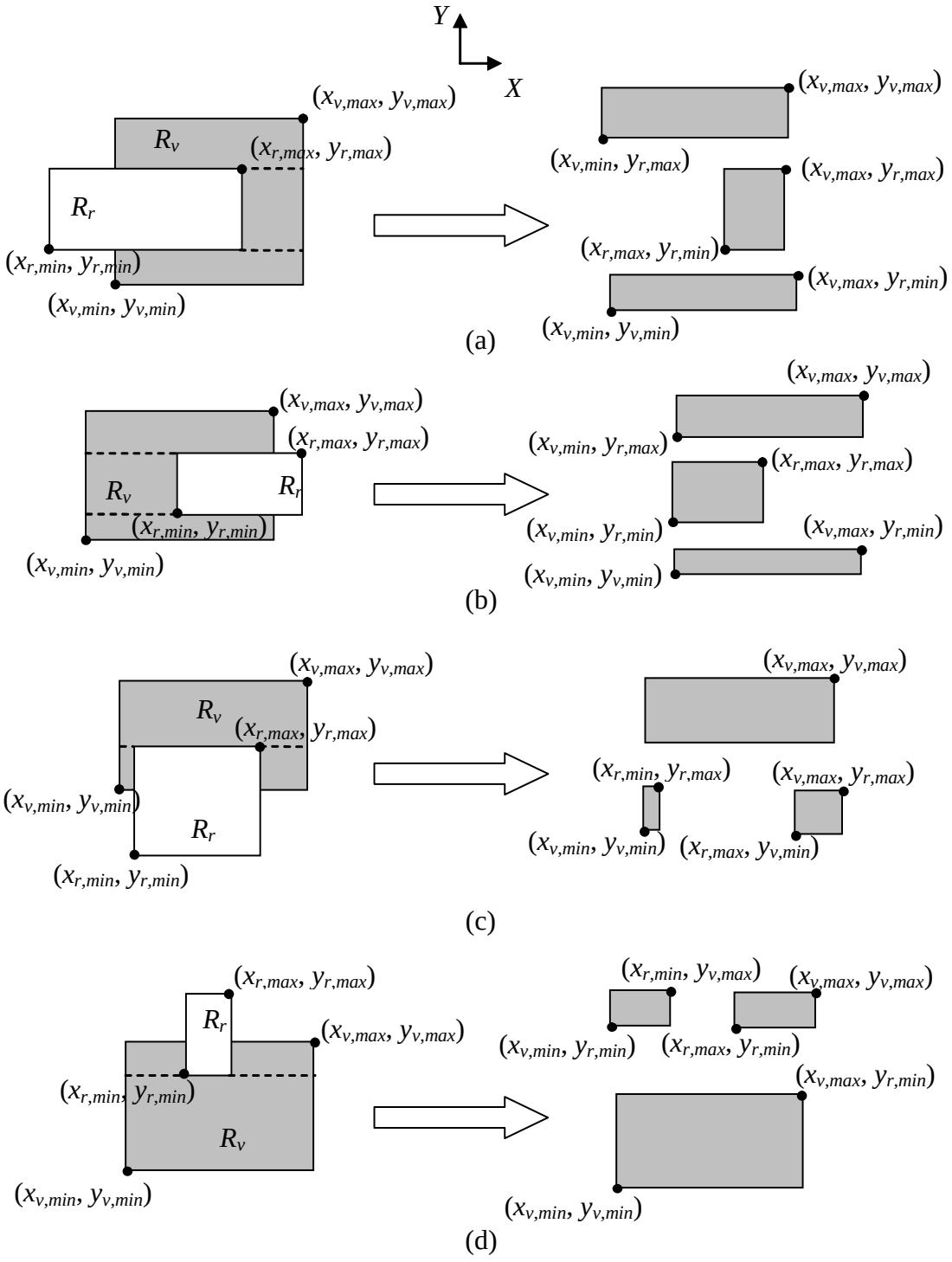


Figure 5.31. The four possibilities for R_r intersecting R_v in either x or y and R_r contained in R_v in the other direction. Examples for when the intersecting possibility is (a) $X1$, (b) $X2$, (c) $Y1$, (d) $Y2$. Dark grey area is kept in L_v. Dotted lines split dark area into three rectangles.

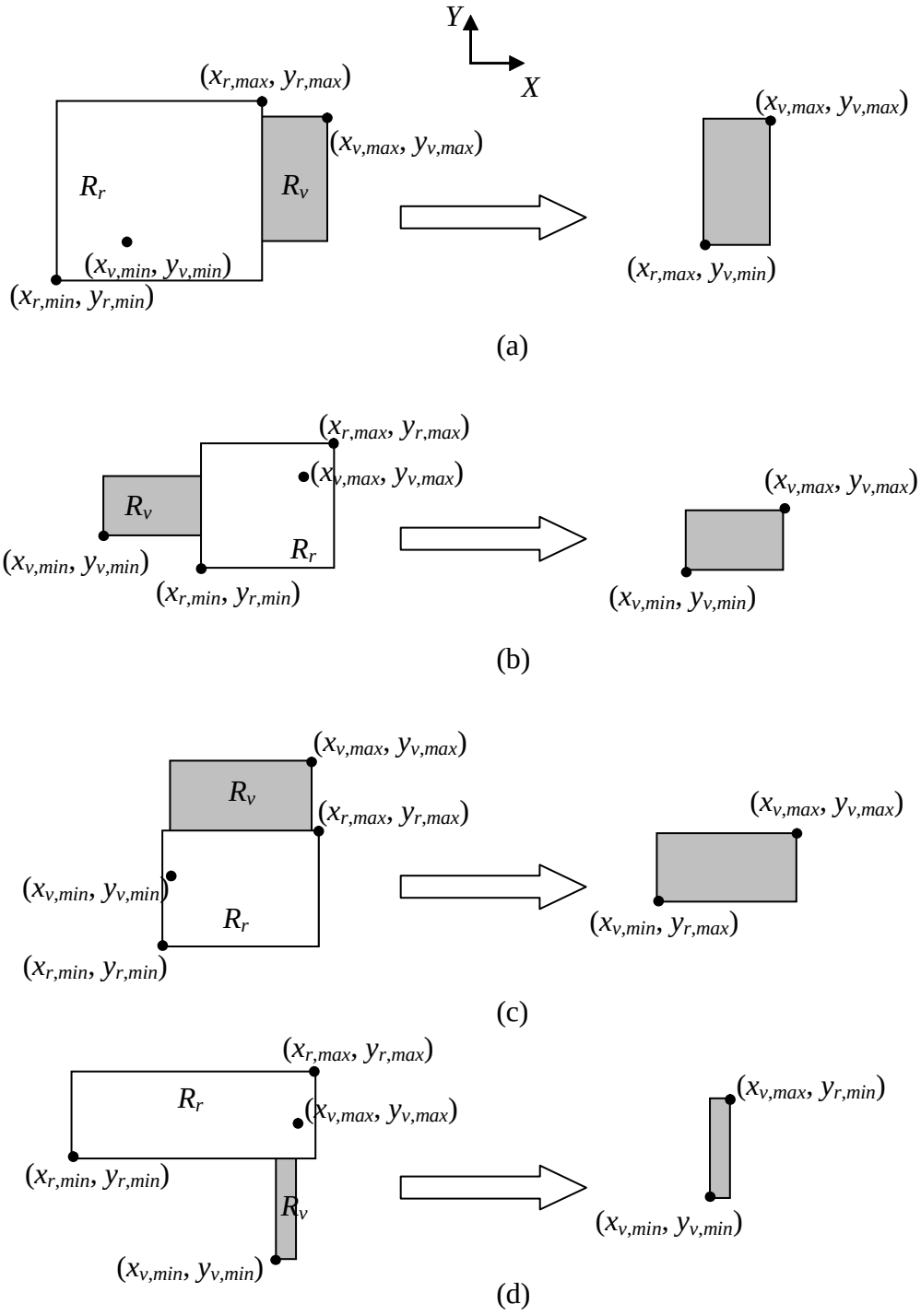


Figure 5.32. The four possibilities for R_r intersecting R_v in either x or y and R_r contains R_v in the other direction. Examples for when the intersecting possibility is (a) $X1$, (b) $X2$, (c) $Y1$, (d) $Y2$. Dark grey area is kept in L_v.

Another possibility is for R_r to be contained in R_v in both x and y. This results in the regions shown in Figure 5.33, with the area of R_v to remain surrounding R_r. This area can be split into four rectangles.

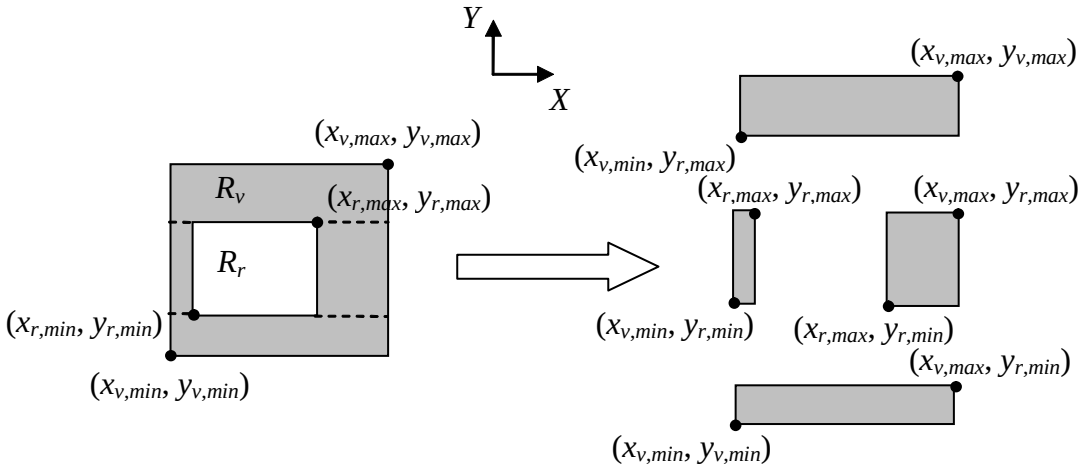


Figure 5.33. R_r contained in R_v in both x or y. Dark grey area is kept in L_v. Dotted lines split grey area into four rectangles.

The next possibility is for R_r to be contained in R_v in x and contain R_v in y and vice versa. This results in the two possibilities shown in Figure 5.34. In this case, R_r splits R_v's entry in L_v into two rectangles.

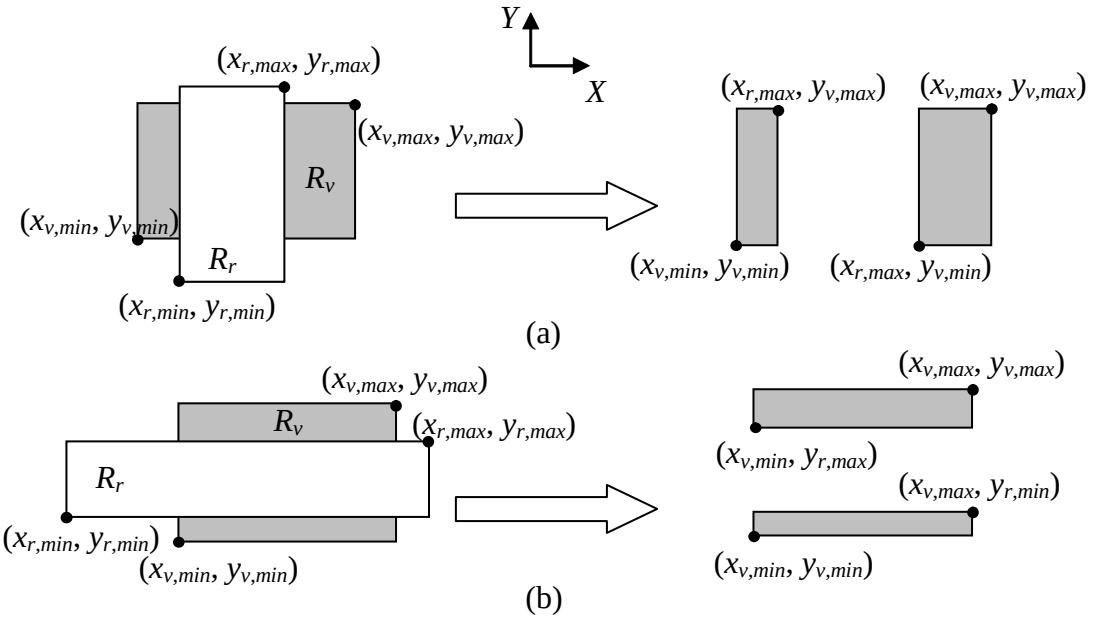


Figure 5.34. The two possibilities for R_r contained in R_v in either x or y and R_r containing R_v in the other direction. (a) R_r contained in R_v in x (b) R_r contained in R_v in y. Dark grey areas are kept in L_v.

The final possibility for the relationship between R_r and R_v is that R_r contains R_v in both x and y. When this occurs, we see that R_r completely overlaps R_v, as shown in Figure 5.35. Thus, R_v's entry in L_v is removed.

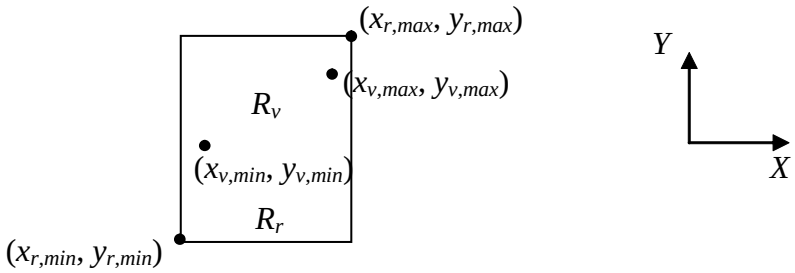


Figure 5.35. R_r contains R_v in both x or y.

After the process of analyzing all of the rectangles in L_v, L_v becomes a list of all of the portions of all the rectangles that are visible from the viewpoint.

This method has the same shortcomings that the AABB algorithm has. The empty space contained in a rectangle is treated as a solid and, therefore, it can cause a portion of a rectangle to be marked invisible when it is not.

5.9. Summary

In this section, the details for determining the visibility of an AABB are given. This is the last process required to determine what the display routines should render. First, the process for determining the visibility of non-contained AABBs was presented. Then, a rule was applied to determine the visibility of contained AABBs. The rendering routine uses the visibility of the parts to determine which parts should be displayed. The next section discusses the verification and validation of the process, from the determination of the AABBs to the determination of visibility.

6. An Application: SCAMP

The verification and validation of the research described in Chapters 3-5 was performed on CAD models of NASA's Supplemental Camera And Maneuvering Platform (SCAMP) telerobot shown in Figure 6.1 and Figure 6.2. This robot is the result of a joint effort between the Space Systems Laboratory of the University of Maryland and NASA Johnson Space Center. The SCAMP telerobot is an experimental neutrally buoyant teleoperated vehicle that is designed to help further space vehicle research. The model consists of 490 parts, of which 202 are visible from the outside of the robot and 288 are invisible. Verification and validation were performed for each stage of the algorithm to ensure that the algorithms functioned as intended.

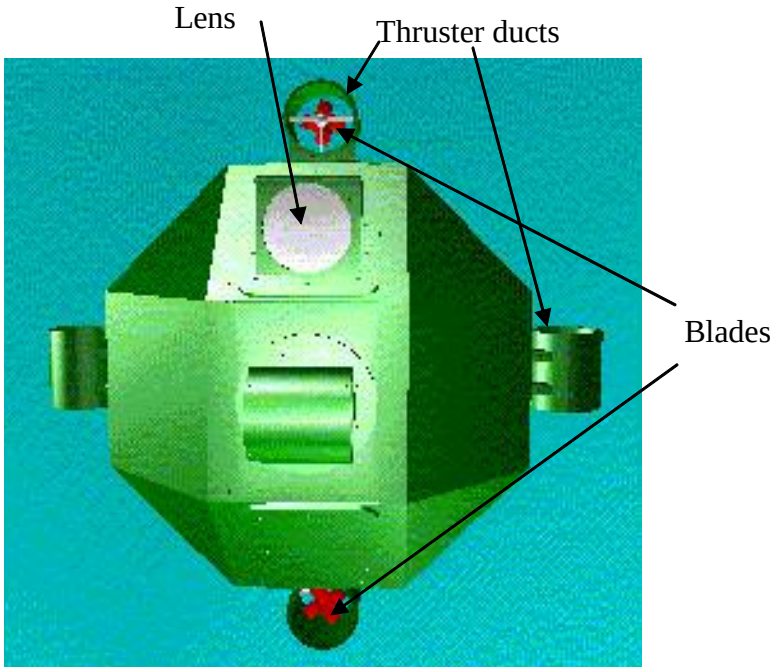


Figure 6.1. CAD model of SCAMP telerobot.

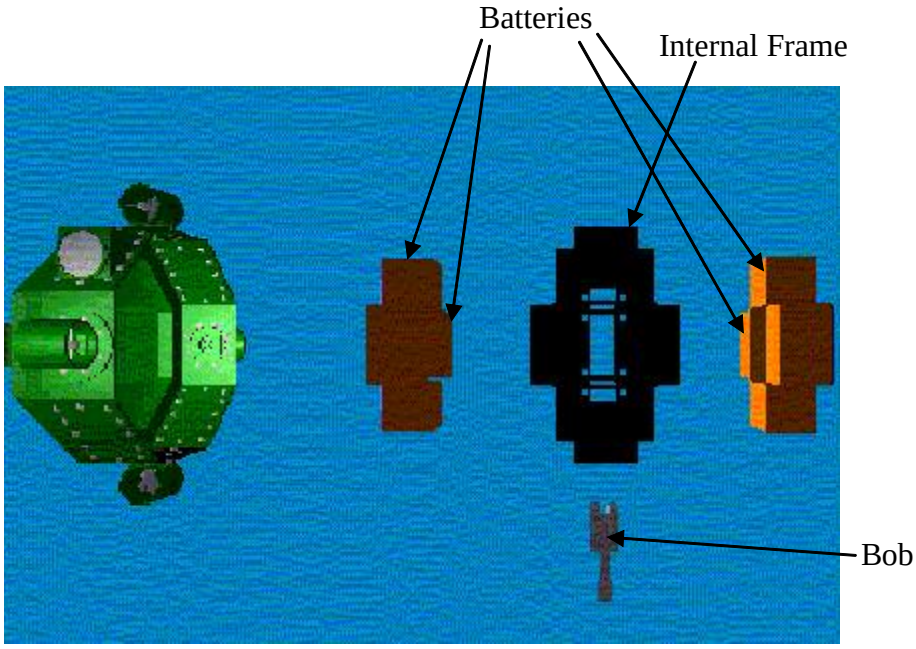


Figure 6.2. Interior parts of the SCAMP model.

6.1. Preparation of the Model

The model existed as parts in the CAD package Pro/Engineer, a software modeling package. This software package has the capability of translating models from its proprietary format to the IGES format. The algorithms described in Chapters 3-5 were then used on the IGES format of these models to determine visibility.

6.2. AABB Determination

First, we verify that the determination of AABBs is correct. This determination begins with the calculation of the AABBs of each surface of a part. To verify that the surface AABBs are correct, the validation must be performed manually by looking at them through a visualization or CAD program. The coordinates of each surface AABB are output from the developed algorithm. Then, using Pro/Engineer, each AABB is

manually created from the coordinates. Once this is done, each AABB and its original part are displayed together to verify that the AABB determination was correct. An example of what is displayed is shown in Figure 6.3 for one of the thruster ducts on the vehicle. Each surface box is represented by a different color. It was verified that each of the surface boxes generated was correctly determined for several randomly selected parts. (This process was limited by Pro/Engineer's inability to make parts partially invisible. This made it hard to visually confirm that the AABB determination was correct; hence not all parts were verified). Once the surface AABB determination was verified, the next step was to confirm the validity of the part AABB that is calculated based on the surface AABBs. Thus, for each part whose surface AABBs were verified, the AABB of the entire part, which was determined by the algorithm presented in Chapter 3, was also manually created in Pro/Engineer. For example, the resultant AABB for the thruster duct is shown in Figure 6.4. It was verified that these AABBs consisted of the extremes of all of the surface AABBs and were the minimum surrounding AABBs of these parts. Through Pro/Engineer, the AABBs from several other components were used to verify that their AABBs were correct.

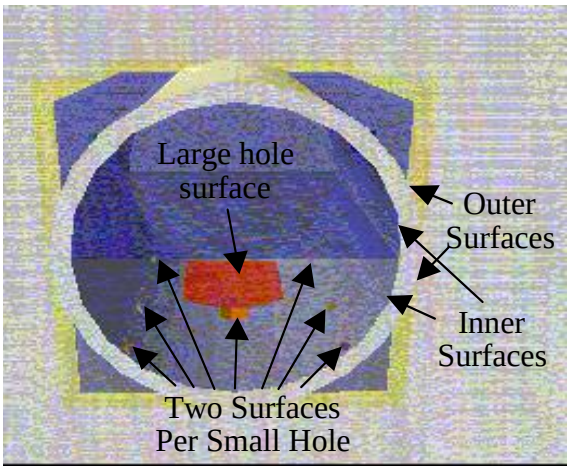


Figure 6.3. Surface AABBs for thruster duct.

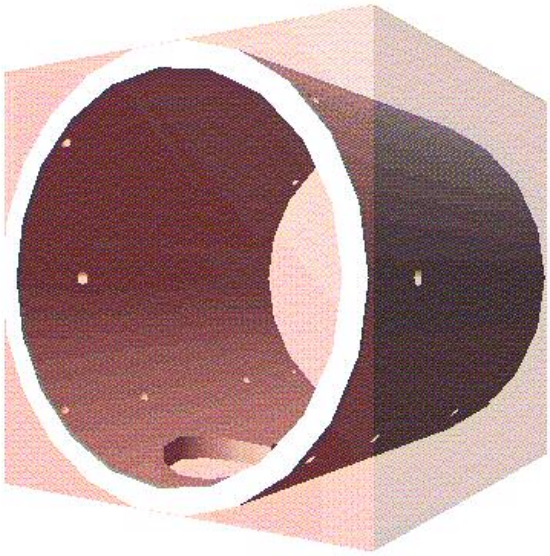


Figure 6.4. AABB for the thruster duct.

6.3. Relationship Determination

The determination of three classifications of relationships between AABBs, as described in Chapter 4, has been implemented:

- One AABB contained in another
- One AABB intersecting another
- One AABB adjacent to another

Applying the program to the SCAMP model yields relationships between all of the AABBs. These relationships were output from the algorithm presented in Chapter 4, and a simplified relationship tree was generated manually, which is shown in Figure 6.5. In this example, only interior parts are shown and identical parts have been grouped together to make it easier to view the relationships. For example, the four batteries in the vehicle have been grouped together in the figure.

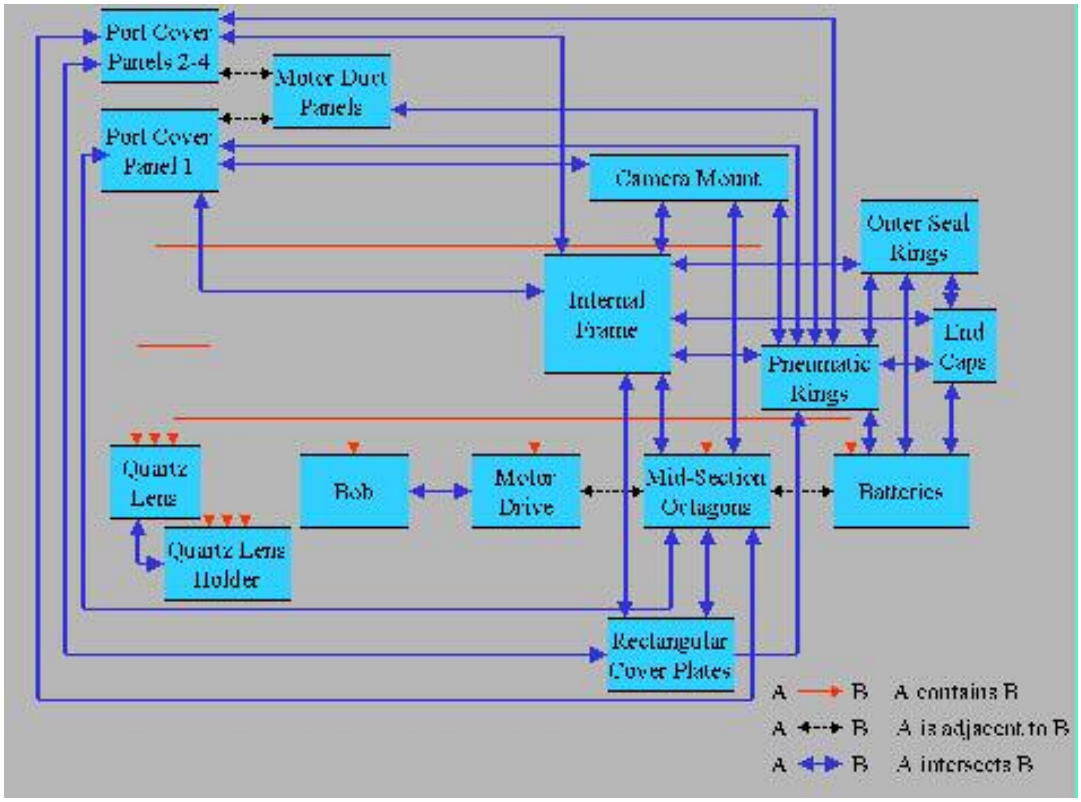


Figure 6.5. Relationship tree for selected parts of the vehicle.

These relationships were verified visually with representations of the AABBs in the modeling program. For example, Figure 6.5 indicates that the AABB of Port Cover Panel 1 contains the AABBs of both the quartz lens and the quartz lens holder. This was verified using Pro/Engineer, as shown in Figure 6.6. The AABB for the Port Cover Panel 1 (blue) contains the AABBs for the lens (green) and lens holder (red). Figure 6.5

shows that the AABBs for the batteries intersect the outer seal rings' AABBs. Figure 6.7 shows a battery AABB intersecting the AABB for one of the rings. Finally, the motor drive AABB is adjacent to a mid-section octagon's AABB, as shown in Figure 6.8.

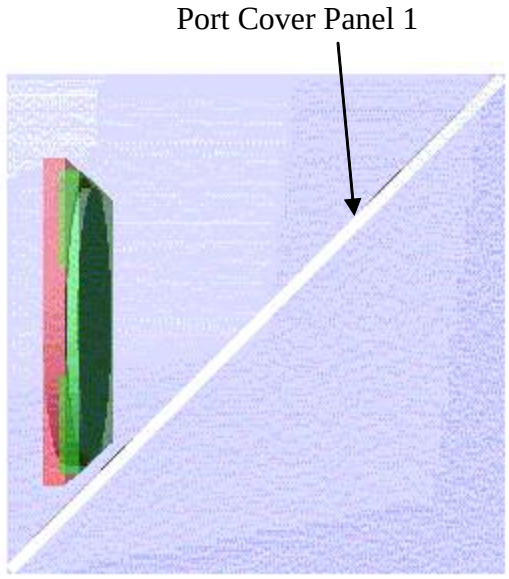


Figure 6.6. Side view of the AABBs for the quartz lens (green) and the quartz lens holder(red) contained in the AABB for Port Cover Panel 1 (blue).

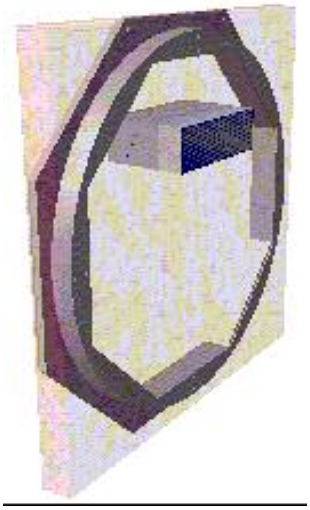


Figure 6.7. AABBs for the intersection of an outer seal ring (pink) and a battery (grey).

Figure 6.8. The two adjacent AABBs of the mid-section octagon (red) and the motor drive (blue).

6.4. Object Visibility Determination Trace

Visible objects were computed using a trace around a cross section of the AABBs as described in Chapter 5. To verify that the trace is correct, a visual validation was performed using an Excel spreadsheet. For several cross sections, the coordinates of all of the AABBs encountered were input into the spreadsheet and then used to create a visual representation of the rectangles in the profile. For an example, consider the section that goes through the parts shown in Figure 6.9. The corresponding rectangles have the corner coordinates in Table 6.1, which has the profile shown in Figure 6.10. Each AABB is represented by a different color rectangle. The coordinates of the trace generated by the algorithms are used to superimpose a trace on this profile shown by dotted black lines. Using this method, we visually and numerically verify that the exterior edge trace is correct because it does not appear on any of the interior edges.

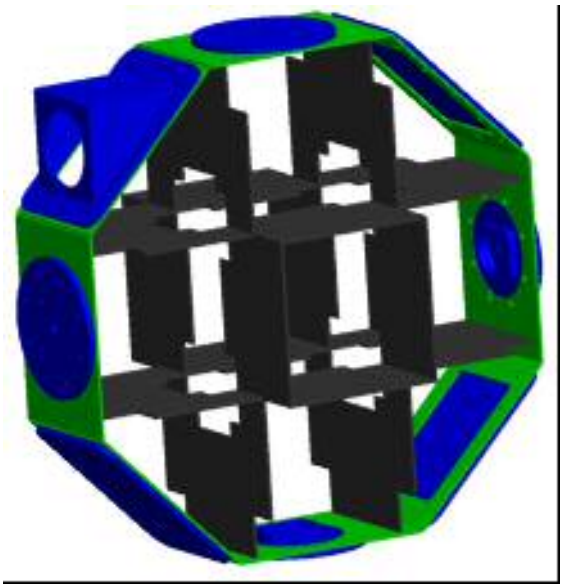


Figure 6.9. Parts through which cross-section is taken.

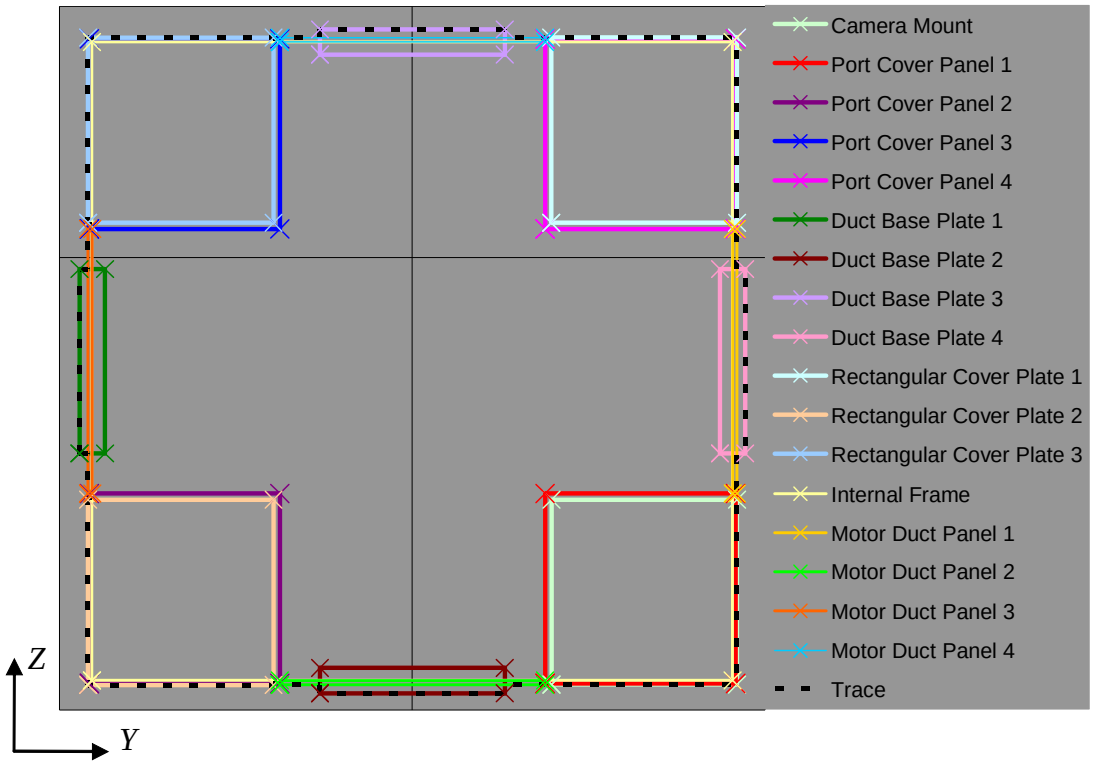


Figure 6.10. Visibility exterior edge trace of example cross-section profile.

Table 6.1. Rectangles and their coordinates		
Rectangle Part	Lower Left Coordinate (Y^{min}, Z^{min})	Upper Right Coordinate (Y^{max}, Z^{max})
Camera Mount	(4.146, -12.742)	(9.663, -7.225)
Port Cover Panel 1	(3.955, -12.712)	(9.625, -7.041)
Port Cover Panel 2	(-9.612, -12.711)	(-3.941, -7.041)
Port Cover Panel 3	(-9.611, 0.855)	(-3.941, 6.526)
Port Cover Panel 4	(3.955, 0.855)	(9.626, 6.525)
Duct Base Plate 1	(-9.898, -5.843)	(-9.143, -0.343)
Duct Base Plate 2	(-2.743, -12.998)	(2.757, -12.243)
Duct Base Plate 3	(-2.743, 6.056)	(2.757, 6.811)
Duct Base Plate 4	(9.157, -5.843)	(9.912, -0.343)
Rectangular Cover Plate 1	(4.138, 1.039)	(9.664, 6.565)
Rectangular Cover Plate 2	(-9.65, -12.751)	(-4.124, -7.225)
Rectangular Cover Plate 3	(-9.651, 1.038)	(-4.125, 6.564)
Internal Frame	(-9.532, -12.618)	(9.532, 6.437)
Motor Duct Panel 1	(9.537, -7.041)	(9.662, 0.855)
Motor Duct Panel 2	(-3.941, -12.748)	(3.955, -12.623)
Motor Duct Panel 3	(-9.648, -7.041)	(-9.523, 0.855)
Motor Duct Panel 4	(-3.941, 6.437)	(3.955, 6.562)

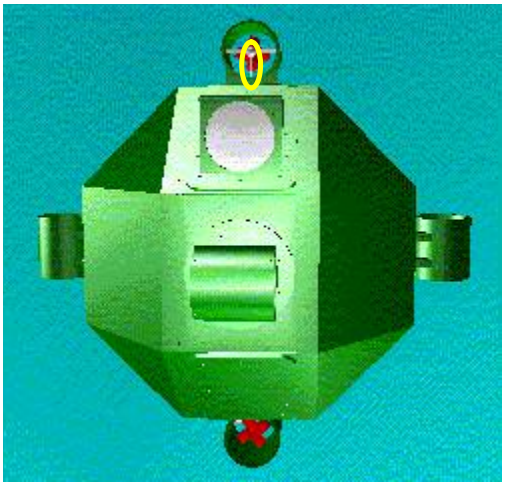


Figure 6.11. One of the pipes that was incorrectly marked invisible, which is circled in yellow.

6.5. Validation of the Visibility Detection Algorithm

The final verification comes from the overall outcome of the program, which is the determination of the components that are visible and those that are not. This

validation is based on the human visual perception of what components are visible from any viewing angle. Of the 490 parts in the vehicle, 202 of them are visible from the outside of the vehicle, leaving 288 parts hidden in the interior. The visibility as determined by the program showed that it was able to correctly mark invisible all 288 parts. In addition, it detected as visible 200 of the 202 visible parts on the vehicle. The 2 components that were incorrectly made invisible are 2 of 6 small pipes, each being contained in one of the 6 exterior thrusters on the vehicle. One of these pipes is shown circled in yellow in Figure 6.11. These pipes were marked invisible because their AABBs are contained in a collection of AABBs, but not in any single AABB. Any part whose AABB fits this criteria will not be detected as visible, as the exterior traces performed on the cross section profiles will go around the rectangles surrounding the current part's rectangle and miss this AABB.

To validate the performance advantage that the program provides during simulation, we created visualizations and simulations of the underwater vehicle assembly model. A vehicle visualization shows the transition of the model from a wire-frame view to a rendered view. Also, the simulations depicted the actions of the vehicle flying about in a neutral buoyancy tank in search of a lost thruster. One visualization was created for the original assembly model and for the simplified model. This visualization simply involved changing from a wire frame view to a solid shaded view. In addition, one simulation was created for the original assembly model and for the simplified model; that is, one whose parts were marked visible by the visibility algorithm. This simulated the process of the robot swimming around an object to scan it with its camera.

The system process time was examined for the simulation program while it was performing each visualization and each simulation. This process time is a measure of how much time the CPU spends making calculations necessary for the display of the assemblies and their movement. Therefore, a comparison between the original and simplified vehicles can be made to assess the performance benefit of the algorithm developed in Chapters 3-5.

6.6. Results

The results given in this section are for the SCAMP telerobot model, which will vary with other assemblies since the values will depend on the specific geometry of the assemblies. Assemblies that do not have many hidden parts will not exhibit significant savings, while those with many hidden parts will show dramatic increases in simulation speed.

The accuracy rate of the program was found to be 99.6% for this assembly model, as 488 out of the 490 components' visibilities are correct. However, from a qualitative standpoint, the two incorrect parts have little practical visual consequence in the simulation of this vehicle. Also, in terms of calculation time, averaging ten runs, the time to calculate the AABBs for all 490 components on a Pentium 4 2.4 GHz machine is 37.375 ± 0.0314 seconds, an average of 0.0763 seconds per part. By relative comparison, the visibility calculation using the AABBs is almost one thousand times faster, taking only 0.041 seconds to determine the visibility of all the parts.

The results of the created simulations are summarized in Table 6.2. They indicate that the algorithm increased the simulation and visualization speeds at which

the rendered SCAMP robot model is displayed. For visualization, the speed is 2.2 times faster than that of the original; that is, the original visualization required 2.2 times more CPU time than the model simplified by the algorithms presented here. Also, for simulation, the speed of the developed algorithm is 1.8 times faster than that used by the original method. This verifies the initial intent of the algorithms, which was to provide faster simulation and visualization times for complex models while maintaining visual accuracy. Although these results may vary for different assemblies, they show the potential that the algorithms have for making the display and simulation of complex assemblies faster.

Table 6.2: Results of simulation and visualization of the SCAMP robot.

Model Type	Visualization CPU Time (seconds)	Simulation CPU Time (seconds)
Original	33	126
Simplified by Present Algorithm	15	70
Performance Benefit of Present Algorithm (Original time over Present Algorithm time)	2.2	1.8

6.7. Discussion

The results of using the algorithms on a real-world application are promising. The correctness of the algorithm in determining the visibility of the parts in the SCAMP robot, while not necessarily indicative of how well the algorithms will perform on other applications, shows that there is merit to this approach. In addition, the potential for

savings in not rendering invisible parts is a major benefit for those assemblies that have numerous interior parts.

One area that may be of concern is the processing time required for the algorithms. The AABB determination required 37 seconds to complete, which is somewhat long considering that the time saved for the simulation is 56 seconds. However, this is a little misleading, as the entire 37 seconds is not needed for further use of the models. First, this is the pre-processing time, and thus only needs to be done once for the same configuration of the model. If the same configuration of the SCAMP is used several times in different simulations, or the same simulation is viewed multiple times, the savings increases dramatically. In addition, since the AABB has already been found for each part, if the configuration of the model changes, it is only necessary to calculate the AABBs for the parts that were changed. Thus, the AABB determination will not require the entire 37 seconds for further iterations of the SCAMP. The whole process is even benefited more by the fact that the visibility determination after the AABBs are calculated is extremely fast, using only 0.041 seconds to calculate the visibility of all 490 parts. As such, a change in configuration will still have nearly instantaneous results after the AABBs for the changed parts are calculated. Thus, using the algorithms on models that change frequently is beneficial after using the algorithms on the first version of the models, as the visibility of subsequent versions can be determined quickly.

6.8. Summary

This section presented an application of the new algorithms. First, the determination of the AABBs was verified. Then, we validated the determination of the

relationships between the AABBs. Next, the cross-section determination algorithm was verified. Finally, the visibility determination of the parts in the assembly and the performance increase that results from the use of the new algorithm was illustrated.

7. Conclusion

It has been demonstrated that the algorithms developed in this thesis can be used to determine the visibility of parts in an assembly. The significant contributions of the research are as follows:

1. The most significant contribution is the use of the cross-section edge trace. It makes the determination of visibility much faster than ray tracing, which is a standard way of determining visibility of parts without rendering all of them.
2. Considerable savings can be made in processor time during the display of large assemblies, depending on the geometry of the assembly.
3. The pre-processing nature of the algorithms means that they only need to be done once for the same configuration of the model, no matter how many times it is used in various simulations or how many times the model is viewed. This can result in considerable savings in the processing time needed to render the parts.
4. For subsequent small changes to the models, only the AABBs for the parts that have been changed need to be found. Thus, the time to process these iterations is even shorter than the initial run of the algorithms.

Straightforward Extensions to the algorithms

1. The visibility determination algorithms, after the AABBs have been found, are extremely fast. As such, if the modeling program already incorporates AABB determination, or if it can be incorporated easily into the program, then the visibility algorithms can be performed with virtually no performance degradation.
2. The current visibility detection scheme can be modified easily to allow for more accuracy as well as to work on the visibility of surfaces instead of parts.

3. Because AABBs are used in these algorithms, one can also make use of them for other purposes, such as visibility determination based on different viewing angles.

8. References

Armstrong, C.G., McKeag, R. M., Ou, H., and Price, M. A. (2000). Geometric processing for analysis. IEEE Proc. Geometric Modeling and Processing: Theory and Applications, pages 45-56.

Bittner, J., Havran, V., and Slavik, P. (1998). Hierarchical visibility culling with occlusion trees. Proc. Computer Graphics International, pages 207-219.

Bormann, K. (2001). Occlusion culling in large virtual environments. Presence: Teleoperators & Virtual Environments, 10(5): 477-494

Brodsky, D. and Watson, B. (2000). Model simplification for interactive applications. Proc. IEEE Virtual Reality, page 286.

Durand, F., Drettakis, G., Thollot, J., and Puech, C. (2000). Conservative visibility preprocessing using extended projections. Proc. Siggraph, pages 239-248.

Hoff III, K. E. (1997). Faster 3D game graphics by not drawing what is not seen. ACM Crossroads, 3(4).

Hudson, T., Manocha, D., Cohen, J., Lin, M., Hoff, K., and Zhang, H. (1997). Accelerated occlusion culling using shadow volumes. Proc. of ACM Symposium on Computational Geometry.

Huerta, J., Chover, M., Ribelles, J., and Quirós, R. (1998). Multiresolution modeling using binary space partitioning trees. Computer Networks and ISDN Systems, 30(20-21): 1941-1950.

IGES/PDES Organization. (1988). IGES 4.0 Specification.

Iones, A., Zhukov, S., and Krupkin, A. (1998). On optimality of OBBs for visibility tests for frustum culling, ray shooting and collision detection. Proc. Computer Graphics International, pages 256-63.

Kitamura, Y. (1998). A Real-Time Algorithm for Accurate Collision Detection for Deformable Polyhedral Objects. Presence: Teleoperators & Virtual Environments, 7(1): 36-52.

Krus, M., Bourdot, P., Guisnel, F., Thibault, G. (1997). Level of Detail & Polygonal Simplification. ACM Crossroads, 3(4).

Kumar, S., Manocha, D., Garrett, W., and Lin, M. (1996). Hierarchical backface computation. Proc. of 7th Eurographics Workshop on Rendering.

Levi, O., Zohar, R., Barad, H., and Klimovitski, A. (1999). A compact method for backface culling. Gamasutra, 3(31). http://www.gamasutra.com.

Martin, R. R. and Stephenson, P. C. (1998). Putting objects into boxes. Computer Aided Design, 20(9): 506-514.

Möller, T. and Haines, E. (1999). Occlusion culling algorithms. Gamasutra. http://www.gamasutra.com.

Sanna, A. and Montuschi, P. (1995). Spatial bounding of complex CSG objects. IEEE Proc. Computers and Digital Techniques. 142(6): 431-439.

Shaikh,S., Magrab, E., Hagner, J., and Ou, J. (2000). Simulation Simplification Tool. Final Report NAS3-00078. Technology Promotion International, Ltd, College Park, MD. Report # NASA SBIR 99.1.1. June 2000.

Suri. S., Hubbard. P. M., and Hughes. J. F. (1999). Analyzing bounding boxes for object intersection. ACM Transactions on Graphics, 18(3): 257-277.

Teller, S. and Séquin, C. (1991). Visibility preprocessing for interactive walkthroughs. Computer Graphics, 25(4): 61-69.

Yu, Y., Wu, M., and Zhou, J. (1996). An octree algorithm for dynamic interference detection using space partitioning. Proc. Of the Design Engineering Technical Conference: Design Automation Conference.

Zachmann, G. (1997). Real-time and exact collision detection for interactive virtual prototyping. Proc. Of Design Engineering Technical Conferences: Computers in Engineering.

Zhang, H. and Hoff, K. (1997). Fast backface culling using normal masks. ACM Symposium on Interactive 3D Graphics.

Zhang, H., Manocha, D., Hudson, T., and Hoff, K. (1997). Visibility culling using hierarchical occlusion maps. Proc. of Siggraph.

Zhou, Y. and Suri, S. (1999). Analysis of a bounding box heuristic for object intersection. Journal of the ACM, 46(6): 833-857.