Chapter 6 Two-Dimensional Viewing and Clipping

Much like what we see in real life through a small window on the wall or the viewfinder of a camera, a computer-generated image often depicts a partial view of a large scene. Objects are placed into the scene by modeling transformations to a master coordinate system, commonly referred to as the world coordinate system (WCS). A rectangular window with its edges parallel to the axes of the WCS is used to select the portion of the scene for which an image is to be generated (see Fig 6-1). Sometimes an additional coordinate system called the viewing coordinate system is introduced to simulate the effect of moving and/or tilting the camera.

On the other hand, an image representing a view often becomes part of a larger image, like a photo on an album page, which models a computer monitor's display area. Since album pages vary and monitor sizes differ from one system to another, we want to introduce a device-independent tool to describe the display area. This tool is called the normalized device coordinate system (NDCS) in which a unit (1×1) square whose lower left corner is at the origin of the coordinate system defines the display area of a virtual display device. A rectangular viewport with its edges parallel to the axes of the NDCS is used to specify a sub-region of the display area that embodies the image.

The process that converts object coordinates in WCS to normalized device coordinates is called window-to-viewport mapping or normalization transformation, which is the subject of Sect. 6.1. The process that maps normalized device coordinates to discrete device/image coordinates is called workstation transformation, which is essentially a second window-to-viewport mapping, with a workstation window in the normalized device coordinate system and a workstation viewport in the device coordinate system. Collectively, these two
coordinate-mapping operations are referred to as viewing transformation.

Workstation transformation is dependent on the resolution of the display device/frame buffer. When the whole display area of the virtual device is mapped to a physical device that does not have a 1/1 aspect ratio, it may be mapped to a square sub-region (see Fig. 6-1) so as to avoid introducing unwanted geometric distortion.

Along with the convenience and flexibility of using a window to specify a localized view comes the need for clipping, since objects in the scene may be completely inside the window, completely outside the window, or partially visible through the window (e.g. the mountain-like polygon in Fig. 6-1). The clipping operation eliminates objects or portions of objects that are not visible through the window to ensure the proper construction of the corresponding image.

Note that clipping may occur in the world coordinate or viewing coordinate space, where the window is used to clip the objects; it may also occur in the normalized device coordinate space, where the viewport/workstation window is used to clip. In either case we refer to the window or the viewport/workstation window as the clipping window. We discuss point clipping, line clipping, and polygon clipping in Secs. 6.2, 6.3, and 6.4, respectively.

6.1 Window-to-viewport Mapping

A window is specified by four world coordinates: $w_{x\min}$, $w_{x\max}$, $w_{y\min}$ and $w_{y\max}$ (see Fig. 6-2). Similarly, a viewport is described by four normalized device coordinates: $v_{x\min}$, $v_{x\max}$, $v_{y\min}$ and $v_{y\max}$. The objective of window-to-viewport mapping is to convert the world coordinates $(wx, wy)$ of an arbitrary point to its corresponding normalized device coordinates $(vx, vy)$. In order to maintain the same relative placement of the point in the viewport as in the window, we require:

$$\frac{wx-w_{x\min}}{w_{x\max}-w_{x\min}} = \frac{vx-v_{x\min}}{v_{x\max}-v_{x\min}} \quad \text{and} \quad \frac{wy-w_{y\min}}{w_{y\max}-w_{y\min}} = \frac{vy-v_{y\min}}{v_{y\max}-v_{y\min}}$$

Thus

$$\begin{align*}
v_x &= \frac{v_{x\max}-v_{x\min}}{w_{x\max}-w_{x\min}} (wx-w_{x\min}) + v_{x\min} \\
v_y &= \frac{v_{y\max}-v_{y\min}}{w_{y\max}-w_{y\min}} (wy-w_{y\min}) + v_{y\min}
\end{align*}$$

Since the eight coordinate values that define the window and the viewport are just constants, we can express these two formulas for computing $(vx, vy)$ from $(wx, wy)$ in terms of a translate-scale-translate transformation $N$.

$$\begin{pmatrix} \frac{wx}{w_{x\max}} \\ \frac{wy}{w_{y\max}} \\ \frac{w}{1} \end{pmatrix} = N \cdot \begin{pmatrix} \frac{vx}{v_{x\max}} \\ \frac{vy}{v_{y\max}} \\ \frac{v}{1} \end{pmatrix}$$

where
\[
N = \begin{pmatrix}
1 & 0 & v_{x_{\text{min}}} \\
0 & 1 & v_{y_{\text{min}}} \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
\frac{v_y_{\text{max}} - v_y_{\text{min}}}{w_y_{\text{max}} - w_y_{\text{min}}} & 0 & 0 \\
0 & \frac{v_y_{\text{max}} - v_y_{\text{min}}}{w_y_{\text{max}} - w_y_{\text{min}}} & 0 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 & -w_x_{y_{\text{min}}} \\
0 & 1 & -w_y_{y_{\text{min}}}
\end{pmatrix}
\]

Note that geometric distortions occur (e.g. squares in the window become rectangles in the viewport) whenever the two scaling constants differ.

6.2 Point Clipping

Point clipping is essentially the evaluation of the following inequalities:

\[
x_{\text{min}} \leq x \leq x_{\text{max}} \quad \text{and} \quad y_{\text{min}} \leq y \leq y_{\text{max}}
\]

where \(x_{\text{min}}, x_{\text{max}}, y_{\text{min}}\) and \(y_{\text{max}}\) define the clipping window. A point \((x, y)\) is considered inside the window when the inequalities all evaluate to true.

6.3 Line Clipping

Lines that do not intersect the clipping window are either completely inside the window or completely outside the window. On the other hand, a line that intersects the clipping window is divided by the intersection point(s) into segments that are either inside or outside the window. The following algorithms provide efficient ways to decide the spatial relationship between an arbitrary line and the clipping window and to find intersection point(s).

The Cohen-Sutherland Algorithm

In this algorithm we divide the line clipping process into two phases: (1) identify those lines which intersect the clipping window and so need to be clipped and (2) perform the clipping.
All lines fall into one of the following clipping categories:

1. **Visible** – both endpoints of the line lie within the window.

2. **Not visible** – the line definitely lies outside the window. This will occur if the line from \((x_1, y_1)\) to \((x_2, y_2)\) satisfies any one of the following four inequalities:
   
   \[
   \begin{align*}
   x_1, x_2 &> x_{\text{max}} \quad y_1, y_2 > y_{\text{max}} \\
   x_1, x_2 &< x_{\text{min}} \quad y_1, y_2 < y_{\text{min}}
   \end{align*}
   \]

3. **Clipping candidate** – the line is in neither category 1 nor 2.

In Fig. 6-3, line AB is in category 1 (visible); lines CD and EF are in category 2 (not visible); and lines GH, IJ, and KL are in category 3 (clipping candidate).

![Fig. 6-3](image)

The algorithm employs an efficient procedure for finding the category of a line. It proceeds in two steps:

1. Assign a 4-bit region code to each endpoint of the line. The code is determined according to which of the following nine regions of the plane the endpoint lies in.
Starting from the leftmost bit, each bit of the code is set to true (1) or false (0) according to the scheme:

Bit 1 $\equiv$ endpoint is above the window = sign $(y - y_{\text{max}})$

Bit 2 $\equiv$ endpoint is below the window = sign $(y_{\text{min}} - y)$

Bit 3 $\equiv$ endpoint is to the right of the window = sign $(x - x_{\text{max}})$

Bit 4 $\equiv$ endpoint is to the left of the window = sign $(x_{\text{min}} - x)$

We use the convention that sign $(a) = 1$ if $a$ is positive, 0 otherwise. Of course, a point with code 0000 is inside the window.

2. The line is visible if both region codes are 0000, and not visible if the bitwise logical AND of the codes is not 0000, and a candidate for clipping if the bitwise logical AND of the region codes is 0000.

For a line in category 3 we proceed to find the intersection point of the line with one of the boundaries of the clipping window, or to be exact, with the infinite extension of one of the boundaries (see Fig. 6-4).

![Fig. 6-4](image)

We choose an endpoint of the line, say $(x_1, y_1)$, that is outside the window, i.e., whose region code is not 0000. We then select an extended boundary line by observing that those boundary lines that are candidates for intersection are the ones for which the chosen endpoint must be "pushed across" so as to change a "1" in its code to a "0" (see Fig. 6-4). This means:

- If bit 1 is 1, intersect with line $y = y_{\text{max}}$
- If bit 2 is 1, intersect with line $y = y_{\text{min}}$
- If bit 3 is 1, intersect with line $x = x_{\text{max}}$
- If bit 4 is 1, intersect with line $x = x_{\text{min}}$
Consider line CD in Fig. 6-4. If endpoint C is chosen, then the bottom boundary line $y=y_{\text{min}}$ is selected for computing intersection. On the other hand, if endpoint D is chosen, then either the top boundary line $y=y_{\text{max}}$ or the right boundary line $x=x_{\text{max}}$ is used. The coordinates of the intersection point are

\[
\begin{align*}
  x_i &= x_{\text{min}} \quad \text{or} \quad x_{\text{max}} \quad \text{if the boundary line is vertical} \\
  y_i &= y_1 + m(x_i - x_1)
\end{align*}
\]

or

\[
\begin{align*}
  x_i &= x_1 + (y_i - y_1)/m \quad \text{if the boundary line is vertical} \\
  y_i &= y_{\text{min}} \quad \text{or} \quad y_{\text{max}}
\end{align*}
\]

where $m = (y_2 - y_1)/(x_2 - x_1)$ is the slope of the line.

Now we replace endpoint $(x_1, y_1)$ with the intersection point $(x_i, y_i)$, effectively eliminating the portion of the original line that is on the outside of the selected window boundary. The new endpoint is then assigned an updated region code and the clipped line re-categorized and handled in the same way. This iterative process terminates when we finally reach a clipped line that belongs to either category 1 (visible) or category 2 (not visible).

**Midpoint Subdivision**

An alternative way to process a line in category 3 is based on binary search. The line is divided at its midpoint into two shorter line segments. The clipping categories of the two new line segments are then determined by their region codes. Each segment in category 3 is divided again into shorter segments and categorized. This bisection and categorization process continues until each line segment that spans across a window boundary (hence encompasses an intersection point) reaches a threshold for line size and all other segments are either in category 1 (visible) or in category 2 (invisible). The midpoint coordinates $(x_m, y_m)$ of a line joining $(x_1, y_1)$
and \((x_2, y_2)\) are given by

\[
\begin{align*}
    x_m &= \frac{x_1 + x_2}{2} \\
    y_m &= \frac{y_1 + y_2}{2}
\end{align*}
\]

The example in Fig. 6-5 illustrates how midpoint subdivision is used to zoom in onto the two intersection points \(I_1\) and \(I_2\) with 10 bisections. The process continues until we reach two line segments that are, say, pixel-sized, i.e., mapped to one single pixel each in the image space. If the maximum number of pixels in a line is \(M\), this method will yield a pixel-sized line segment in \(N\) subdivisions, where \(2^N = M\) or \(N = \log_2 M\). For instance, when \(M=1024\) we need at most \(N = \log_2 1024 = 10\) subdivisions.

**The Liang-Barsky Algorithm**

The following parametric equations represent a line from \((x_1, y_1)\) to \((x_2, y_2)\) along with its infinite extension:

\[
\begin{align*}
    x &= x_1 + \Delta x \cdot u \\
    y &= y_1 + \Delta y \cdot u
\end{align*}
\]

where \(\Delta x = x_2 - x_1\) and \(\Delta y = y_2 - y_1\). The line itself corresponds to \(0 \leq u \leq 1\) (see Fig. 6-6). Notice that when we traverse along the extended line with \(u\) increasing from \(-\infty\) to \(+\infty\), we first move from the outside to the inside of the clipping window's two boundary lines (bottom and left), and then move from the inside to the outside of the other two boundary lines (top and right). If we use \(u_l\) and \(u_r\), where \(u_1 \leq u_2\), to represent the beginning and end of the visible portion of the line, we have \(u_1 = \text{maximum}(1, u_f, u_b)\) and \(u_2 = \text{minimum}(1, u_f, u_t)\), where \(u_1, u_b, u_f\) and \(u_t\) correspond to the intersection point of the extended line with the window's left, bottom, top, and right boundary, respectively.
Now consider the tools we need to turn this basic idea into a simple algorithm. For point \((x, y)\) inside the clipping window, we have

\[
\begin{align*}
\min x & \leq x_i + \Delta x \cdot u \leq \max x \\
\min y & \leq y_i + \Delta y \cdot u \leq \max y
\end{align*}
\]

Rewrite the four inequalities as

\[
p_k u \leq q_k, \quad k = 1, 2, 3, 4
\]

where

\[
\begin{align*}
p_1 &= -\Delta x & q_1 &= x_i - \min x & \text{(left)} \\
p_2 &= \Delta x & q_2 &= \max x - x_i & \text{(right)} \\
p_3 &= -\Delta y & q_3 &= y_i - \min y & \text{(bottom)} \\
p_4 &= \Delta y & q_4 &= \max y - y_i & \text{(top)}
\end{align*}
\]

Observe the following facts:

- if \(p_k = 0\), the line is parallel to the corresponding boundary and

\[
\begin{cases}
\text{if } q_k < 0, & \text{the line is completely outside the boundary and can be eliminated} \\
\text{if } q_k \geq 0, & \text{the line is inside the boundary and needs further consideration},
\end{cases}
\]

- if \(p_k < 0\), the extended line proceeds from the outside to the inside of the corresponding boundary line,

- if \(p_k > 0\), the extended line proceeds from the inside to the outside of the corresponding boundary line,

- when \(p_k \neq 0\), the value of \(u\) that corresponds to the intersection point is \(q_k / p_k\).

The Liang-Barsky algorithm for finding the visible portion of the line, if any, can be stated as a four-step process:

1. If \(p_k = 0\) and \(q_k < 0\) for any \(k\), eliminate the line and stop. Otherwise proceed to the next step.

2. For all \(k\) such that \(p_k < 0\), calculate \(r_k = q_k / p_k\). Let \(u_1\) be the maximum of the set containing \(0\) and the calculated \(r\) values.

3. For all \(k\) such that \(p_k > 0\), calculate \(r_k = q_k / p_k\). Let \(u_2\) be the minimum of the set containing \(1\) and the calculated \(r\) values.

4. If \(u_1 > u_2\), eliminate the line since it is completely outside the clipping window. Otherwise, use \(u_1\) and \(u_2\) to calculate the endpoints of the clipped line.
6.4 Polygon Clipping

In this section we consider the case of using a polygonal clipping window to clip a polygon.

Convex Polygonal Clipping Windows

A polygon is called convex if the line joining any two interior points of the polygon lies completely inside the polygon (see Fig. 6-7). A non-convex polygon is said to be concave*.

By convention, a polygon with vertices \( P_1, \ldots, P_N \) (and edges \( P_{i-1}P_i \) and \( P_NP_1 \)) is said to be positively oriented if a tour of the vertices in the given order produces a counterclockwise circuit.

Equivalently, the left hand of a person standing along any directed edge \( P_{i-1}P_i \) or \( P_NP_1 \) would be pointing inside the polygon [see orientations in Figs. 6-8(a) and 6-8(b)].

Let \( A(x_1, y_1) \) and \( B(x_2, y_2) \) be the endpoints of a directed line segment. A point \( P(x, y) \) will be to the left of the line segment if the expression \( C = (x_2-x_1)(y-y_1)-(y_2-y_1)(x-x_1) \) is positive (see Prob.6.13). We say that the point is to the right of the line segment if this quantity is negative,
If a point $P$ is to the right of any one edge of a positively oriented, convex polygon, it is outside the polygon. If it is to the left of every edge of the polygon, it is inside the polygon.

This observation forms the basis for clipping any polygon, convex or concave, against a convex polygonal clipping window.

**The Sutherland-Hodgman Algorithm**

Let $P_1, P_2, \ldots, P_N$ be the vertex list of the polygon to be clipped. Let edge $E$, determined by endpoints $A$ and $B$, be any edge of the positively oriented, convex clipping polygon. We clip each edge of the polygon in turn against the edge $E$ of the clipping polygon, forming a new polygon whose vertices are determined as follows.

Consider the edge $P_{i-1}P_i$:

1. If both $P_{i-1}$ and $P_i$ are to the left of the edge, vertex $P_i$ is placed on the vertex output list of the clipped polygon [Fig. 6-9(a)].
2. If both $P_{i-1}$ and $P_i$ are to the right of the edge, nothing is placed on the vertex output list [Fig. 6-9(b)].
3. If $P_{i-1}$ is to the left and $P_i$ is to the right of the edge $E$, the intersection point $I$ of line segment $P_{i-1}P_i$ with the extended edge $E$ is calculated and placed on the vertex output list [Fig. 6-9(c)].
4. If $P_{i-1}$ is to the right and $P_i$ is to the left of edge $E$, the intersection point $I$ of the line segment $P_{i-1}P_i$ with the extended edge $E$ is calculated. Both $I$ and $P_i$ are placed on the vertex output list [Fig. 6-9(d)].

The algorithm proceeds in stages by passing each clipped polygon to the next edge of the...
window and clipping. See Probs. 6.14 and 6.15.

Special attention is necessary in using the Sutherland–Hodgman algorithm in order to avoid unwanted effects. Consider the example in Fig. 6-10(a). The correct result should consist of two disconnected parts, a square in the lower left corner of the clipping window and a triangle at the top [see Fig. 6-10(b)]. However, the algorithm produces a list of vertices (see Prob. 6.16) that forms a figure with the two parts connected by extra edges [see Fig. 6-10(c)]. The fact that these edges are drawn twice in opposite direction can be used to devise a post-processing step to eliminate them.

![Fig. 6-10](image)

**The Weller-Atherton Algorithm**

Let the clipping window he initially called the clip polygon, and the polygon to be clipped the subject polygon [see Fig. 6-11(a)]. We start with an arbitrary vertex of the subject polygon and trace around its border in the clockwise direction until an intersection with the clip polygon is encountered:

- If the edge enters the clip polygon, record the intersection point and continue to trace the subject polygon.
- If the edge leaves the clip polygon, record the intersection point and make a right turn to follow the clip polygon in the same manner (i.e., treat the clip polygon as subject polygon and the subject polygon as clip polygon and proceed as before).

Whenever our path of traversal forms a sub-polygon we output the sub-polygon as part of the overall result. We then continue to trace the rest of the original subject polygon from a recorded intersection point that marks the beginning of a not-yet-traced edge or portion of an edge. The algorithm terminates when the entire border of the original subject polygon has been traced exactly once.
For example, the numbers in Fig. 6-11 (a) indicate the order in which the edges and portions of edges are traced. We begin at the starting vertex and continue along the same edge (from 1 to 2) of the subject polygon as it enters the clip polygon. As we move along the edge that is leaving the clip polygon we make a right turn (from 4 to 5) onto the clip polygon, which is now considered the subject polygon. Following the stone logic leads to the next tight turn (from 5 to 6) onto the current clip polygon, which is really the original subject polygon. With the next step done (from 7 to 8) in the same way we have a sub-polygon for output [see Fig. 6-11(b)]. We then resume our traversal of the original subject polygon from the recorded intersection point where we first changed our course. Going from 9 to 10 to 11 produces no output. After skipping the already-traversed 6 and 7, we continue with 12 and 13 and come to an end. The figure in Fig.6-11(b) is the final result.

Applying the Weiler-Atherton algorithm to clip the polygon in Fig. 6-10(a) produces correct result [see Fig. 6-12(a) and (b)].

![Diagram](image-url)