

UNIT 3

Clipping

(7 Hours) — Computer Graphics and Animation (CACS305)

Clipping is one of the most important operations in computer graphics. Before a scene is displayed on the screen, only the portion of the scene that lies inside a specified region — called the clip window — should actually be drawn. Clipping algorithms identify which parts of a picture lie inside this region and discard (or trim) everything outside it. This unit covers the window-to-viewport transformation and the classical line and polygon clipping algorithms used to implement this idea efficiently.

3.1 Window to Viewport Transformation

A window is a specified area of interest defined in the world-coordinate system in which a picture is created. A viewport is the area on the display device (screen) where the contents of the window are actually mapped and displayed. The process of mapping a world-coordinate scene, or a part of it, to device coordinates is called the viewing transformation, sometimes also called the window-to-viewport transformation.

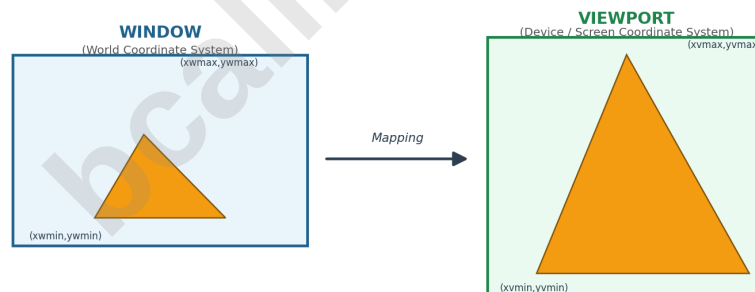


Figure 3.1 — Mapping objects from the window (world coordinates) to the viewport (device coordinates)

The window defines what is to be viewed, and the viewport defines where it is to be displayed. This two-step process allows different portions of a scene to be viewed at different positions and sizes on the output device — this is the underlying mechanism behind zooming and panning.

Coordinate Systems Involved

Coordinate System	Description
World Coordinates	The coordinate system in which the picture / scene is originally defined

Coordinate System	Description
Window Coordinates	The rectangular region of the world coordinate system selected for display
Viewport (Device) Coordinates	The rectangular region of the display device where the window is mapped
Normalized Coordinates	An intermediate coordinate system (0 to 1) sometimes used before mapping to device coordinates

Derivation of the Transformation

Let the window be defined by its lower-left and upper-right corners (x_{wmin}, y_{wmin}) and (x_{wmax}, y_{wmax}) , and let the viewport be defined by (x_{vmin}, y_{vmin}) and (x_{vmax}, y_{vmax}) . A point (x_w, y_w) inside the window must be mapped to a corresponding point (x_v, y_v) inside the viewport such that the relative position of the point within the window is preserved within the viewport.

This requires that:

- $(x_w - x_{wmin}) / (x_{wmax} - x_{wmin}) = (x_v - x_{vmin}) / (x_{vmax} - x_{vmin})$
- $(y_w - y_{wmin}) / (y_{wmax} - y_{wmin}) = (y_v - y_{vmin}) / (y_{vmax} - y_{vmin})$

Solving for x_v and y_v gives the window-to-viewport mapping equations:

Window-to-Viewport Transformation Equations

$$x_v = x_{vmin} + (x_w - x_{wmin}) \cdot S_x$$

$$y_v = y_{vmin} + (y_w - y_{wmin}) \cdot S_y$$

where the scale factors are:

$$S_x = (x_{vmax} - x_{vmin}) / (x_{wmax} - x_{wmin})$$

$$S_y = (y_{vmax} - y_{vmin}) / (y_{wmax} - y_{wmin})$$

If S_x and S_y are equal, relative proportions of the object are maintained (no distortion). If they differ, the object is stretched or compressed differently along the x and y directions.

Steps in the Viewing Transformation Pipeline

1. Construct the world-coordinate scene using modeling transformations.
2. Define a window (region of interest) in world coordinates.
3. Transform the window contents into normalized coordinates, if used.
4. Clip the scene against the window/clip boundaries.
5. Map the clipped window contents into the viewport using the above equations.

- 6. Transfer the viewport contents to the output device (workstation transformation).

Note: Clipping is usually performed either before or as part of the window-to-viewport mapping, so that only visible primitives are transformed and drawn — saving significant processing time.

3.2 Clipping, Line Clipping

Clipping is the procedure that identifies the portions of a picture that lie inside or outside a specified region, called the clip window (or clipping window). Any part of the picture outside the window is removed ('clipped off'), and only the part inside the window is retained for display. Clipping is essential because a scene may be much larger than the visible display area, and rendering objects that will never be seen wastes processing time and can also produce incorrect output on devices with a limited addressable range.

Why Clipping is Needed

- To restrict the display to the visible portion of a large scene.
- To remove objects (or parts of objects) that lie outside the viewport, preventing wasted computation and incorrect rendering.
- To implement zooming, panning, and multi-window views efficiently.
- To prepare geometric primitives for the rasterization/scan-conversion stage.

Types of Clipping

Type	Description
Point Clipping	Determines whether a point lies inside or outside the clip window
Line Clipping	Clips a line segment so only the portion inside the window remains
Polygon (Area) Clipping	Clips the boundary of a filled region/polygon against the window
Curve Clipping	Clips curves such as circles and splines using their defining equations
Text Clipping	Clips text either as whole characters, whole strings, or at the pixel level

Line Clipping

A line segment can lie completely inside the clip window, completely outside it, or partially inside and partially outside. Line-clipping algorithms must correctly classify each case:

- Trivial Accept: Both endpoints of the line lie inside the clip window — the entire line is kept as it is.

- **Trivial Reject:** Both endpoints lie outside the window on the same side (e.g. both entirely above, below, left, or right of it) — the entire line is discarded.
- **Clipping Required:** The line crosses one or more window boundaries — the intersection points with the window edges must be calculated, and only the segment lying inside the window is retained.

Naively computing the intersection of every line with every window edge using simultaneous equations for all lines is computationally expensive. Efficient algorithms therefore use quick tests to reject or accept lines trivially before resorting to full intersection calculations. The most widely used such algorithm is the Cohen–Sutherland line clipping algorithm, discussed next.

3.3 Cohen–Sutherland Line Clipping Algorithm

The Cohen–Sutherland algorithm is a fast, widely used method for clipping a line segment against a rectangular clip window. It works by dividing the 2D plane into 9 regions using the extended edges of the clip window, and assigning each region a unique 4-bit region code (also called an outcode) that indicates the position of a point relative to the window's boundaries.

Region Codes

Each of the four bits in the code corresponds to one of the four window boundaries, ordered as follows (from left to right in the code):

4-Bit Region Code Format: [Top Bottom Right Left]

Bit 1 (leftmost) — TOP: set to 1 if point is above y_{wmax}

Bit 2 — BOTTOM: set to 1 if point is below y_{wmin}

Bit 3 — RIGHT: set to 1 if point is to the right of x_{wmax}

Bit 4 (rightmost) — LEFT: set to 1 if point is to the left of x_{wmin}

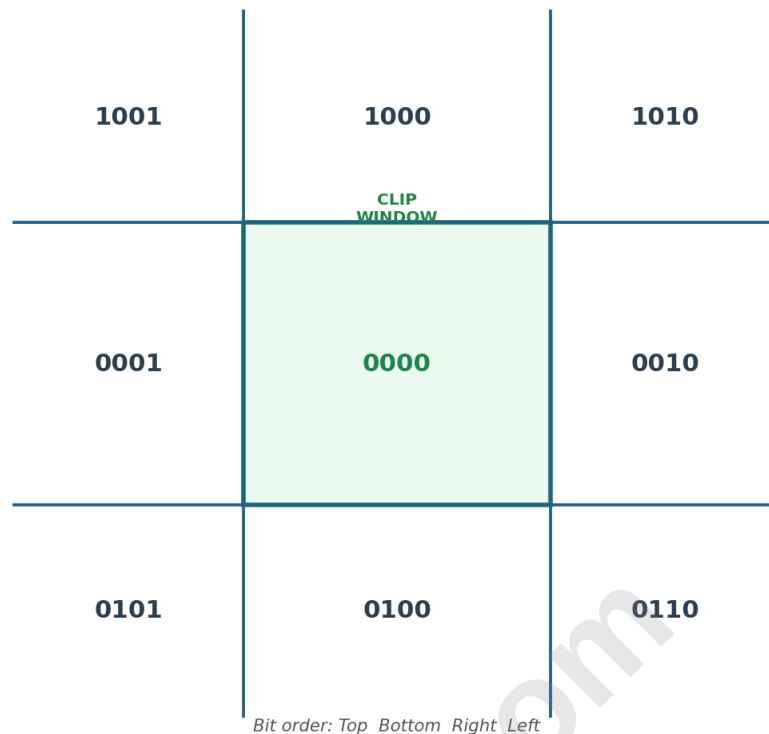


Figure 3.2 — The 9 regions formed around a clip window and their 4-bit region codes. The clip window itself has code 0000.

A region code of 0000 means the point lies inside the clip window. Any nonzero bit indicates a violation of the corresponding boundary.

Algorithm Steps

- 1. Compute the 4-bit region code for both endpoints, P1 and P2, of the line.
- 2. Trivial Accept: If both codes are 0000 (i.e., their bitwise OR is 0), the line lies completely inside — accept it as it is.
- 3. Trivial Reject: If the bitwise AND of the two codes is nonzero, both endpoints share a common outside region — the line lies completely outside the window and is rejected.
- 4. Otherwise: The line may be partially visible. Choose an endpoint with a nonzero code, and clip the line against one of the boundaries it violates (in the order top, bottom, right, left) by computing the intersection point using the line's slope.
- 5. Replace the outside endpoint with this intersection point and recompute its region code.
- 6. Repeat steps 2–5 until the line is either trivially accepted or trivially rejected.

Intersection Calculations

If the line has slope $m = (y_2 - y_1) / (x_2 - x_1)$, the intersection points with the four boundaries are computed as:

Boundary Intersection Formulas

Left/Right boundary ($x = x_{wmin}$ or x_{wmax}):

$$y = y1 + m \cdot (x - x1)$$

Top/Bottom boundary ($y = y_{wmax}$ or y_{wmin}):

$$x = x1 + (y - y1) / m$$

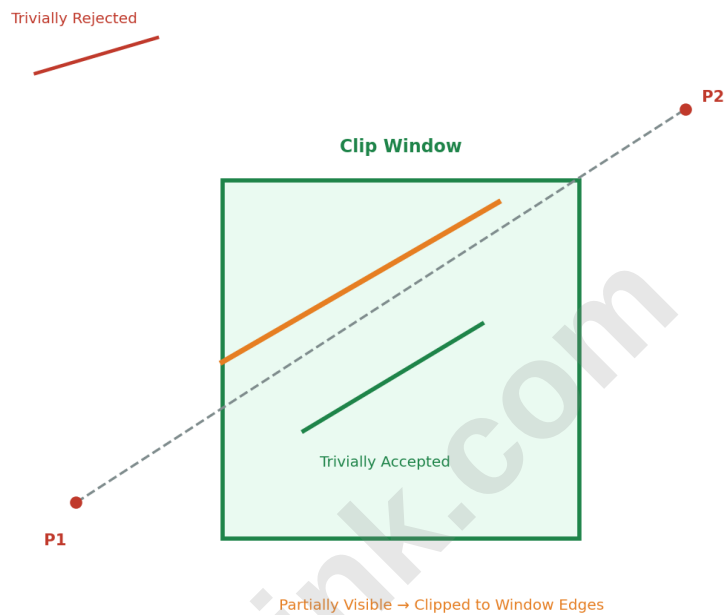


Figure 3.3 — Example outcomes of Cohen–Sutherland clipping: trivial reject, trivial accept, and a line clipped against the window boundary

Worked Example

Let the clip window have corners $(x_{\min}, y_{\min}) = (20, 20)$ and $(x_{\max}, y_{\max}) = (60, 60)$.

Let a line run from $P1(10, 30)$ to $P2(50, 80)$.

Region code of $P1$: LEFT bit set $\rightarrow$ code = 0001 ($P1$ is left of the window)

Region code of $P2$: TOP bit set $\rightarrow$ code = 1000 ($P2$ is above the window)

OR of codes = 1001 (nonzero) $\rightarrow$ not trivially accepted.

AND of codes = 0000 $\rightarrow$ not trivially rejected either $\rightarrow$ line must be clipped.

Clip $P1$ against the LEFT boundary ($x = 20$): using the line equation, find the new y at $x = 20$, giving a new intermediate point on the left edge.

Clip the resulting point against the TOP boundary ($y = 60$) to find the final visible segment endpoint.

The algorithm terminates once both endpoints of the (updated) line have region code 0000, giving the final clipped segment.

Note: Cohen–Sutherland is most efficient when most lines are either completely inside or completely outside the window, since these cases are resolved with simple bitwise operations without any intersection computation.

3.4 Polygon Clipping

Polygon clipping is more involved than line clipping because clipping the individual edges of a polygon independently, as if they were separate line segments, can produce disconnected line segments instead of a single closed polygon boundary. A correct polygon-clipping procedure must generate a new, closed polygon whose boundary consists of parts of the original polygon edges and parts of the clip-window boundary, wherever the clipped polygon boundary must follow the window edge to stay closed.

Why a Special Algorithm is Needed

- Clipping each edge of a polygon separately (as in line clipping) does not maintain the connectivity of the polygon boundary.
- The clipped region may need new edges added along the clip-window boundary to remain a closed, filled polygon.
- A dedicated polygon-clipping algorithm processes the edges of the polygon in sequence and outputs a new list of vertices that correctly describes the clipped polygon.

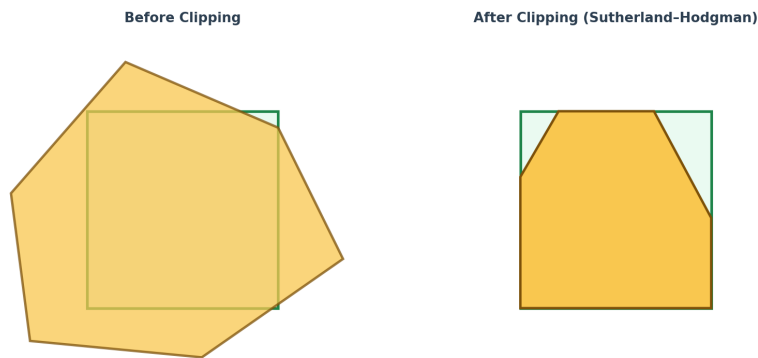


Figure 3.4 — A polygon before and after clipping against a rectangular clip window; the clipped boundary follows the window edge where needed

The most common approach for clipping convex polygons against a convex clip window (particularly a rectangle) is the Sutherland–Hodgman polygon clipping algorithm, described next.

3.5 Sutherland and Hodgman Polygon Clipping Algorithm

The Sutherland–Hodgman algorithm clips a polygon against each edge of the (convex) clip window one at a time. The polygon is successively clipped against the left, right, bottom, and top boundaries of the window (in some convenient order); the output vertex list from clipping against one boundary becomes the input to clipping against the next boundary. After the polygon has been clipped against all four boundaries, the final vertex list describes the clipped polygon.

Clipping Against a Single Boundary

For each edge of the polygon (from vertex V1 to vertex V2), the algorithm considers the edge's position relative to the current clip boundary and outputs zero, one, or two vertices according to four possible cases:

Case	Condition	Output Vertices
1	Both V1 and V2 are inside the boundary	Output V2
2	V1 is inside, V2 is outside	Output the intersection point only
3	V1 is outside, V2 is outside	Output nothing
4	V1 is outside, V2 is inside	Output the intersection point, then V2

Algorithm Steps

- 1. Start with the list of polygon vertices in order.

- 2. For each of the four clip boundaries (left, right, bottom, top), process the vertex list edge by edge and build a new output vertex list using the four cases above.
- 3. Use the output vertex list from one boundary as the input vertex list for the next boundary.
- 4. After processing all four boundaries, the resulting vertex list gives the vertices of the clipped polygon.
- 5. If the vertex list becomes empty at any stage, the polygon lies completely outside the clip window.

Worked Example (Conceptual)

Consider a hexagonal polygon that partially overlaps a rectangular clip window.

Step 1 — Clip against LEFT boundary: vertices to the left of x_{wmin} are replaced by intersection points; vertices already inside are kept unchanged.

Step 2 — Clip against RIGHT boundary: apply the same four-case test using the output of Step 1 as input.

Step 3 — Clip against BOTTOM boundary, then Step 4 — clip against TOP boundary, each time using the previous step's output list as input.

The final output vertex list traces the boundary of the clipped polygon, correctly following the clip-window edges wherever the original polygon boundary was cut off.

Note: The Sutherland–Hodgman algorithm works correctly for convex clip regions. When clipping against a non-convex clip window, additional processing is required, since a single output list may not correctly represent the clipped region.

Advantages of the Sutherland–Hodgman Approach

- Processes the polygon boundary as a whole, producing a single closed, correctly connected clipped polygon.
- Clips against multiple boundaries by simple, repeated application of the same four-case procedure — easy to implement and pipeline.
- Can be generalized to clip against any convex polygonal clipping region, not just a rectangle.

Key Takeaways

- ✓ The window-to-viewport transformation maps a selected world-coordinate region (window) to a display region (viewport) using scale factors S_x and S_y .
- ✓ Clipping restricts a picture to the portion lying inside a clip window, and may be applied to points, lines, polygons, curves, or text.
- ✓ A line may be trivially accepted (both endpoints inside), trivially rejected (both endpoints outside on the same side), or require clipping.
- ✓ The Cohen–Sutherland algorithm uses 4-bit region codes and simple bitwise tests to quickly accept/reject lines, calculating intersections only when necessary.
- ✓ Polygon clipping requires a dedicated algorithm because clipping edges independently can break the closed boundary of the polygon.
- ✓ The Sutherland–Hodgman algorithm clips a polygon against each boundary of a convex clip window in turn, using a four-case vertex test at every edge.

— End of Unit 3 —