

# Computer Graphics

P. Healy

CS1-08  
Computer Science Bldg.  
tel: 202727  
`patrick.healy@ul.ie`

Spring 2021–2022

# Outline

## 1 Clipping Algorithms; §8-5

# Announcements

- Mid-term: Week07

# Clipping: Introduction

- It is common for a region to be defined so that only graphics inside (or outside) the region are drawn  
Since vertices, lines and simple polygons (made up of straight lines) make up so much of our graphics scenes it is important that we be able to rapidly decide how much – if any – of these lines to draw
- This process is called **clipping**
- Although we will only work with 2-D **rectangular** windows clipping algorithms are not limited to this case
- We can have regions with non-linear boundaries such as circles, ellipses, spheres, as well as spline curves and spline surfaces
- Point clipping** is easiest: for a bounding box of  $(x_{bl}, y_{bl})$  to  $(x_{ur}, y_{ur})$  we only select the point  $(x, y)$  if

$$x_{bl} \leq x \leq x_{ur}$$

$$y_{bl} \leq y \leq y_{ur}$$

# Clipping: Introduction

- It is common for a region to be defined so that only graphics inside (or outside) the region are drawn
- This process is called **clipping**
- Although we will only work with 2-D **rectangular** windows clipping algorithms are not limited to this case
- We can have regions with non-linear boundaries such as circles, ellipses, spheres, as well as spline curves and spline surfaces
- **Point clipping** is easiest: for a bounding box of  $(x_{bl}, y_{bl})$  to  $(x_{ur}, y_{ur})$  we only select the point  $(x, y)$  if

$$x_{bl} \leq x \leq x_{ur}$$

$$y_{bl} \leq y \leq y_{ur}$$

# Clipping: Introduction

- It is common for a region to be defined so that only graphics inside (or outside) the region are drawn
- This process is called **clipping**
- Although we will only work with 2-D **rectangular** windows clipping algorithms are not limited to this case
- We can have regions with non-linear boundaries such as circles, ellipses, spheres, as well as spline curves and spline surfaces
- **Point clipping** is easiest: for a bounding box of  $(x_{bl}, y_{bl})$  to  $(x_{ur}, y_{ur})$  we only select the point  $(x, y)$  if

$$x_{bl} \leq x \leq x_{ur}$$

$$y_{bl} \leq y \leq y_{ur}$$

# Clipping: Introduction

- It is common for a region to be defined so that only graphics inside (or outside) the region are drawn
- This process is called **clipping**
- Although we will only work with 2-D **rectangular** windows clipping algorithms are not limited to this case
- We can have regions with non-linear boundaries such as circles, ellipses, spheres, as well as spline curves and spline surfaces
- **Point clipping** is easiest: for a bounding box of  $(x_{bl}, y_{bl})$  to  $(x_{ur}, y_{ur})$  we only select the point  $(x, y)$  if

$$x_{bl} \leq x \leq x_{ur}$$

$$y_{bl} \leq y \leq y_{ur}$$

# Clipping: Introduction

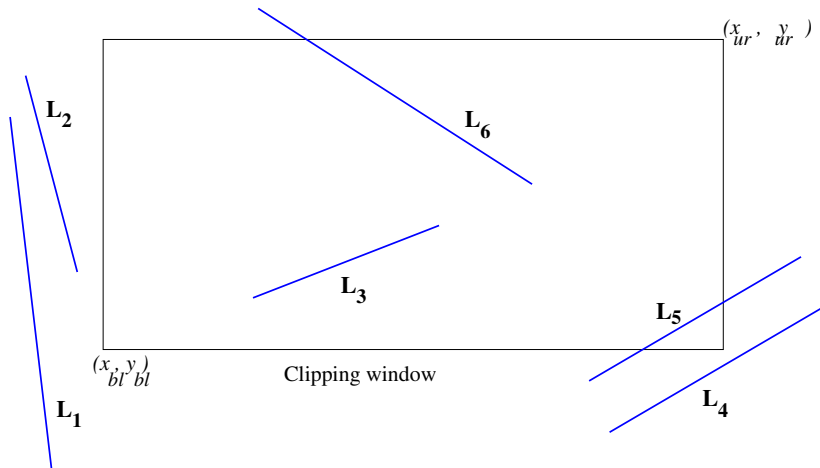
- It is common for a region to be defined so that only graphics inside (or outside) the region are drawn
- This process is called **clipping**
- Although we will only work with 2-D **rectangular** windows clipping algorithms are not limited to this case
- We can have regions with non-linear boundaries such as circles, ellipses, spheres, as well as spline curves and spline surfaces
- **Point clipping** is easiest: for a bounding box of  $(x_{bl}, y_{bl})$  to  $(x_{ur}, y_{ur})$  we only select the point  $(x, y)$  if

$$x_{bl} \leq x \leq x_{ur}$$

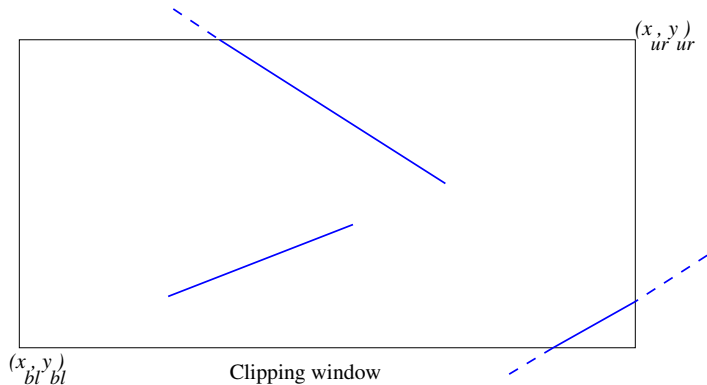
$$y_{bl} \leq y \leq y_{ur}$$






# Line Clipping; §6-7




# Line Clipping



# Line Clipping

- Line clipping algorithms are greatly sped up by lines either being entirely  the clipping rectangle or entirely 
- So lines  $L_1$ ,  $L_2$  or  $L_3$  previously will be easier for almost all clipping algorithms than 
- Contrast this with **culling**, the process of eliminating from consideration those objects that are entirely outside the window; done prior to clipping
- Lines  $L_5$  and  $L_4$  will prove hardest and most time consuming to clip

# An Inefficient Line Clipper

- Use  of a line between start  $p = (x_s, y_s)$  and end  $q = (x_e, y_e)$

$$x = x_s + u(x_e - x_s)$$

$$y = y_s + u(y_e - y_s), \quad 0 \leq u \leq 1 \quad (\text{or, } u \in [0, 1])$$


- What does it mean if  $u \notin [0, 1]$ ?
- We can find where this line crosses the west side of the rectangle ( $x = x_{bl}$ ) by solving for  $u$  in

$$x_{bl} = x_s + u(x_e - x_s)$$

$$u = (x_{bl} - x_s) / (x_e - x_s)$$


- The line can **cross** western border only if  $u \in [0, 1]$
- If  $u \notin [0, 1]$  we can test one end point against  $x_{bl}$  e.g.  $L_2$
- If  $u \in [0, 1]$  we can find value of  $y = y'$  at  $x = x_{bl}$  and we **clip** off portion outside
- Repeat for other three sides

# Cohen-Sutherland Line Clipper

- To each endpoint assign a **region code** or  that codes which of the 8 regions with respect to clip region that point is
- Code is a 4-bit value  $c = b_4b_3b_2b_1$ :
  - if  $x < x_{bl}$  (left outside) then set  $b_1$
  - if  $x > x_{ur}$  (right outside) then set  $b_2$
  - if  $y < y_{bl}$  (below outside) then set  $b_3$
  - if  $y > y_{ur}$  (above outside) then set  $b_4$
- A point is in clipping region if and only if code  $c = 0000$
- These can be set in a (marginally) more efficiently way by, for example, setting  $b_2$  to sign bit of  $x_{ur} - x$
- Not all 16 ( $=2^4$ ) cases can occur: a point can't be both left and right of window, but **can** be neither (if  $x_{bl} \leq x \leq x_{ur}$ )


# Cohen-Sutherland Line Clipper

What do we do with this information?

- Look at codes  $c_s$  and  $c_e$  for start and end points of line using bitwise operators, `or` and `and`
- Any lines  in clipping region have code 0000; using bitwise-or:  $c_s | c_e == 0$
- Any line with 1 in same position must be **outside**; using bitwise-and:  $c_s \& c_e > 0$
- For remaining cases we need to consider each of four borders in turn, compute intersection points with them, clip off the part outside and re-test if line is entirely in one region
-


# Cohen-Sutherland Line Clipper

What do we do with this information?

- Look at codes  $c_s$  and  $c_e$  for start and end points of line using bitwise operators, `or` and `and`
- Any lines  in clipping region have code 0000; using bitwise-or:  $c_s | c_e == 0$
- Any line with 1 in same position must be **outside**; using bitwise-and:  $c_s \& c_e > 0$
- For remaining cases we need to consider each of four borders in turn, compute intersection points with them, clip off the part outside and re-test if line is entirely in one region
-

# Cohen-Sutherland Line Clipper

What do we do with this information?


- Look at codes  $c_s$  and  $c_e$  for start and end points of line using bitwise operators, `or` and `and`
- Any lines  in clipping region have code 0000; using bitwise-or:  $c_s | c_e == 0$
- Any line with 1 in same position must be **outside**; using bitwise-and:  $c_s \& c_e > 0$
- For remaining cases we need to consider each of four borders in turn, compute intersection points with them, clip off the part outside and re-test if line is entirely in one region





# Cohen-Sutherland Line Clipper


What do we do with this information?

- Look at codes  $c_s$  and  $c_e$  for start and end points of line using bitwise operators, `or` and `and`
- Any lines  in clipping region have code 0000; using bitwise-or:  $c_s | c_e == 0$
- Any line with 1 in same position must be **outside**; using bitwise-and:  $c_s \& c_e > 0$
- For remaining cases we need to consider each of four borders in turn, compute intersection points with them, clip off the part outside and re-test if line is entirely in one region



# Cohen-Sutherland Line Clipper


What do we do with this information?

- Look at codes  $c_s$  and  $c_e$  for start and end points of line using bitwise operators, `or` and `and`
- Any lines  in clipping region have code 0000; using bitwise-or:  $c_s | c_e == 0$
- Any line with 1 in same position must be **outside**; using bitwise-and:  $c_s \& c_e > 0$
- For remaining cases we need to consider each of four borders in turn, compute intersection points with them, clip off the part outside and re-test if line is entirely in one region
- To compute  $y$  corresponding to  $x = x_{bl}$  or  $x = x_{ur}$

$$y = y_s + m(x - x_s)$$

# Cohen-Sutherland Line Clipper

What do we do with this information?

- Look at codes  $c_s$  and  $c_e$  for start and end points of line using bitwise operators, `or` and `and`
- Any lines  in clipping region have code 0000; using bitwise-or:  $c_s | c_e == 0$
- Any line with 1 in same position must be **outside**; using bitwise-and:  $c_s \& c_e > 0$
- For remaining cases we need to consider each of four borders in turn, compute intersection points with them, clip off the part outside and re-test if line is entirely in one region
- To compute  $x$  corresponding to  $y = y_{bl}$  or  $y = y_{ur}$

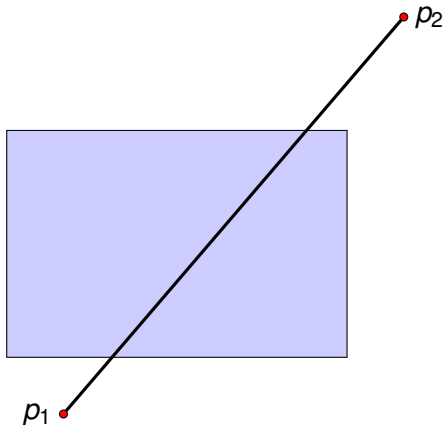
$$x = x_s + \frac{1}{m}(y - y_s)$$

# Cohen-Sutherland Line Clipper

- Assume borders are processed in order left, right, bottom, top
- No points west of left border
- Point  $p_2$  is to east of right border so we clip it, but **only** to  $p'_2$
- Point  $p_1$  is south of bottom border so we clip it **at**  $p'_1$
- Point  $p'_2$  now gets considered and we get, finally...

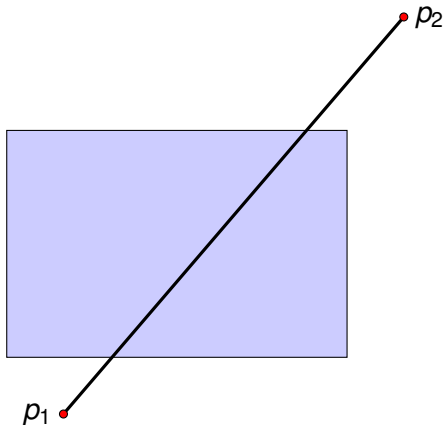
# Cohen-Sutherland Line Clipper

- Assume borders are processed in order left, right, bottom, top
- No points west of left border
- Point  $p_2$  is to east of right border so we clip it, but **only** to  $p'_2$
- Point  $p_1$  is south of bottom border so we clip it **at**  $p'_1$
- Point  $p'_2$  now gets considered and we get, finally...



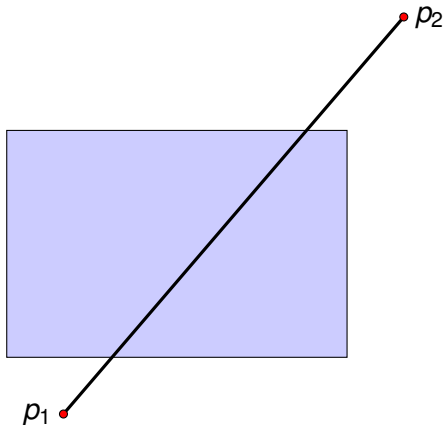
# Cohen-Sutherland Line Clipper

- Assume borders are processed in order left, right, bottom, top
- No points west of left border
- Point  $p_2$  is to east of right border so we clip it, but **only** to  $p'_2$
- Point  $p_1$  is south of bottom border so we clip it **at**  $p'_1$
- Point  $p'_2$  now gets considered and we get, finally...



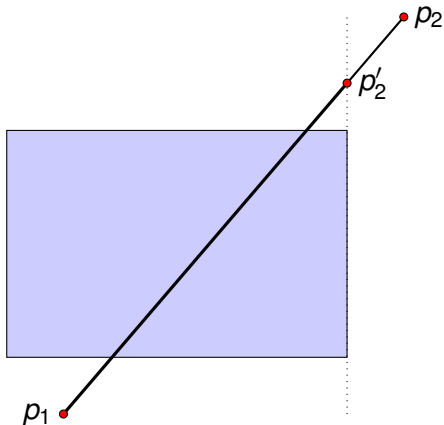
# Cohen-Sutherland Line Clipper

- Assume borders are processed in order left, right, bottom, top
- No points west of left border
- Point  $p_2$  is to east of right border so we clip it, but **only** to  $p'_2$
- Point  $p_1$  is south of bottom border so we clip it at  $p'_1$
- Point  $p'_2$  now gets considered and we get, finally...



# Cohen-Sutherland Line Clipper

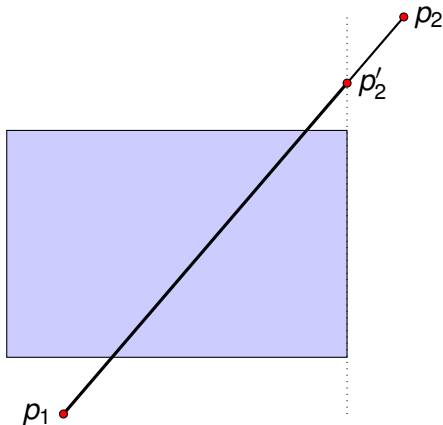
- Assume borders are processed in order left, right, bottom, top
- No points west of left border
- Point  $p_2$  is to east of right border so we clip it, but **only** to  $p'_2$
- Point  $p_1$  is south of bottom border so we clip it at  $p'_1$
- Point  $p'_2$  now gets considered and we get, finally...





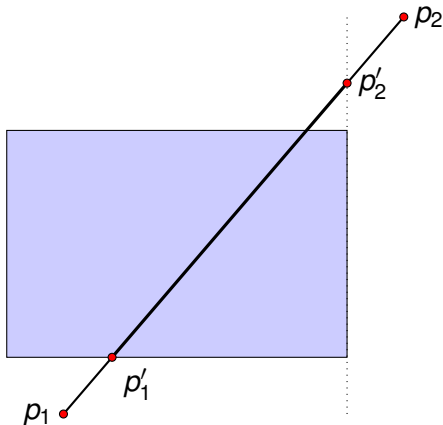
# Cohen-Sutherland Line Clipper

- Assume borders are processed in order left, right, bottom, top
- No points west of left border
- Point  $p_2$  is to east of right border so we clip it, but **only** to  $p'_2$
- Point  $p_1$  is south of bottom border so we clip it at  $p'_1$
- Point  $p'_2$  now gets considered and we get, finally...



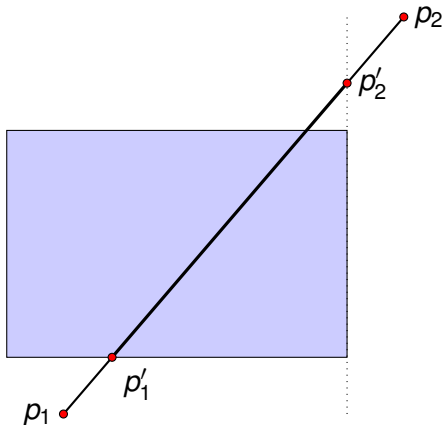
# Cohen-Sutherland Line Clipper

- Assume borders are processed in order left, right, bottom, top
- No points west of left border
- Point  $p_2$  is to east of right border so we clip it, but **only** to  $p'_2$
- Point  $p_1$  is south of bottom border so we clip it at  $p'_1$
- Point  $p'_2$  now gets considered and we get, finally...



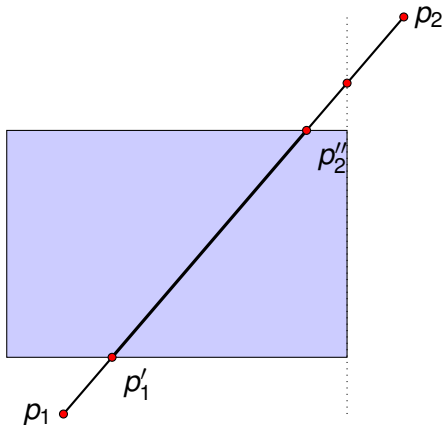
# Cohen-Sutherland Line Clipper

- Assume borders are processed in order left, right, bottom, top
- No points west of left border
- Point  $p_2$  is to east of right border so we clip it, but **only** to  $p'_2$
- Point  $p_1$  is south of bottom border so we clip it at  $p'_1$
- Point  $p'_2$  now gets considered and we get, finally...



# Cohen-Sutherland Line Clipper

- Assume borders are processed in order left, right, bottom, top
- No points west of left border
- Point  $p_2$  is to east of right border so we clip it, but **only** to  $p'_2$
- Point  $p_1$  is south of bottom border so we clip it at  $p'_1$
- Point  $p'_2$  now gets considered and we get, finally...



# Analysis of Cohen-Sutherland Alg.

In general, the Cohen-Sutherland algorithm is not optimal

- In picture to right, if clipping order is north, west, east, south then much wasted work in the form of **multiple shortenings** of the line is performed
- Also, each C-S shortening requires a floating point division or multiplication (to solve line intersection)

