

## ASSIGNMENT VI

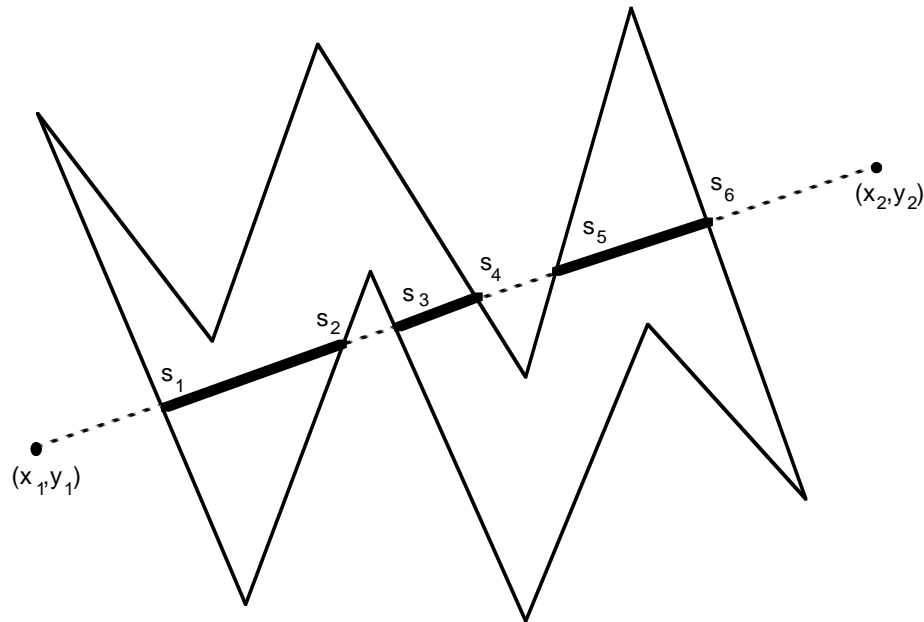
### Clipping a segment by a closed polygonal curve

The next step towards the construction of an algorithm for displaying projections of composite objects is to be able to recognize what portions of a given segment are visible and what portions are invisible due to the presence of objects that partially hide the segment from an observer's view.

This given, in this assignment you are to write a MATHEMATICA procedure with heading

**CLIP**[*seg*\_, *curve*\_]

which displays a user chosen *closed polygonal* “*curve*” and the portions of a *segment* that lie inside the region bounded by the curve. The example below illustrates the type of display you want to get.



The input variable **seg** should be a list

$$\mathbf{seg} = \{\{x_1, y_1\}, \{x_2, y_2\}\} \quad (1)$$

giving the end points of the segment to be *clipped* and **curve** should likewise be a list giving the coordinates of the successive vertices of the polygon that does the clipping.

We assume that you have carried out the fifth assignment and that you are already in possession of the procedure **inside**, which upon the call

**inside**[**P**, *curve*]

returns **True** if the point **P** is inside the region bounded by **curve** and returns **False** if **P** is outside.

This given you should start by writing a MATHEMATICA function with heading

**cut**[*seg*\_, *cseg*\_]

where the input variable **seg** is the segment to be *cut* and **cseg** will be one of the segments of the polygon that does the *cutting*. Basically, the task of this function is to construct the point of intersection (if any) of the two input segments **seg** and **cseg**. However, it is essential (as we shall soon see) that **cut** returns only the *barycentric coordinate* of this point in **seg**.

To give a precise description of what **cut** is to compute let us say that **seg** is given as in (1) and that

$$\mathbf{cseg} = \{ \{ \alpha_1, \beta_1 \}, \{ \alpha_2, \beta_2 \} \} \quad (2)$$

then the parametric equations of **seg** are  $\mathbf{P}[s] = (x[s], y[s])$  with

$$\begin{aligned} x[s] &= x_1 + s(x_2 - x_1) \\ y[s] &= y_1 + s(y_2 - y_1) \end{aligned} \quad (0 \leq s \leq 1) \quad (3)$$

The parameter  $s$  is usually referred to as a *barycentric coordinate* since the point  $\mathbf{P}[s]$  is the center of gravity of two masses of sizes  $(1 - s)$  and  $s$  placed at  $(x_1, y_1)$  and  $(x_2, y_2)$  respectively. Analogously, the parametric equations of **cseg** may be written as  $\mathbf{Q}[t] = (\alpha[t], \beta[t])$  with

$$\begin{aligned} \alpha[t] &= \alpha_1 + t(\alpha_2 - \alpha_1) \\ \beta[t] &= \beta_1 + t(\beta_2 - \beta_1) \end{aligned} \quad (0 \leq t \leq 1) \quad (4)$$

Note that if  $s$  and  $t$  in (3) and (4) are allowed to vary in  $(-\infty, \infty)$  then the points  $\mathbf{P}[s]$  and  $\mathbf{Q}[t]$  describe the entire lines supporting the segments. The point of intersection of these two lines is obtained by equating  $\mathbf{P}[s]$  to  $\mathbf{Q}[t]$  and solving for  $s$  and  $t$ . This gives the system of equations

$$\begin{aligned} x_1 + s(x_2 - x_1) &= \alpha_1 + t(\alpha_2 - \alpha_1) \\ y_1 + s(y_2 - y_1) &= \beta_1 + t(\beta_2 - \beta_1) \end{aligned} ,$$

which are better rewritten in the form

$$\begin{aligned} s(x_2 - x_1) + t(\alpha_1 - \alpha_2) &= \alpha_1 - x_1 \\ s(y_2 - y_1) + t(\beta_1 - \beta_2) &= \beta_1 - y_1 \end{aligned} .$$

We can thus use Cramer's rule and obtain

$$s = \frac{\det \begin{bmatrix} \alpha_1 - x_1 & \alpha_1 - \alpha_2 \\ \beta_1 - y_1 & \beta_1 - \beta_2 \end{bmatrix}}{\Delta} , \quad t = \frac{\det \begin{bmatrix} x_2 - x_1 & \alpha_1 - x_1 \\ y_2 - y_1 & \beta_1 - y_1 \end{bmatrix}}{\Delta} \quad (5)$$

where

$$\Delta = \det \begin{bmatrix} x_2 - x_1 & \alpha_1 - \alpha_2 \\ y_2 - y_1 & \beta_1 - \beta_2 \end{bmatrix} .$$

It is not difficult to see that **seg** and **cseg** intersect at a single point if and only if

$$\begin{cases} 1) & \Delta \neq 0 \\ 2) & 0 \leq s \leq 1 \\ 3) & 0 \leq t \leq 1 \end{cases} . \quad (6)$$

This given your function **cut** must return **s** if all the conditions in (6) are satisfied and return a large number say 100 if at least one of them fails.

The reason we require such an output from **cut** is that we are going to make several calls of **cut** by letting **cseg** run through all the edges of **curve**, and we want to keep record of the  $s$  parameter **only** for the *significant* cut points. Now if  $\Delta = 0$  the two supporting lines are either parallel and distinct or identical. In the first case **seg** and **cseg** clearly cannot intersect and we should indicate that in some manner. In the second case there is still the possibility that **seg** and **cseg** may intersect in a segment. But then, the two edges of **curve** that are adjacent to **cseg** will cut **seg** at the end points of this latter segment, and we need not bother processing **cseg** any further and act as if **seg** and **cseg** did not intersect.

Once you have programmed **cut** the procedure **clip** can be written as follows. Using the **Table** command you produce first all the barycentric coordinates of the points of intersection of **seg** with the successive edges of **curve**. This will be a list of numbers between 0 and 1 intermixed with a few 100's. Note that even if we throw away the 100's the remaining list of  $s$  values does not give the intersection points in the order they are encountered as we travel from  $(x_1, y_1)$  to  $(x_2, y_2)$  in **seg**. Why this can happen can be easily seen from the example illustrated above. But this is no major problem. You simply use the **MATHEMATICA Sort** command to sort the  $s$ -values in increasing order. Let us assume then that the sorted list is stored as **sl**. It will be convenient to prepend to it the  $s$  value 0 and append the value 1 if the sorted list does not begin with 0 or end with 1. This done you can have **cut** enter a **Do**  $i$ -loop where at the  $i^{th}$  stage, it tests whether the segment

$$\mathbf{P}[\mathbf{sl}[[i]]] \rightarrow \mathbf{P}[\mathbf{sl}[[i + 1]]] \quad (7)$$

is or is not inside the polygonal curve. This can be achieved by a call of the function **inside** of the previous assignment, with **P** the midpoint of this segment. Whenever **inside** returns **True** and only then you concatenate the segment in (7) into your growing list of segments that are to be displayed.

This done you terminate **clip** with the graphics commands that cause the display of the clipping curve and the resulting clipped segment.

Note that **clip** may be used to construct the visible portion of an object placed behind a window bounded by **curve**. You may add some fun to your task if you use your program to display your favorite *polygonal object* peering through your favorite window as illustrated in the figure below

