9 CHAPTER Nine

WINDOWS AND CLIPPING

Chapter Outline

- Introduction
- Window-to View Port Co-Ordinate Transformation
- Clipping
- Line Clipping
- Curve Clipping
- Interior and Exterior Clipping
- Multiple Windowing
- The Viewing TransformatioN
- Point Clipping
- Polygon Clipping
- Text Clipping
- Generalized Clipping

9.1. INTRODUCTION

When drawing are too complex, they become difficult to read. In such situations, it is useful to display only those portions of the drawing that are of immediate interest. This gives the effect of looking at the image through a window. Furthermore it is desirable to enlarge these portions to take full advantage of the display surface. The method for selecting, and enlarging portions of a drawing is called **Windowing.** The technique for not showing that part of the drawing which one is not interested in is called '**clipping**'.

9.2. THE VIEWING TRANSFORMATION

Displaying an image of a picture involves mapping the co-ordinates of the points and lines that form the picture into the appropriate co-ordinates on the device or workstation where the image is to be displayed. This is done through the use of co-ordinate transformations known as *viewing transformations.* To perform a viewing transformation, we deal with a finite region in the WCS (World Co-ordinate System) called a *Window.* The window can be mapped directly on the display area of the device or on to a subregion of the display called a **'viewport'.** Thus, a world-coordinate area selected for display is called a window and an area on a display device to which a window is mapped is called a viewport, *i.e.* the window defines what is to be viewed the viewport defines where it is to be displayed.

If we are changing the position of window by keeping the viewpart location constant, then the different part of the object is displayed at the same position on the display device. Similarly if we change the location of viewport then we will see the same part of the object drawn at different places on screen.

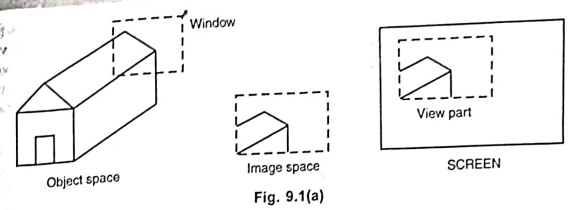


Fig. 9.1(a)

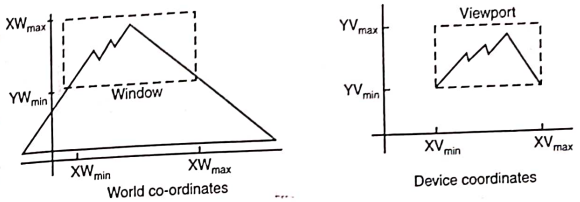
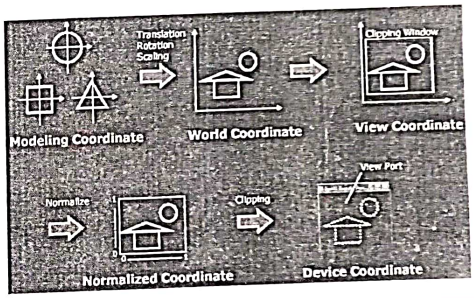
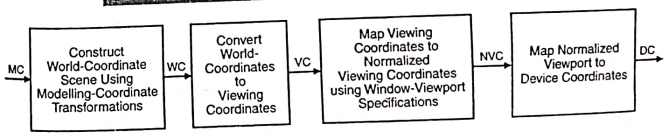


Fig. 9.1(b)

Viewing Pipeline

Some graphics packages that provide window and viewport operations allow only standard rectangles, but a more general approach is to allow the rectangular window to have any orientation. In this case, we carry out the viewing transformation in several steps, as indicated below.



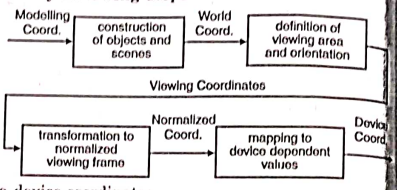


First, we construct the scene in world coordinates using the output primitives. Next, to obtain a particular orientation for the window, we can set up a two-dimensional viewing coordinate system in the world-coordinate plane, and define a window in the viewing-coordinate system. The viewing coordinate reference frame is used to provide a method for setting up arbitrary orientations for rectangular windows. Once the viewing reference frame is established, we can transform descriptions in world coordinates to viewing coordinates.

We then define a viewport in normalized coordinates (in the range from 0 to 1) and map the viewing-coordinate description of the scene to normalized coordinates. At the final step all parts of the picture that outside the viewport are clipped, and the contents of the viewport are transferred to device coordinates.

2D viewing pipeline can be achieved by following steps

- Construct world-coordinate scene using modeling-coordinate transformations.
- Convert world-coordinates to viewing coordinates.
- Transform viewing-coordinates to normalized-coordinates (ex: between 0 and 1, or between – 1 and 1).
- Map normalized-coordinates to device-coordinates.



9.3. WINDOW-TO VIEW PORT CO-ORDINATE TRANSFORMATION *(UPTU 2008)*

In general the mapping of a part of world co-ordinate scene to device co-ordinates is referred as *viewing transformation*. Sometimes it is also called *Window-viewport transformation* or *windowing transformation*.

The viewing transformation is a process of 3 steps.

Step 1: The object together with its window is translated until the lower-left corner of the window is at the origin.

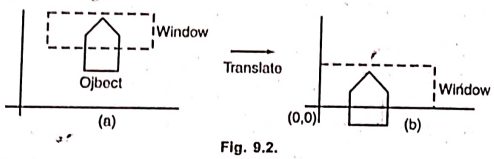


Fig. 9.2.

Step 2: The object and the window are now scaled until the window has the dimension same as view port. In other words we are converting the object into image and window in view port.

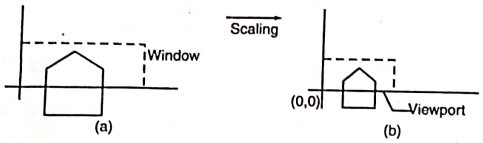


Fig. 9.3.

Step 3: The final transformation step is another translation to move the viewport to its correct position on the screen.

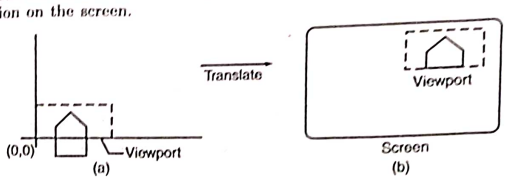


Fig. 9.4.

So, the viewing transformation performs three steps:

1. Translation
2. Scaling
3. Translation.

EXAMPLE 9.1

Suppose there is a rectangle ABCD whose co-ordinates are A(1, 1), B(4, 1), C(4, 4), D(1, 4) and the window co-ordinates are (2, 2), (5, 2), (5, 5), (2, 5) and the given viewport location is (0.5, 0), (1, 0), (1, 0.5), (0.5, 0.5). Calculate the viewing transformation matrix.

Solution: First we draw the rectangle *ABCD* and window

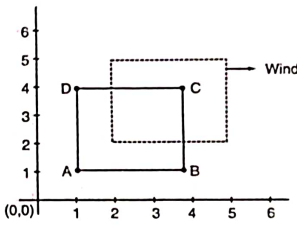


Fig. 9.5.

For viewing transformation, first step is translation. We have to shift left lower corner of window to origin, *i.e.*

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & -2 & 1 \end{vmatrix}$$

Now the second step is scaling. Here we have to scale window to view port *i.e.* old dimensions will be of windows dimensions and new will be of viewport.

Scaling in x-direction *i.e.*

$$S_x = \frac{\text{width of viewport}}{\text{width of window}} = \frac{1-0.5}{5-2} = \frac{0.5}{3} = 0.16$$

Similarly $$S_y = \frac{\text{heightof viewport}}{\text{height of window}} = \frac{1-0.5}{5-2} = \frac{0.5}{3} = 0.16$$

Thus the scaling matrix will be $\begin{vmatrix} 0.16 & 0 & 0 \\ 0 & 0.16 & 0 \\ 0 & 0 & 1 \end{vmatrix}$

In third step, we have to translate the viewport to the desired location on the screen. So translation matrix will be

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0 & 1 \end{vmatrix}$$

Hence viewing transformation matrix will be

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & -2 & 1 \end{vmatrix} \times \begin{vmatrix} 0.16 & 0 & 0 \\ 0 & 0.16 & 0 \\ 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 0.16 & 0 & 0 \\ 0 & 0.16 & 0 \\ 0.18 & -0.32 & 1 \end{vmatrix}$$

In general, the viewing transformation is

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -W_{XL} & -W_{YL} & 1 \end{vmatrix} = \begin{vmatrix} \frac{(V_{XH}-V_{XL})}{(W_{XH}-W_{XL})} & 0 & 0 \\ 0 & \frac{(V_{YH}-V_{YL})}{(W_{YH}-W_{YL})} & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ V_{XL} & V_{YL} & 1 \end{vmatrix}$$

The variables have been named according to the rule that V stands for viewport W for windows. X for the position of a vertical boundary and Y for a horizontal boundary. H for the high boundary and L for the low boundary, *i.e.*

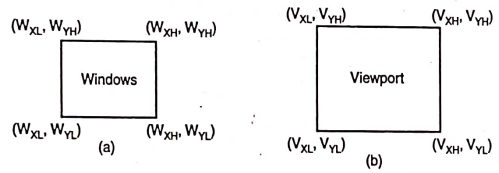


Fig. 9.6.

After multiplication the generalized viewing transformation matrix will be

$$\begin{vmatrix} \frac{V_{XH}-V_{XL}}{W_{XH}-W_{XL}} & 0 & 0 \\ 0 & \frac{V_{YH}-V_{YL}}{W_{YH}-W_{YL}} & 0 \\ (V_{XL}-W_{XL}).\frac{(V_{XH}-V_{XL})}{(W_{XH}-W_{XL})} & (V_{YL}-W_{YL}).\frac{V_{YH}-V_{YL}}{W_{YH}-W_{YL}} & 1 \end{vmatrix}$$

If the height and width of the window do not have the same properties as the height and width of the veiwport, then the viewing transformation will cause some distortion of the image.

9.4. CLIPPING

Generally, any procedure, that identifies those portions of a picture that are either inside or outside of a specified region of space is referred to as a clipping algorithm or simply **Clipping**. The region against which an object is to clipped is called a **clip window.**

Applications of clipping include extracting part of a defined scene for viewing, identifying visible surfaces in three-dimensional views; antialiasing line segments or object boundaries; creating objects using solid-modelling procedures, displaying a multiwindow environment, and drawing and painting operations that allow parts of a picture to be selected for copying, moving, erasing or duplicating. Depending on the application, the clip window can be a general polygon or it can even have curved boundaries. Clipping algorithms can be applied in world coordinats, so that only the contents of the window interior are mapped to device coordinates. Alternately the complete world coordinate picture can be mapped first to device coordinates or normalized device coordinates, then clipped against the viewport boundaries. World-coordinate clipping removes those primitives outside the window from further consideration, thus eliminating the processing necessary to transform those primitives to device space. Viewport clipping, on the other hand, can reduce calculations by allowing concatenation of viewing and geometric transformation matrices. But viewport clipping does require that the transformation to device coordinates be performed for all objects, including those outside the window area.

In the following sections, we consider algorithms for clipping the following primitive types:

1. Point clipping
2. Line clipping
3. Polygon clipping (Area)
4. Curve clipping
5. Text clipping.

9.5. POINT CLIPPING

Assuming that the clip window is a rectangle in standard position, we save a point $P(x, y)$ for display if the following inequalities are satisfied.

$$x\omega_{min} \le x \le x\omega_{max}$$
$$y\omega_{min} \le y \le y\omega_{max}$$

where the edges of the clip window ($x\omega_{min}$, $x\omega_{max}$, $y\omega_{min}$, $y\omega_{max}$) can be either the world-coordinate window boundaries or viewport boundaries.

If any one of these four inequalities is not satisfied, the point is clipped *i.e.*, not saved for display.

9.6. LINE CLIPPING

Lines that do not intersect the clipping window are either completely inside the window or completely outside the window. All line segments fall into one of the following clipping categories:

1. **Visible:** Both end points of the line segment lie within the window.
2. **Non-visible:** When line definitely lies outside the window. This will occur if the line segment from (x_1, y_1) to (x_2, y_2) satisfies any one of the following four inequalities:

$$\boxed{\begin{array}{ll} x_1, x_2 > x_{max} & y_1, y_2 > y_{max} \\ x_1, x_2 < x_{min} & y_1, y_2 < y_{min} \end{array}}$$

3. **Partially visible (or clipping condidate):** A line is partially visible when a part of its lies within the window.

Line clipping algorithms include the *Cohen-Sutherland method*, the *Liang-Barsky method*, and the *Nicholl-Lee-Nicholl method*. The Cohen-Sutherland method is widely used. In this region codes are used to identify the position of line endpoints relative to the rectangular,

clipping window boundaries. Lines that cannot be immediately identified as completely inside the window or completely outside are then clipped against window boundaries. *Liang and Barsky* use a parametric line representation to set up a more efficient line-clipping procedure that reduces intersection calculations. The *Nicholl-Lee-Nicholl method* uses more region testing in the *xy* plane to reduce intersection calculations even further.

The basic principles (assuming the clip window *W* is convex) are:

- If both endpoints of a line segment fall within *W*, then display the line segment.
- If both endpoints of a line segment fall outside of *W* because they violate the same point clipping inequality, then do not display the line segment.
- If one endpoint falls within *W* and another falls outside, then part of the line segment is displayed.
- If both endpoints fall outside *W*, but do not violate a common point clipping inequality, then part of the line may be visible.

9.6.1. Cohen-Sutherland Algorithm

In computer graphics, the **Cohen-Sutherland** algorithm is a line clipping algorithm. The algorithm divides a 2D space into 9 parts, using the infinite extensions of the four linear boundaries of the window. Assign a bit pattern to each region as shown below:

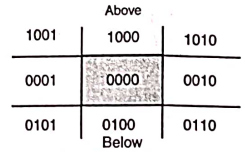


Fig. 9.7.

The numbers in the figure above are called **outcodes.** An outcode is computed for each of the two points in the line. The bits in the outcode represent: **Top, Bottom, Right, Left.** Each bit in the code is set to either a 1 (true) or a 0 (false). If the region is to the *left* of the window, the ***first*** bit of the code is set to 1. If the region is to the *Right* of the window, the *second* bit of the code is set to 1. If to the ***Bottom,*** the ***third*** bit is set, and if to the ***Top,*** the ***fourth*** bit is set. The 4 bits in the code then identify each of the nine regions.

For any endpoint (x, y) of a line, the code can be determined that identifies which region the endpoint lies. The code's bits are set according to the following conditions:

- First bit set 1: Point lies to **left** of window $x < x_{min}$
- Second bit set 1: Point lies to **right** of window $x > x_{max}$
- Third bit set 1: Point lies **below** (bottom) window $y < y_{min}$
- Fourth bits set 1: Point lies **above** (top) window $y > y_{max}$

For example,

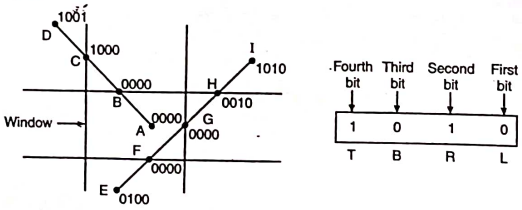


Fig. 9.8.

The outcode 1010 represents a point that is top-right of the viewport. Note that the outcodes for endpoints **must** be recalculated on each iteration after the clipping occurs.

The Algorithm

The Cohen-Sutherland algorithm uses a divide-and-conquer strategy. The line segment's endpoints are tested to see if the line can be trivially accepted or rejected. If the line cannot be trivially accepted or rejected, an intersection of the line with a window edge is determined and the trivial reject/accept test is repeated. This process is continued until the line is accepted.

To perform the trivial acceptance and rejection tests, we extend the edges of the window to divide the plane of the window into the nine regions. Each endpoint of the line segment is then assigned the code of the region in which it lies.

1. Given a line segment with endpoint $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$
2. Compute the 4-bit codes for each endpoint.
   If both codes are **0000,** (bitwise OR of the codes yields 0000) line lies completely **inside** the window: pass the endpoints to the draw routine.
   If both codes have a 1 in the same bit position (bitwise AND of the codes is **not** 0000), the line lies **outside** the window. It can be trivially rejected.
3. If a line cannot be trivially accepted or rejected, at least one of the two endpoints must lie outside the window and the line segment crosses a window edge. This line must be **clipped** at the window edge before being passed to the drawing routine.
4. Examine one of the endpoints, say $P_1 = (x_1, y_1)$. Read P_1's 4-bit code in order: **Left-to-Right, Bottom-to-Top.**
5. When a set bit (1) is found, compute the **intersection I** of the corresponding window edge with the line from P_1 to P_2. Replace P_1 with I and repeat the algorithm.

Note: If both endpoints of a line entirely to one side of the window the line must lie entirely outside of the window. It is trivally rejected and if both end points of a line lie4 inside the window, the entire lie lies inside the window. It is trivially accepted.

Before Clipping

1. Consider the line segment **AD.**
   Point **A** has an outcode of **0000** and point **D** has an outcode of **1001.** The logical AND of these outcodes is zero; therefore, the line cannot be trivially rejected. Also, the logical OR of the outcodes is no zero; therefore, the line cannot be trivially accepted. The algorithm then chooses **D** as the outside point (its outcode contiain 1's).

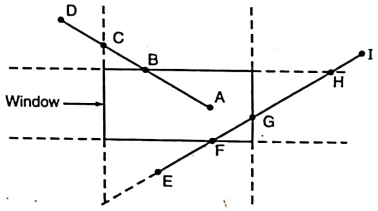


Fig. 9.9.

   By our testing order, we first use the top edge to clip **AD** at **B.** The algorithm then recomputes **B's** outcode as **0000.** With the next iteration of the algorithm, **AB** is tested and is trivially accepted and displayed.
2. Consider the line segment **EI**
   Point **E** has an outcode of **0100,** while point **I's** outcode is **1010.** The results of the trivial tests show that the line can neither be trivially rejected or accepted. Point **E** is determined to be an outside point, so the algorithm clips the line against the

bottom edge of the window. Now line EI has been clipped to be line FI. Line FI is tested and cannot be trivially accepted or rejected. Point F has an outcode of 0000, so the algorithm chooses point I as an outside point since its outcodde is 1010. The line FI is clipped against the window's top edge, yielding a new line FH. Line FH cannot be trivially accepted or rejected. Since H's outcode is 0010, the next iteration of the algorthm clips against the window's right edge, yielding line FG. The next iteration of the algorithm tests FG, and it is trivially accepted and display.

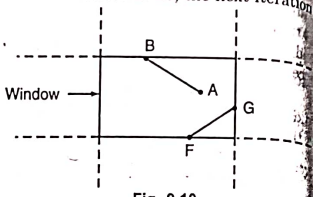


Fig. 9.10.

After Clipping

After clipping the segments AD and EI, the result is that only the line segments AB and FG can be seen in the window.

9.6.2. Line Intersection and Clipping

We determine the intersection points of the lines in category 3 *i.e.* clipping condidate with the boundaries of the window. There intersection points then subdivide the line segment into several line segments which can belong only to visible or nonvisible category *i.e.* category 1 or category 2. The segment in category 1 will be clipped line segment.

The intersection points are found by solving the equations representing the line segment and the boundary lines.

- For left window edge, intersection point will be (x_L, y)
$$y = m(x_L - x_1) + y_1,\ m \neq \infty$$
- For right window edge, intersection point will be (x_R, y)
$$y = m(x_R - x_1) + y_1,\ m \neq \infty$$
- For top window edge, intersection point will be (x, y_T)
$$y = x_1 + \frac{1}{m}(y_T - y_1),\ m \neq 0$$
- For bottom window edge, intersection point will be (x, y_B)
$$x = x_1 + \frac{1}{m}(y_B - y_{1m}),\ m \neq 0$$

Note:
In all above equations
x_L is x-value of left edge of clipping window
x_R is x-value of right edge of clipping window
y_B is y-value of bottom edge of clipping window
y_T is y-value of top edge of clipping window.
and acceptable value of x and y are
$$x_L \leq x \leq x_R$$
$$y_B \leq y \leq y_T$$

EXAMPLE 9.2

Use the cohen Sutherland algorithm to clip line P_1 (70, 20) and P_2 (100, 10) against a window lower left hand corner (50, 10) and upper right hand corner (80, 40).

Solution: $P_1(70, 20)$ and $P_2(100, 10)$
Window lower left corner = (50, 10)
Window upper right corner = (80, 40)

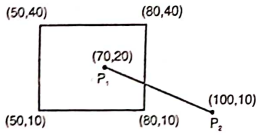


Fig. 9.11.

So, the window is
Now, we assign 4 bit binary outcode.
Point P_1 is inside the window so outcode of P_1 = 00000
and the outcode of P_2 = 0010 as point P_2 is right of window AND of P_1 and P_2

$$\begin{array}{r} 0\ 0\ 0\ 0 \\ 0\ 0\ 1\ 0 \\ \hline 0\ 0\ 0\ 0 \end{array}$$

The result of AND operation is zero so line is partially visible. Slop of line P_1P_2 is $m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{10-20}{100-70} = \frac{-10}{30} = -\frac{1}{3}$

We have to find intersection of line $P_1\ P_2$ with right edge of window *i.e.* Point P_2
Let the intersection point be (x, y)
Here $x = 80$, we have to find the value of y.
We use point $P_2(x_2, y_2) = P_2(100, 10)$

$$m = \frac{y - y_2}{x - x_2}$$

$$-\frac{1}{3} = \frac{y - 10}{80 - 100}$$

$$y - 10 = \frac{20}{3} \Rightarrow y = \frac{20}{3} + 10 = 16.66.$$

$$\therefore \quad y = 16.66$$

Thus the intersection point P_3 = (80, 16.66)
So, after clipping line P_1P_2 against window, new line is P_1P_3 with co-ordinates P_1(70, 20) and P_3(80, 16.66).

EXAMPLE 9.3

Given a clipping window A(20, 20) B(60, 20) C(60, 40), D(20, 40). Using Sutherland Cohen algorithm find the visible portion of line segment joining the points P(40, 80) Q(120, 30).

Solution:

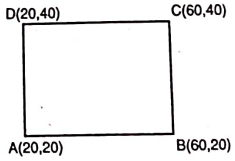


Fig. 9.12.

Here $x_L = 20 \quad y_B = 20$

$x_R = 60 \quad y_T = 40$

As we know, the outcodes can be calculated as

Bit 1 = sign of $(y - y_T)$

Bit 2 = sign of $(y_B - y)$

Bit 3 = sign of $(x - x_R)$

Bit 4 = sign of $(x_L - x)$

and sign = 1 if value is +ve and sign = 0 if value is –ve.

Thus, the outcodes of $P(40, 80)$ is 1000

and $Q(120, 30)$ is 0010.

Both endpint codes are not zero and their logical AND is zero. Hence, line cannot be rejected as invisible.

$$\text{Slope of } PQ(m) = \frac{y_2 - y_1}{x_2 - x_1} = \frac{30 - 80}{120 - 40} = -\frac{5}{8}$$

and intersections with window edge are

- **Left** $y = m(x_L - x_1) + y_1 = -\frac{5}{8}(20 - 40) + 80$

  $= 92.5$, which is greater than y_T and hence rejected.

- **Right** $y = m(x_R - x_1) + y_1$

  $= -\frac{5}{8}(60 - 40) + 80 = 67.5$, which is greater than y_T

  so, it is rejected.

- **Top** $x = x_1 + \frac{1}{m}(y_T - y_1) = 40 + \frac{1}{-\frac{5}{8}}(40 - 80) = 104,$

  which is greater than x_R and hence rejected.

- **Bottom** $x = x_1 + \frac{1}{m}(y_B - y_1) = 40 + \frac{1}{-\frac{5}{8}}(20 - 80) = 136,$

  which is greater than x_R and hence rejected.

Since both the values of y are greater than y_T and both are also greater than x_R. Therefore the line segment PQ completely outside the window.