Tribhuvan University
Institute of Science and Technology
Bachelor of Science in
Computer Science and Information Technology

3RD Semester

CSC209: Computer Graphics

-RAJESH KUMAR BAJGAIN

U3:2D Geometric Transformations 5Hrs

3.1 Two-Dimensional translation, Rotation, Scaling, Reflection and Shearing

3.2 Homogeneous Coordinate and 2D Composite Transformations. Transformation between Co-ordinate Systems.

3.3 Two Dimensional Viewing: Viewing pipeline, Window to viewport coordinate transformation

3.4 Clipping: Point, Lines(Cohen Sutherland line clipping, Liang-Barsky Line Clipping) , Polygon Clipping(Sutherland Hodgeman polygon clipping)

2D Geometric Transformation

- Changing co-ordinate description of an object is called transformation.

- Rigid body transformation (transformation without change in shape.)
- Non rigid body transformation (transformation with change in shape.)

- When a transformation takes place on a 2D plane, it is called 2D transformation.
- The three basic transformations are

Translation **Rotation** **Scaling**

- Other transformation includes **reflection** and **shearing**.

Homogenous form

- Provides a uniform framework for handling different geometric transformations, simply as multiplication of matrices.
- To perform more than one transformation at a time, homogeneous coordinates are used.
- They reduce unwanted calculations, intermediate steps, saves time and memory and produce a sequence of transformations.
- We represent each Cartesian coordinate position (x, y) with the homogeneous coordinate triple(x_h, y_h, h), where, $x = x_h/h$, $y = y_h/h$. (h is 1 usually for 2D case).
- Therefore, (x, y) in Cartesian system is represented as (x, y, 1) in homogeneous co-ordinate system.

Composite transformation

- When two or more transformations are performed on a figure to produce a new figure is called composite transformation

The basic purpose of composing transformation is

- to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformation, one after another
- i.e. to reduce the number of operations
- To make transformations compact

2D Translation

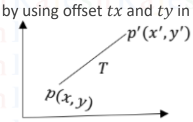
Repositioning of object along a straight-line path from one coordinate location to another is called translation.

Translation is performed on a point by adding offset to its coordinate so as to generate a new coordinate position.

Let p(x, y) be translated to $p'(x', y')$ by using offset tx and ty in x & y direction. Then,

$xx' = x + t_x$

$yy' = y + t_y$

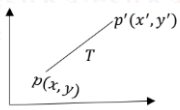


2D Translation

In matrix form,

i.e. $P' = P + T$ where T is transformation matrix.

$P = \begin{bmatrix} x \\ y \end{bmatrix}$ $T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$ $P' = \begin{bmatrix} x' \\ y' \end{bmatrix}$



Homogeneous representation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow P' = T(t_x, t_y).P$$

Composite 2D translation

If two successive translation vector (tx1, ty1) & (tx2, ty2) is applied on position $p(x, y)$, then translated point is given by,

$PP' = T_{(tx2,ty2)}.\, T_{(tx1,ty1)}.\, P$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Which gives,

$xx' = x + t_{x1} + t_{x2}$

$yy' = y + t_{y1} + t_{y2}$

2D Rotation

Changing the co-ordinate position along a circular path is called rotation.

2D rotation is applied to re-position the object along a circular path in XY-plane. Rotation is generated by specifying rotation angle (θ) and pivot point (rotation point).

The positive θ rotates object in anti-clockwise direction and the negative value of θ rotates the object in clockwise direction.

Let $p(x, y)$ be a point rotated by θ about origin to new point $p'(x', y')$.

2D Rotation

Here,

$xx' = rcos(\emptyset + \theta)$

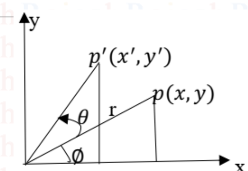
$= rcos\emptyset cos\theta - rsin\emptyset sin\theta$

But $x = rcos\emptyset$ & $y = rsin\emptyset$

$\therefore x' = \boldsymbol{xcos\theta - ysin\theta}$ .... (i)

Similarly,

$\therefore y' = \boldsymbol{xsin\theta + ycos\theta}$ .... (ii)



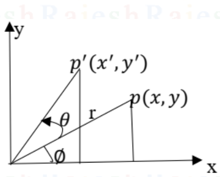
Which are equation for rotation of (x, y) with angle θ and taking pivot as origin.

2D Rotation

In matrix form

$PP' = R.\, P$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



2D Rotation

In homogeneous co-ordinate:

Anticlockwise direction

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow P' = R(\theta).P$$

Clockwise direction

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow P' = R(\theta).P$$

2D Rotation

If the pivot point is at(x_r, y_r).

$xx' = x_r + (x - x_r)cos\theta - (y - y_r)sin\theta$

$yy' = y_r + (x - x_r)sin\theta + (y - y_r)cos\theta$

a) Translate the object so that pivot point position is moved to coordinate origin.

b) Rotate the object about coordinate origin.

c) Translate the object so that pivot point is returned to original position.

2D Rotation

Matrix representation:

$$\begin{bmatrix}1 & 0 & x_r\\0 & 1 & y_r\\0 & 0 & 1\end{bmatrix}.\begin{bmatrix}cos\theta & -sin\theta & 0\\sin\theta & cos\theta & 0\\0 & 0 & 1\end{bmatrix}.\begin{bmatrix}1 & 0 & -x_r\\0 & 1 & -y_r\\0 & 0 & 1\end{bmatrix}$$

$$= \begin{bmatrix}cos\theta & -sin\theta & x_r(1-cos\theta)+y_r sin\theta\\sin\theta & cos\theta & y_r(1-cos\theta)-x_r sin\theta\\0 & 0 & 1\end{bmatrix}$$

2D composite rotation

$PP' = (R(\theta_2)).\ (R(\theta_1)).\ P$

$= \mathrm{R}\ (\theta_1 + \theta_2).\ P$

2D Scaling

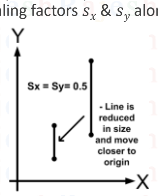
Alters the size of object and operation is performed by multiplying object position (x, y) with scaling factors s_x & s_y along x & y direction to produce (x', y').

$xx' = x.\ s_x$ & $y' = y.\ s_y$

In matrix form,

$PP' = S.\ P$

$$\begin{bmatrix}x'\\y'\end{bmatrix} = \begin{bmatrix}s_x & 0\\0 & s_y\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}$$



2D Scaling

If the scaling factor is less than 1, the size of object is decreased and if it is greater than 1 the size of object is increased.

The scaling factor = 1 for both direction does not change the size of the object.

If both scaling factors have same value then the scaling is known as uniform scaling.

If the value of s_x and s_y are different, then the scaling is known as differential scaling.

The differential scaling is mostly used in the graphical package to change the shape of the object.

2D Scaling

In homogeneous system

$$\begin{bmatrix}x'\\y'\\1\end{bmatrix} = \begin{bmatrix}s_x & 0 & 0\\0 & s_y & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\y\\1\end{bmatrix} \Rightarrow P' = S(s_x, s_y).P$$

If fixed point is (x_f, y_f) about which rotation is made, then

$xx' = x.\ s_x + x_f(1 - s_x)$

$yy' = y.\ s_y + y_f(1 - s_y)$

$$\begin{bmatrix}1 & 0 & x_f\\0 & 1 & y_f\\0 & 0 & 1\end{bmatrix}.\begin{bmatrix}s_x & 0 & 0\\0 & s_y & 0\\0 & 0 & 1\end{bmatrix}.\begin{bmatrix}1 & 0 & -x_f\\0 & 1 & -y_f\\0 & 0 & 1\end{bmatrix}$$

$$= \begin{bmatrix}s_x & 0 & x_f(1-s_x)\\0 & s_y & y_f(1-s_y)\\0 & 0 & 1\end{bmatrix}$$

2D composite Scaling

$PP' = S_{(Sx2,Sy2)}.\ S_{(Sx1,Sy1)}.\ P$

$$P' = \begin{bmatrix}s_{x2} & 0 & 0\\0 & s_{y2} & 0\\0 & 0 & 1\end{bmatrix}.\begin{bmatrix}s_{x1} & 0 & 0\\0 & s_{y1} & 0\\0 & 0 & 1\end{bmatrix}.\begin{bmatrix}x\\y\\1\end{bmatrix}$$

Prove that two successive translations are additive.

If two successive translation vector (tx1, ty1) & (tx2, ty2) is applied to coordinate position P, the final transformed location P' is calculated with the following composite transformation as,

$TT = T_{(tx2,ty2)}.\ T_{(tx1,ty1)}$

$$= \begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{x1}+t_{x2} \\ 0 & 1 & t_{y1}+t_{y2} \\ 0 & 0 & 1 \end{bmatrix}$$

Hence, $T_{(tx2,ty2)}.\ T_{(tx1,ty1)} = T_{(tx1+tx2,ty1+ty2)}$ which demonstrates that two successive translations are additive.

Prove that two successive rotation are additive

Let P be the point anticlockwise rotated by angle $\theta 1$ to point P' and again let P' be rotated by angle $\theta 2$ to point P", then the combined transformation can be calculated with the following composite matrix as:

$TT = R(\theta_2).\ R(\theta_1)$

$$= \begin{bmatrix} cos\theta_2 & -sin\theta_2 & 0 \\ sin\theta_2 & cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} . = \begin{bmatrix} cos\theta_1 & -sin\theta_1 & 0 \\ sin\theta_1 & cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} cos\theta_2 * cos\theta_1 - sin\theta_2 * sin\theta_1 & -cos\theta_2 * sin\theta_1 - sin\theta_2 * cos\theta_1 & 0 \\ sin\theta_2 * cos\theta_1 + cos\theta_2 * sin\theta_1 & -sin\theta_2 * sin\theta_1 + cos\theta_2 * cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Prove that two successive rotation are additive

$$= \begin{bmatrix} \cos(\theta_1+\theta_2) & -sin(\theta_1+\theta_2) & 0 \\ sin(\theta_1+\theta_2) & cos(\theta_1+\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

i.e. $R(\theta 2).\ R(\theta 1) = R(\theta 1 + \theta 2)$ which demonstrates that two successive rotations are additive.

Prove that two successive scaling are multiplicative.

Let point P is first scaled with scaling factors $sx1$, $sy1$ to P' and again let P' be scaled by scaling factors $sx2$, $sy2$ to point P", then the combined transformation can be calculated with the following composite matrix

$TT = S_{(Sx2,Sy2)}.\ S_{(Sx1,Sy1)} = \begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$$= \begin{bmatrix} s_{x1}s_{x2} & 0 & 0 \\ 0 & s_{y1}s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

i.e. $S_{(Sx2,Sy2)}.\ S_{(Sx1,Sy1)} = S_{(Sx1Sx2,Sy1Sy2)}$ which demonstrates that two successive scaling are multiplicative.

1. Find the scaled triangle with vertices A(0, 0), B(1, 1) & C(5, 2) after it has been magnified twice its size.
2. Rotate a triangle A(0, 0), B(2, 2), C(4, 2) about the origin by the angle of 45 degree.
3. Rotate a triangle (5, 5), (7, 3), (3, 3) about fixed point (5, 4) in counter clockwise by 90 degree.
4. Rotate a triangle A(7, 15), B(5, 8) & C(10, 10) by 45 degree clockwise about origin and scale it by (2, 3) about origin.
5. A square with vertices A(0, 0), B(2, 0), C(2, 2) & D(0, 2) is scaled 2 units in x & y direction about the fixed point (1, 1). Find the coordinates of the vertices of new square.
6. A triangle having vertices A(3, 3), B(8, 5) & C(5, 8) is first translated by 2 units about fixed point (5, 6) & finally rotated 90 degree anticlockwise about pivot point (2, 5). Find the final position of triangle.
7. Rotate the △ABC by 90° anti-clock wise about (5, 8) and scale it by (2, 2) about (10, 10).

Reflection

Providing a mirror image about an axis of an object is called reflection.

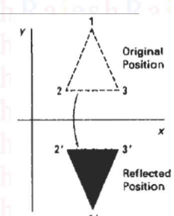
Reflection about x-axis (y=0)

The reflection of a point P(x, y) on x-axis, changes the y-coordinate sign i.e. P(x, y) changes to P'(x, -y).

In matrix form, $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

$PP' = R_{fx}.\ P$

RR_{fx}= Reflection matrix about x-axis.



Reflection about y-axis (x=0)

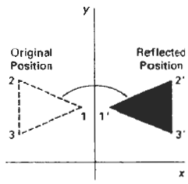
The reflection of a point P(x, y) on y-axis changes the sign of x-coordinate i.e. P(x, y) changes to P'(-x, y).

In matrix form,

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$PP' = R_{fy}. P$

RR_{fy}= Reflection matrix about y-axis.



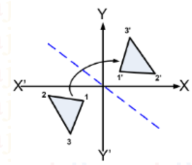
Reflection about origin

Flip both x & y coordinate of a point.

x' = -x

y' = -y

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



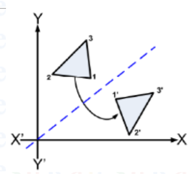
Reflection about line y = x

x' = y

y' = x

Homogeneous Form

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Reflection about line y = -x

x' = -y

y' = -x

Homogeneous Form

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

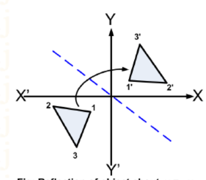


Fig: Reflection of object about y = - x

Reflection about y=mx+c

Perform the following transformation:

a) Translate the line so that it passes through origin.

b) Rotate the line so that it coincides with any coordinate axis.

c) Reflect object about that axis.

d) Perform reverse rotation.

e) Perform reverse translation so that line is placed to its original position.

Shearing

It distorts the shape of an object such that the transformed shape appears as if the object is composed of internal layers and these layers are caused to slide over is shearing.

X-direction Shear:

An X-direction shear relative to x-axis is produced with transformation matrix equation.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Which transforms,

$xx' = x + sh_x. y$

$yy' = y$

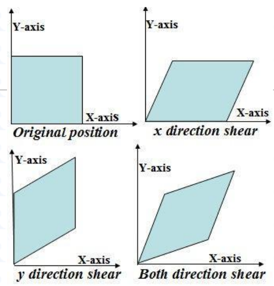


Fig. Two Dimensional shearing

Shearing

Y-direction shear:

A Y-direction shear relative to y-axis is produced by following transformation equations.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Which transforms,

$xx' = x$

$yy' = x.\, sh_y + y$

Shearing

X-direction shear relative to $y = y_{ref}$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & -sh_x.y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$xx' = x + sh_x(y - y_{ref})$

$yy' = y$

Y-direction shear relative to $x = x_{ref}$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y.x_{ref} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$xx' = x$

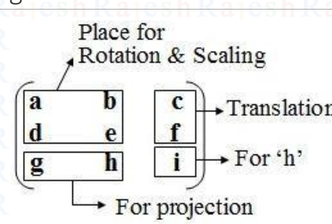
$yy' = y + sh_y(x - x_{ref})$

1. A triangle having vertices A(2, 3), B(6, 3) & C(4,8) is reflected about y=3x+4. Find the final position of triangle.
2. Derive the composite matrix for reflecting an object about any arbitary line y=mx+c.

Find the coordinate of a triangle A(1,3) , B(2,5) and C(3,3) after being rotated about fixed point p(2,4) by 45 in clockwise direction and then translate (3,4) .

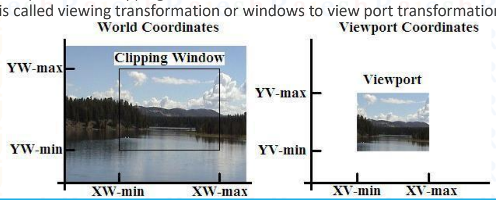
Reflect an object (2, 3), (4, 3), (4, 5) about line y = x +1.

Homogenous Coordinates

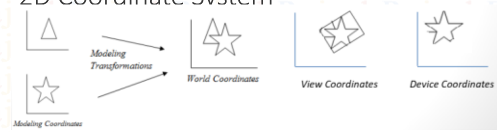


2D Viewing

The process of mapping the world coordinate scene to device coordinate is called viewing transformation or windows to view port transformation.



2D Coordinate System



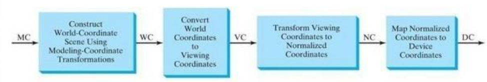
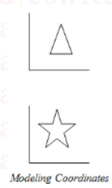


Fig: Two-dimensional viewing transformation pipeline

Modeling Coordinates

- Modeling coordinates are used to construct shape of individual parts (objects or structures) of a 2D scene. For example, generating a circle at the origin with a "radius" of 2 units.
- Here, origin (0, 0), and radius 2 units are modeling coordinates.
- Modeling coordinates define object shape.
- Can be floating-point, integers and can represent units like km, m,miles, feet etc.



World Coordinates

- World coordinates are used to organize the individual parts into a scene.
- World coordinates units define overall scene to be modeled.
- World coordinates represent relative positions of objects.
- Can be floating-point, integers and can represent units like km,m, miles etc.

Viewing Coordinates

- Viewing coordinates are used to define particular view of the user. Viewer's position and view angle i.e. rotated/translated.
- Viewing coordinates specify the portion of the output device that is to be used to present the view.

Normalized viewing coordinates

Normalized viewing coordinates are viewing coordinates between 0 and 1 in each direction.

They are used to make the viewing process independent of the output device (paper, mobile).

Device Coordinates or Screen Coordinates

- The display coordinate system is called device coordinate system.
- Device coordinates are specific to output device.
- Device coordinates are integers within the range (0, 0) to (xmax, ymax) for a particular output device.

Device Coordinates

Window and Viewport

Window: • A world-coordinate area selected for display is called a window or clipping window. That is, window is the section of the 2D scene that is selected for viewing.

- The window defines what is to be viewed.

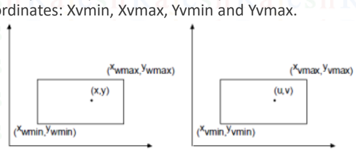
Viewport: • An area on a display device to which a window is mapped is called a viewport.

- The viewport indicates where on an output device selected part will be displayed.

Window to Viewport transformation

A window is specified by four world coordinates: Xwmin, Xwmax, Ywmin and Y wmax .

Similarly, a viewport is described by four normalized device coordinates: Xvmin, Xvmax, Yvmin and Yvmax.



Window to Viewport transformation

1. Translate the window to the origin. That is, apply T (-Xwmin, -Ywmin)
2. Scale it to the size of the viewport. That is, apply S (sx, sy)
3. Translate scaled window to the position of the viewport. That is, apply T(Xvmin, Yvmin).

Therefore, net transformation,

T_{WV} = T (Xvmin, Yvmin). S(sx, sy).T(-Xwmin, -Ywmin)

Let (x, y) be the world coordinate point that is mapped onto the viewport point (u, v), then we must have

$$\frac{u - x_{v\min}}{x_{v\max} - x_{v\min}} = \frac{x - x_{w\min}}{x_{w\max} - x_{w\min}}$$

$$u = x_{vmin} + \frac{x_{v\max} - x_{v\min}}{x_{w\max} - x_{w\min}} \cdot (x - x_{wmin})$$

$$\frac{v - y_{v\min}}{y_{v\max} - y_{v\min}} = \frac{y - y_{w\min}}{y_{w\max} - y_{w\min}}$$

$$v = y_{vmin} + \frac{y_{v\max} - y_{v\min}}{y_{w\max} - y_{w\min}} \cdot (y - y_{wmin})$$

But, we know that

$$u = x_{vmin} + s_x(x - x_{wmin})$$
$$v = y_{vmin} + s_y(y - y_{wmin})$$

$$s_x = \frac{x_{v\max} - x_{w\min}}{x_{w\max} - x_{w\min}}$$

$$s_y = \frac{y_{v\max} - y_{v\min}}{y_{w\max} - y_{w\min}}$$

By solving these equations for the unknown viewport position (u, v) the following becomes true: u = $s_x x + t_x$
$v = s_y y + t_y$

Where, $s_x = \frac{x_{v\,max} - x_{v\,min}}{x_{w\,max} - x_{w\,min}}$ $\quad s_y = \frac{y_{v\,max} - y_{v\,min}}{y_{w\,max} - y_{w\,min}}$

And $t_x = \frac{x_{w\,max} \cdot x_{v\,min} - x_{w\,min} \cdot x_{v\,max}}{x_{w\,max} - x_{w\,min}}$ $\quad t_y = \frac{y_{w\,max} \cdot y_{v\,min} - y_{w\,min} \cdot y_{v\,max}}{y_{w\,max} - y_{w\,min}}$

Now, window- to – viewport transformation matrix is:

$$T_{wv} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Window port is given by (100, 100, 300, 300) and viewport is given by (50, 50, 150, 150). Convert the window port coordinate (200, 200) to the view port coordinate.

Here,

$(xwmin, ywmin)$ = (100,100)

$(xwmax, ywmax)$ = (300,300)

$(xvmin, yvmin)$ = (50,50)

$(xvmax, yvmax)$ = (150,150)

(xw, yw) = (200,200)

Then we have

The equation for mapping window coordinate to view port coordinate is given by,

$xxv = s_x xw + t_x$

$yyv = s_y yw + t_y$

Hence,

xxv = 0.5 × 200 + 0 = 100

yyv = 0.5 × 200 + 0 = 100

The transformed viewport coordinate is (100, 100).

Find the normalization transformation matrix for window to viewport which uses the rectangle whose lower left corner is at (2, 2) and upper right corner is at (6, 10) as a window and the viewport that has lower left corner at (0, 0) and upper right corner at (1,1).

We have

$$s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} = \frac{1-0}{6-2} = 0.25$$

$$s_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}} = \frac{1-0}{10-2} = 0.125$$

$$t_x = \frac{xw_{max} \cdot xv_{min} - xw_{min} \cdot xv_{max}}{xw_{max} - xw_{min}} = \frac{6\times 0 - 2\times 1}{6-2} = -0.5$$

$$t_y = \frac{yw_{max} \cdot yv_{min} - yw_{min} \cdot yv_{max}}{yw_{max} - yw_{min}} = \frac{10\times 0 - 2\times 1}{10-2} = -0.25$$

The composite transformation matrix for transforming the window coordinate to viewport coordinate is given as

$$T = T_{(t_x, t_y)} S_{(s_x, s_y)}$$

$$= \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.25 & 0 & -0.5 \\ 0 & 0.125 & -0.25 \\ 0 & 0 & 1 \end{bmatrix}$$

A world coordinate & viewport have the following geometry: Window (left, right, bottom, top) = (200, 600, 100, 400) Viewport (left, bottom, width, height) = (0, 0, 800, 600) The following vertices are drawn in the world:- P1: (356, 125), P2: (200, 354), P3: (230,400), P4: (564, 200). What coordinate will each occupy in viewport.

Here,

$(xw_{min}, yw_{min}) = (200,100)$

$(xw_{max}, yw_{max}) = (600,400)$

$(xv_{min}, yv_{min}) = (0,0)$

$(xv_{max}, yv_{max}) = (800,600)$

Then we have

$$s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} = \frac{800-0}{600-200} = 2$$

$$s_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}} = \frac{600-0}{400-100} = 2$$

$$t_x = \frac{xw_{max}.xv_{min} - xw_{min}.xv_{max}}{xw_{max} - xw_{min}} = \frac{600\times0 - 200\times800}{600-200} = -400$$

$$t_y = \frac{yw_{max}.yv_{min} - yw_{min}.yv_{max}}{yw_{max} - yw_{min}} = \frac{400\times0 - 100\times600}{400-100} = -200$$

The composite transformation matrix for transforming the window coordinate to viewport coordinate is given as

$$M = T_{(t_x,t_y)}S_{(s_x,s_y)}$$

$$= \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 0 & -400 \\ 0 & 2 & -200 \\ 0 & 0 & 1 \end{bmatrix}$$

Now,

$$P1' = M.P1 = \begin{bmatrix} 2 & 0 & -400 \\ 0 & 2 & -200 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 356 \\ 125 \\ 1 \end{bmatrix} = \begin{bmatrix} 312 \\ 50 \\ 1 \end{bmatrix} = (312, 50)$$

$$P2' = M.P2 = \begin{bmatrix} 2 & 0 & -400 \\ 0 & 2 & -200 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 200 \\ 354 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 508 \\ 1 \end{bmatrix} = (0, 508)$$

$$P3' = M.P3 = \begin{bmatrix} 2 & 0 & -400 \\ 0 & 2 & -200 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 230 \\ 400 \\ 1 \end{bmatrix} = \begin{bmatrix} 60 \\ 600 \\ 1 \end{bmatrix} = (60, 600)$$

$$P4' = M.P4 = \begin{bmatrix} 2 & 0 & -400 \\ 0 & 2 & -200 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 564 \\ 200 \\ 1 \end{bmatrix} = \begin{bmatrix} 728 \\ 200 \\ 1 \end{bmatrix} = (728, 200)$$

Clipping

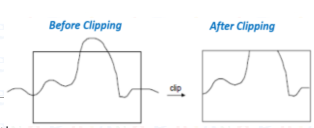
The process of discarding those parts of a picture which are outside of a specified region or window is called clipping.

The procedure using which we can identify whether the portions of the graphics object is within or outside a specified region or space is called clipping algorithm.

The region or space which is used to see the object is called window and the region on which the object is shown is called view port.

Clipping is necessary to remove those portions of the object which are not necessary for further operations.

Clipping



It excludes unwanted graphics from the screen. So, there are three cases:

1. The object may be completely outside the viewing area defined by the window port.
2. The object may be seen partially in the window port.
3. The object may be seen completely in the window port.

For case 1 & 2, clipping operation is necessary but not for case 3.

Applications of clipping:

- 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-modeling procedures
- Displaying a multi-window environment
- Drawing and painting operations that allow parts of a picture to be selected for copying, moving, erasing, or duplicating.

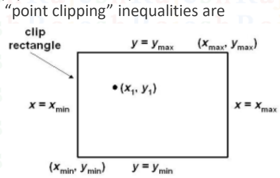
Point Clipping

- Let W denote a clip window with coordinates (Xwmin, Ywmin), (Xwmin, Ywmax), (Xwmax, Ywmin), (Xwmax, Ywmax), then a vertex (x, y) is displayed only if following "point clipping" inequalities are satisfied:

  Xwmin ≤ x ≤ Xwmax,

  Ywmin ≤ y ≤ Ywmax.

- Very simple and efficient.
- Only works for vertices.



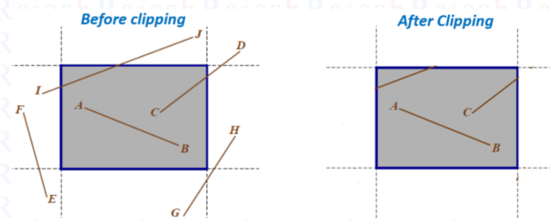
Point Clipping: Algorithm

1. Get the minimum and maximum coordinates of the viewing plane.
2. Get the coordinates for a point.
3. Check whether given input lies between minimum and maximum coordinates of viewing plane.
4. If yes display the point which lies inside the region otherwise discard it.

Line Clipping

In line clipping, a line or part of line is clipped if it is outside the window port. There are three possibilities for the line:

1. Line can be completely inside the window (This line should be accepted).
2. Line can be completely outside of the window (This line will be completely removed from the region).
3. Line can be partially inside the window (We will find intersection point and draw only that portion of line that is inside region).

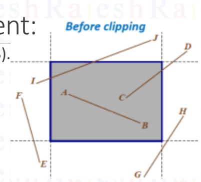


For any particular line segment:

a. Both endpoints are inside the region (line AB).
   - No clipping necessary.

b. One endpoint is inside and one is outside of the clipping window (line CD).
   - Clip at intersection point.

c. Both endpoints are outside the region:
   - No intersection (lines EF, GH)
   - Line intersects the region (line IJ)
   - Discard the line segment.
   - Clip line at both intersection points.

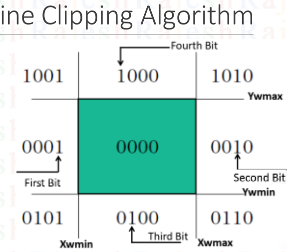


Cohen-Sutherland Line Clipping Algorithm

Divide the coordinate system into nine regions.

All regions have their associated region codes i.e. a bit pattern of TBRL as shown:

Every line endpoint is assigned a four digit binary code (region code or out code) as TBRL.



Establish Region code for all line end point

- First bit is 1, if x < Xwmin (Point lies to left of window) , else set it to 0
- Second bit is 1, if x > Xwmax (Point lies to right of window) , else set it to 0
- Third bit is 1, if y < Ywmin (Point lies to below window), else set it to 0
- Fourth bit is 1, if y > Ywmax (Point lies to above window) , else set it to 0

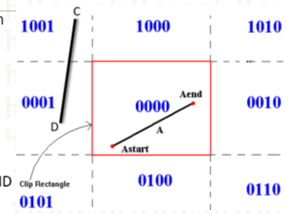
Determine which lines are completely inside window and which are not

a) If logical OR operation of region codes of two end points is '0000' the line is completely inside

b) If logical AND operation of region codes of two end points is NOT '0000' the line is completely outside

Astart 0000
Aend 0000
Completely inside

C 1001
D 0001
AND
0001
Completely Outside



If both tests fail then line is partially visible so we need to find the intersection with boundaries of window

a) If 1st bit is 1 then line intersect with Left boundary and
- $Y_i = Y_1 + m\,(X - X_1)$ where X= Xwmin

b) If 2nd bit is 1 then line intersect with Right boundary and
- $Y_i = Y_1 + m\,(X - X_1)$ where X= Xwmax

c) If 3rd bit is 1 then line intersect with Bottom boundary and
- $X_i = X_1 + (1/m)\,(Y - Y_1)$ where Y= Ywmin

d) If 4th bit is 1 then line intersect with Top boundary and
- $X_i = X_1 + (1/m)\,(Y - Y_1)$ where Y= Ywmax

Here, (X_i , Y_i) are (X ,Y) intercepts for that the step 4 is repeat step 1 through step 3 until line is completely accepted or rejected

Algorithm

Step 1: Given a line segment with endpoints p1(x1,y1) and p2(x2,y2)

Step 2: Compute the 4-bit outcodes for the two endpoints of the line segment

Step 3: If outcodes of p1=outcodes of p2 = 0000 (p1||p2 =0000) then the line lies completely inside the window, so draw the line segment.

Step 4: Else if outcodes of p1 && outcodes of p2 != 0000 (p1&&p2!=0000) then line lies completely outside the window, so discard the line segment.

Step 5: else if outcode of p1 && outcode of p2=0000 (p1&&P2=0000) then compute the intersection of the line segment with the window boundary, and discard the portion of the segment that falls completely outside the window. Assign a new four bit code to the intersection and repeat until either 3 or 4 above are satisfied.

Use the Cohen-Sutherland algorithm to clip the line P1(70, 20) and P2(100, 10) against a window lower left hand corner (50, 10) and upper right hand corner (80, 40)

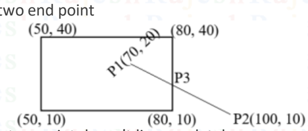
Assign 4 bit binary code to the two end point

P1=0000

P2=0010

Finding logical OR:

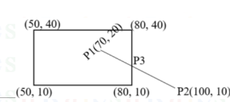
P1||P2=0000||0010=0010



Since P1||P2 !=0000, hence the two point doesn't lie completely inside the window.

Finding logical AND: P1&&P2=0000 && 0010=0000

Since P1&&P2=0000, hence line is partially visible.



Now, finding the intersection of P1 and P2 with the boundary of window.

$pp1(x1, y1) = (70, 20)$ $\quad p2(x2, y2) = (100, 10)$

Slope $m = (10 - 20)/(100 - 70) = -1/3$

We have to find the intersection with right edge of window.

Here, $x = 80$

$yy = y2 + m(x - x2) = 10 + (-1/3)\,(80 - 100) = 10 + 6.67 = 16.67$

Thus the intersection point P3 = (80, 16.67). So, discarding the line segment that lie outside the boundary i.e. P3P2, we get new line P1P3 with coordinate P1(70, 20) and P3(80, 16.67)

Liang-Barsky Line Clipping Algorithm

Based on analysis of parametric equation of a line segment, faster line clippers have been developed, which can be written in the form:

$x = x_1 + u\Delta x$ $y = y_1 + u\Delta y$,

Where, $0 \le u \le 1$ and $\Delta x = x_2 - x_1$ & $\Delta y = y_2 - y_1$

Cryus and Beck developed an algorithm based on these parametric equation which is more efficient than Cohen-Sutherland algorithm.

Later, Liang and Barsky independently devised an even faster parametric line clipping algorithm.

In which, we first write the point clipping in a parametric way as:

Liang-Barsky Line Clipping Algorithm

$xx_{wmin} \le x_1 + u\Delta x \le x_{wmax}$

$yy_{wmin} \le y_1 + u\Delta y \le y_{wmax}$

These four inequalities can be expressed as,

$uup_k \le q_k$

Where, k = 1, 2, 3, 4 (corresponds to the left, right, bottom, and top boundaries, respectively).

Liang-Barsky Line Clipping Algorithm

The parameters p & q are defined as,

$pp1 = -\Delta x$, $q1 = x1 - xwmin$ (Left Boundary)

$pp2 = \Delta x$, $q2 = xwmax - x1$ (Right Boundary)

$pp3 = -\Delta y$, $q3 = y1 - ywmin$ (Bottom Boundary)

$pp4 = \Delta y$, $q4 = ywmax - y1$ (Top Boundary)

Liang-Barsky Line Clipping Algorithm

When the line is parallel to a view/window boundary, the p value for the boundary is zero.

When $pk = 0$ & $qk < 0$ then line is trivially invisible because it is outside view window.

When $pk = 0$ & $qk > 0$ then line is inside the corresponding window boundary.

When $pk < 0$, as u increase line goes from the outside to inside (entering).

When $pk > 0$, line goes from the inside to outside (exiting).

Liang-Barsky Line Clipping Algorithm

Using the following conditions, the position of line can be determined:

Condition	Position of line
$p_k = 0$	Parallel to the clipping boundaries.
$p_k = 0$ & $q_k < 0$	Completely outside the boundary.
$p_k = 0$ & $q_k \ge 0$	Inside the parallel clipping boundary.
$p_k < 0$	Line proceeds from outside to inside.
$p_k > 0$	Line proceeds from inside to outside.

Liang-Barsky Line Clipping Algorithm

Parameters $u1$ & $u2$ can be calculated that define the part of line that lies within the clip rectangle.

1. When $pk < 0$, for $u1$: $maximum(0, \frac{q_k}{p_k})$ is taken.

2. When $pk > 0$, for $u2$: $minimum(1, \frac{q_k}{p_k})$ is taken.

If $u1 > u2$, the line is completely outside the clip window and it can be rejected. Otherwise,

the endpoints of the clipped line are calculated from the two values of parameter u.

Algorithm

1. Read two end points of a line say p1(x1,y1) and p2(x2,y2).
2. read two corners (left top and right bottom) of the window, say (x_{wmin},y_{wmax}, x_{wmax},y_{wmin})
3. Calculate the values of parameters pk and qk for k=1, 2, 3, 4 such that

$pp1 = -\Delta x$, $q1 = x1 - xwmin$

$pp2 = \Delta x$, $q2 = xwmax - x1$

$pp3 = -\Delta y$, $q3 = y1 - ywmin$

$pp4 = \Delta y$, $q4 = ywmax - y1$

Algorithm

4. If pk=0 then

{ the line is parallel to the k[th] boundary.

Now, if qk<0 then

{ line is completely outside the boundary, hence discard the segment and goto stop. }

else

{ Check whether the horizontal or vertical and accordingly check the line end point with corresponding boundaries. If the line endpoint(s) lie within the bounded area then use them to draw line otherwise use boundary coordinates to draw the line. Goto stop. }

}

Algorithm

5. Initialize the values of $u1$ and $u2$ as $u1$=0 and $u2$=1
6. Calculate the values for qk/pk for k=1,2,3,4
7. Select values of qk/pk where pk<0 and assign maximum out of them as $u1$.
8. Select values of qk/pk where pk>0 and assign minimum out of them as $u2$
9. If ($u1$< $u2$) {Calculate the endpoints of the clipped line as follows:

   nx1=x1+ $u1\Delta x$
   ny1=y1+ $u1\Delta$y
   nx2=x1+ $u2\Delta x$
   ny2=y1+ $u2\Delta$y
   Draw line (nx1,ny1,nx2,ny2) }
10. Stop

Liang-Barsky Line Clipping Algorithm

Advantages:

- It is more efficient then Cohen-Sutherland algorithm, since intersection calculations are reduced.
- It requires only one division to update parameters u1 and u2
- Window intersection of the line are computed only once.

Apply Liang Barsky Line Clipping algorithm to the line with coordinates (30, 60) and (60, 25) against the window ($\boldsymbol{xwmin}, \boldsymbol{ywmin}$) = ($\mathbf{10}, \mathbf{10}$) and ($\boldsymbol{xwmax}, \boldsymbol{ywmax}$) = ($\mathbf{50}, \mathbf{50}$)

(x1,y1)=(30,60)

(X2,y2)=(60,25)

($xwmin, ywmin$) = (10, 10) and

($xwmax, ywmax$) = (50, 50)

Set $u1$ = 0, $u2$ = 1

P1=-30	q1=20	q1/p1= -0.67
P2=30	q2=20	q2/p2= 0.67
P3=35	q3=50	q3/p3= 1.43
P4=-35	q4=-10	q4/p4= 0.29

Since, pk!=0, line is not parallel to any of the boundaries.

For pk<0, (entering condition) $u1$ = 0, so updating $u1$ as
$u1$ = max(0, -0.67, 0.29) = 0.29

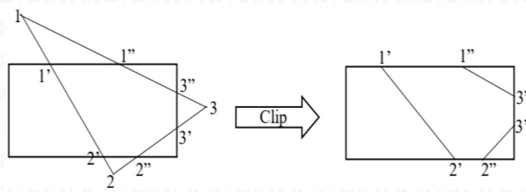
Similarly, for pk>0, (exiting condition) $u2$ = 1, so updating $u2$ as
$u2$ = min(1, 0.67, 1.43) = 0.67

Here, u1<u2, so line is not completely invisible. Hence calculating the new coordinates as

nx1=x1+ $u1\Delta x$ = 30 + 0.29 * 30 = 38.7 = 39
ny1=y1+ $u1\Delta$y = 60 + 0.29 * (-35) = 49.85 50
nx2=x1+ $u2\Delta x$ = 30 + 0.67 * 30 = 50.1 = 50
ny2=y1+ $u2\Delta$y = 60 + 0.67 * (-35) = 36.55 = 37

Finally the coordinates of clipped line to draw is (39,50) and (50,37)

Polygon Clipping



Polygon Clipping

A polygon can be defined as a geometric object "consisting of a number of points (called vertices) and an equal number of line segments (called sides or edges).

Polygon clipping is defined as the process of removing those parts of a polygon that lie outside a clipping window.

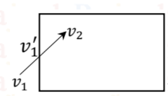
Sutherland-Hodgman Polygon Clipping Algorithm:

In this algorithm, all the vertices of the polygon are clipped against each edge of the clipping window.

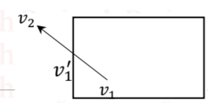
First the polygon is clipped against the left edge of the clipping window to get new vertices of the polygon.

These new vertices are used to clip the polygon against right edge, top edge, bottom edge, of the clipping window.

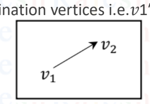
To find new sequence of vertices four cases are considered.



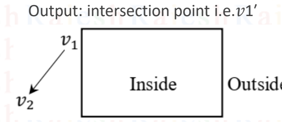
Case I
Movement: $out \rightarrow in$
Output: intersection point, destination vertices i.e. $v1'$, $v2$



Case II
Movement: in → out
Output: intersection point i.e. $v1'$

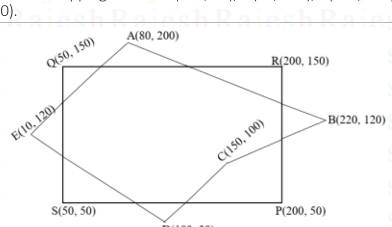


Case III
Movement: in → in
Output: destination vertices i.e. $v2$



Case IV
Movement: out → out
Output: none

Clip polygon against clipping window PQRS. The coordinates of polygon are A(80,200), B(220, 120), C(150, 100), D(100, 30), E(10, 120). Coordinates of clipping window P(200, 50), Q(50, 150), R(200, 150) & S(50, 50).

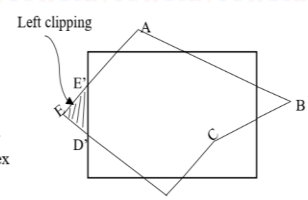


Left Clipping

vertex	case	O/P
AB	$in \rightarrow in$	B
BC	$in \rightarrow in$	C
CD	$in \rightarrow in$	D
DE	$in \rightarrow out$	D’
EA	$out \rightarrow in$	E’, A

New vertex

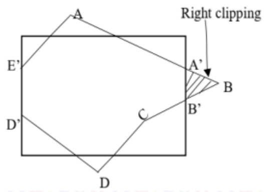
Left clipping



Right Clipping

vertex	case	O/P
AB	$in \rightarrow out$	A’
BC	$out \rightarrow in$	B’, C
CD	$in \rightarrow in$	D
DD’	$in \rightarrow in$	D’
D’E’	$in \rightarrow in$	E’
E’A	$in \rightarrow in$	A

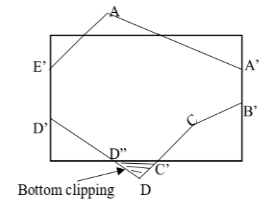
New vertex



Bottom Clipping

vertex	case	O/P
AA’	$in \rightarrow in$	A’
A’B’	$in \rightarrow in$	B’
B’C	$in \rightarrow in$	C
CD	$in \rightarrow out$	C’
DD’	$out \rightarrow in$	D”, D’
D’E’	$in \rightarrow in$	E’
E’A	$in \rightarrow in$	A

New vertex

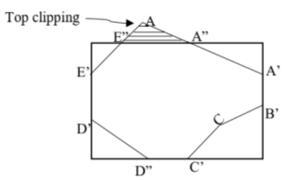


Top Clipping

vertex	case	O/P
AA’	$out \rightarrow in$	A”, A
A’B’	$in \rightarrow in$	B’
B’C	$in \rightarrow in$	C
CC’	$in \rightarrow in$	C’
C’D”	$in \rightarrow in$	D”
D”D’	$in \rightarrow in$	D’
D’E’	$in \rightarrow in$	E’
E’A	$in \rightarrow out$	E”

New vertex

New vertex



Hence, final polygon after clipping: