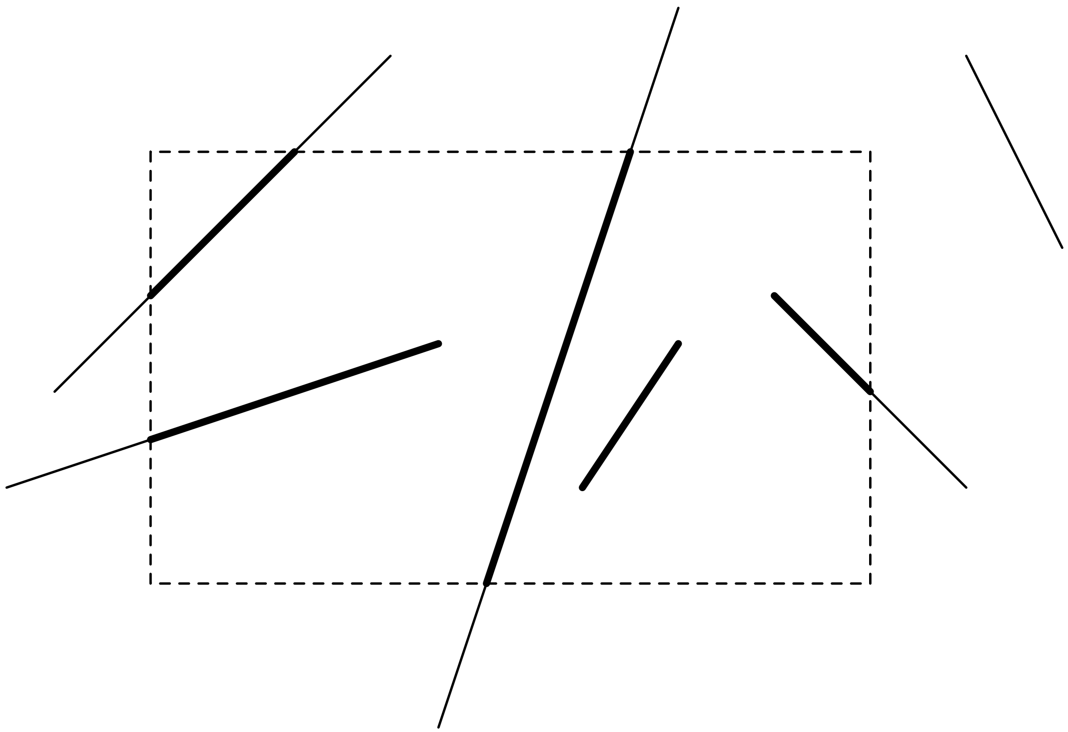
Clipping Lines and Polygons

Clipping involves replacing lines with shorter lines that exactly fit into the boundaries of the window. Clipping is performed in world coordinates.

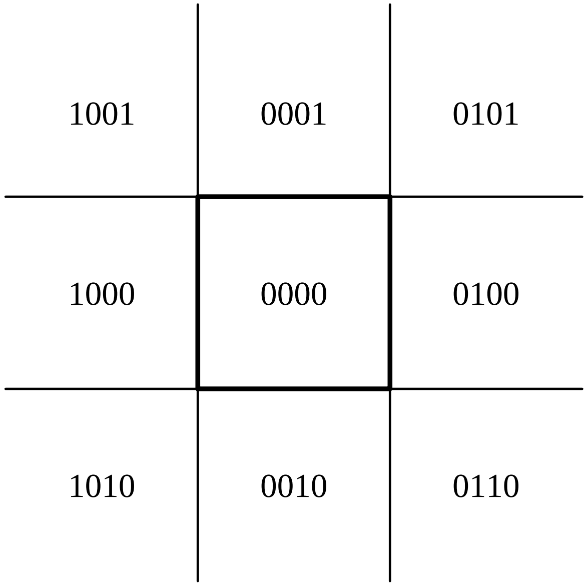


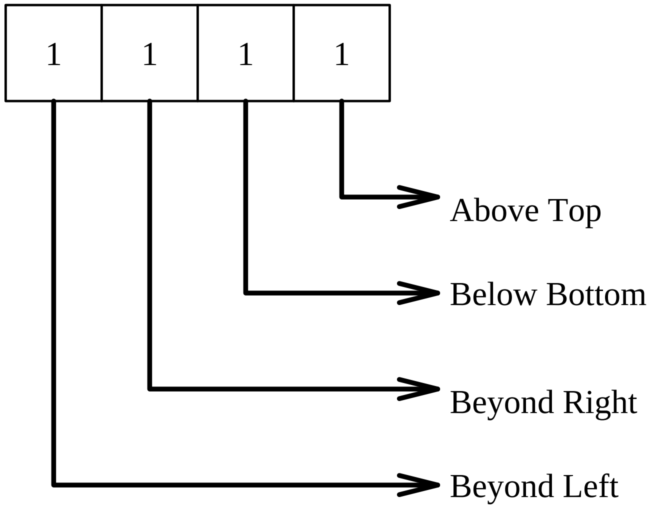
Cohen-Sutherland Line Clipping

- **Filter Out Simple Cases**

  Assign a binary code to each endpoint and perform simple tests.

- **Clip Against Each Side**





Filtering Simple Cases

Let $P1$ and $P2$ be the two endpoints of the line.

Encode $P1$ and $P2$ as binary strings according to which of the 9 regions they occupy.

If the logical AND of these codes is non-zero

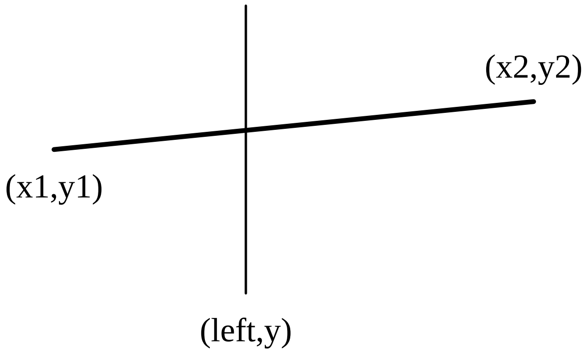
the line may be clipped completed.

If the logical OR of these codes is zero

the line is entirely inside the window, and may be accepted.

Else the line must be processed further.

Clipping Against Each Side



Suppose we want to clip against the `left` side. We must compute the y value where the line intersects the vertical window side $x = left$.

Parametric equation:

$$(x, y) = (x_1, y_1) + s(x_2 - x_1, y_2 - y_1)$$

Solve the x equation for s:

$$left = x_1 + s(x_2 - x_1)$$
$$\frac{left - x_1}{x_2 - x_1} = s$$

Clipping Against Each Side (2)

Now substitute this into the equation for y:

$$
\begin{aligned}
y &= y_1 + s(y_2 - y_1) \\
y &= y_1 + \frac{left - x_1}{x_2 - x_1}(y_2 - y_1)
\end{aligned}
$$

The other cases are similar.

Clipping Algorithm

```
if (rcode & 8) {                                  // bit 3 in region code
   x = l;
   y = y1 + (y2−y1)∗(l−x1)/(x2−x1);
} else if (rcode & 4) {                           // bit 2 in region code
   x = r;
   y = y1 + (y2−y1)∗(r−x1)/(x2−x1);
}
if (rcode & 2)  {                                 // bit 1 in region code
   x = x1 + (x2−x1)∗(b−y1)/(y2−y1);
   y = b;
} else if (rcode & 1) {                           // bit 0 in region code
   x = x1 + (x2−x1)∗(t−y1)/(y2−y1);
   y = t;
}
```

Clipping Example

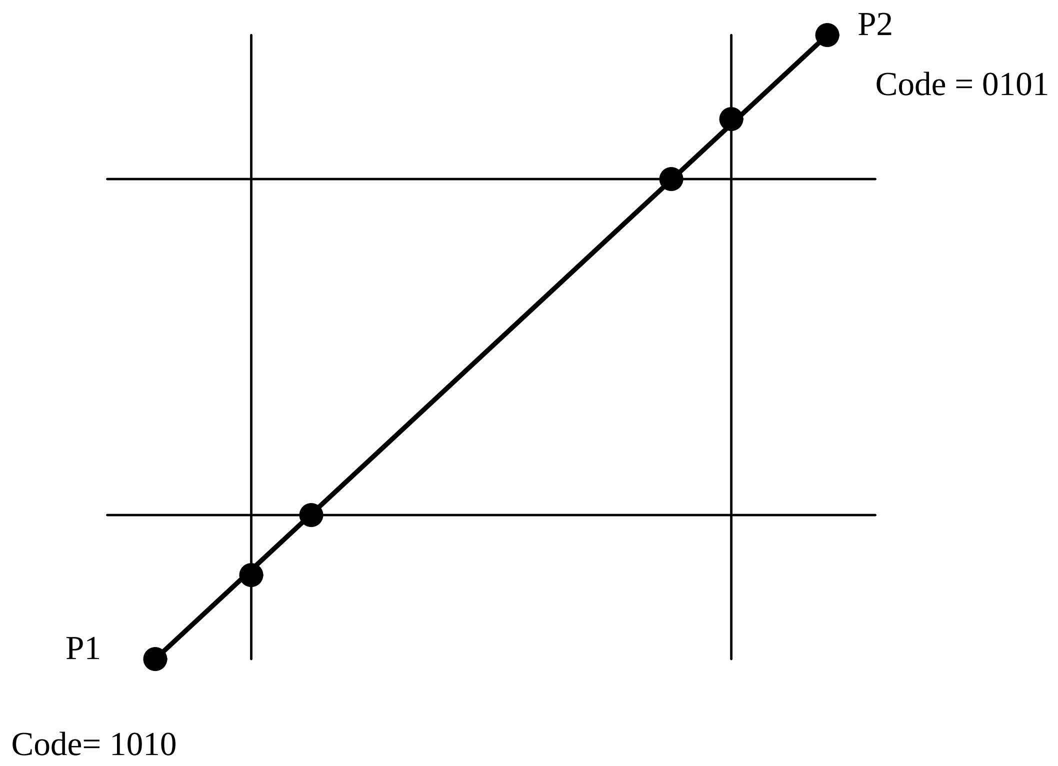


Figure 4.9

We make two passes. On the first pass, $P1$ is changed twice.
On the second pass, $P2$ is changed twice.

Polygon Clipping

Polygon clipping is much more than just line clipping. We must introduce new lines so that the polygon remains a polygon (or becomes multiple polygons).

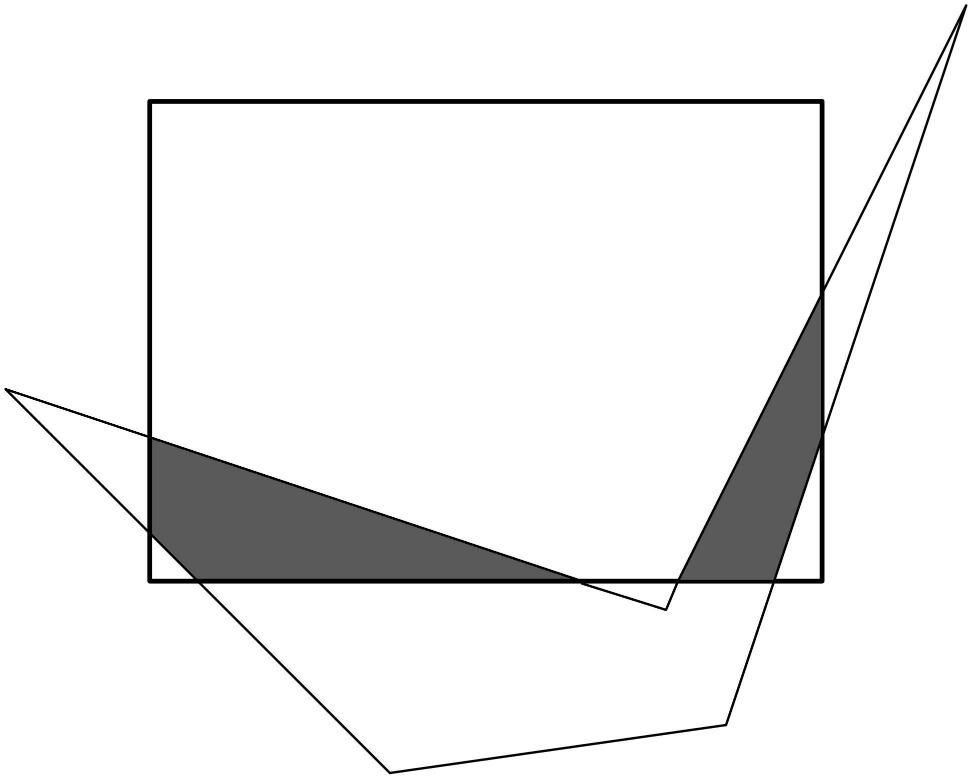
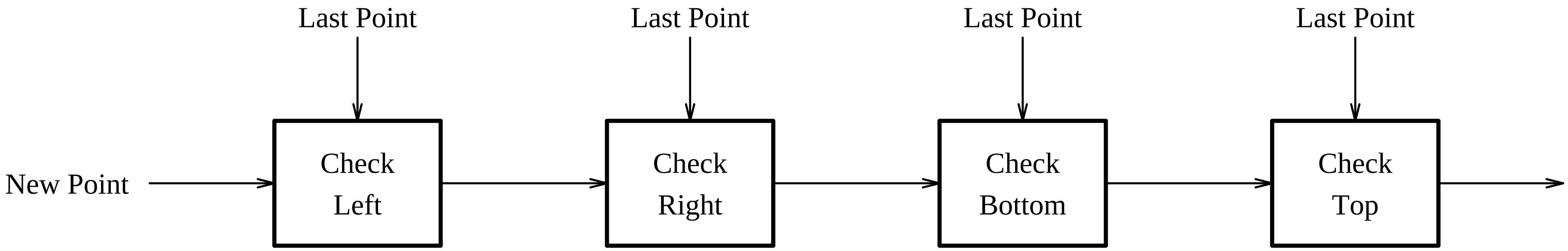


Figure 4.27

The Clipping Pipeline



Each vertex will be sent through a clipping pipeline.

Each box will clip the line from the *Last Point* to the current point.

The final polygon is constructed by connecting the vertexes (in order) that emerge from the pipeline.

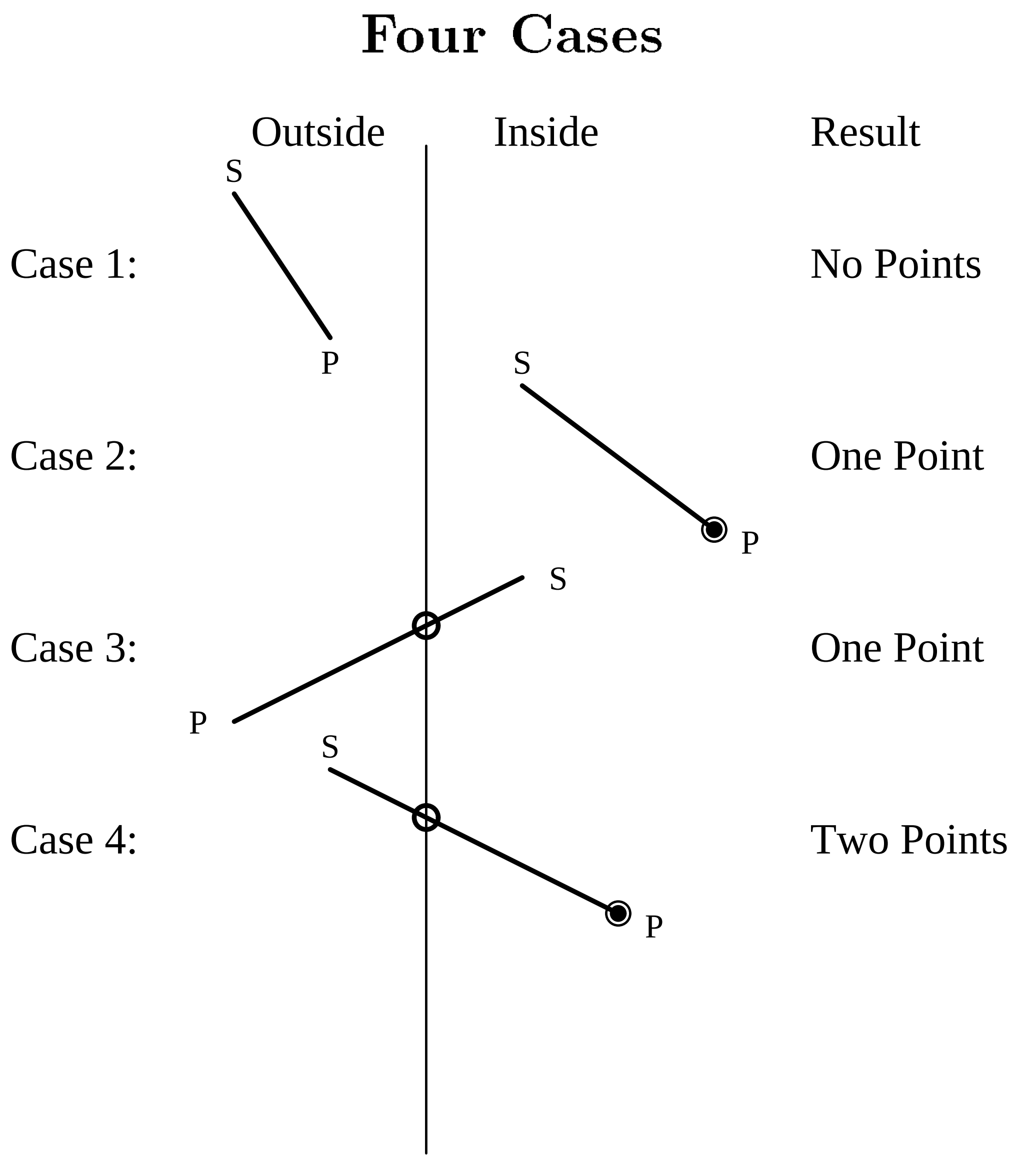


Figure 4.28-31

An Example

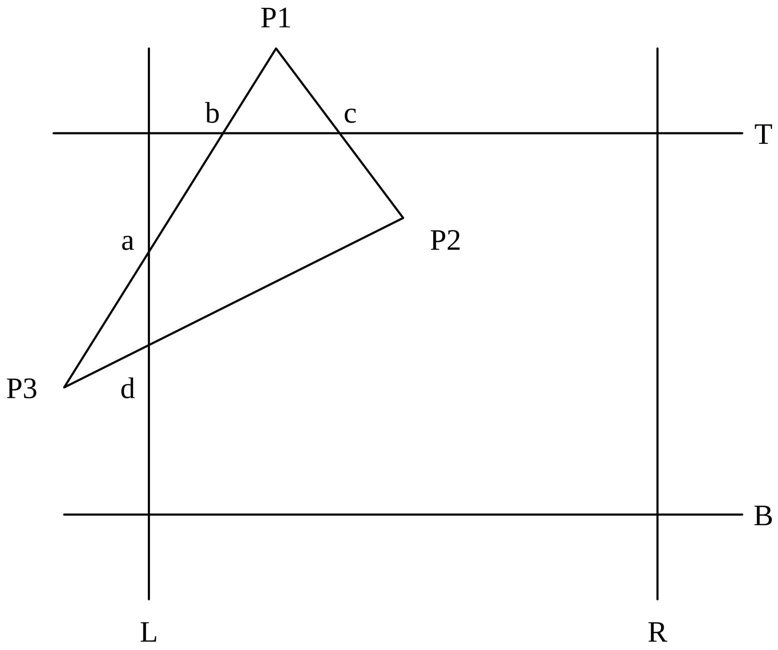
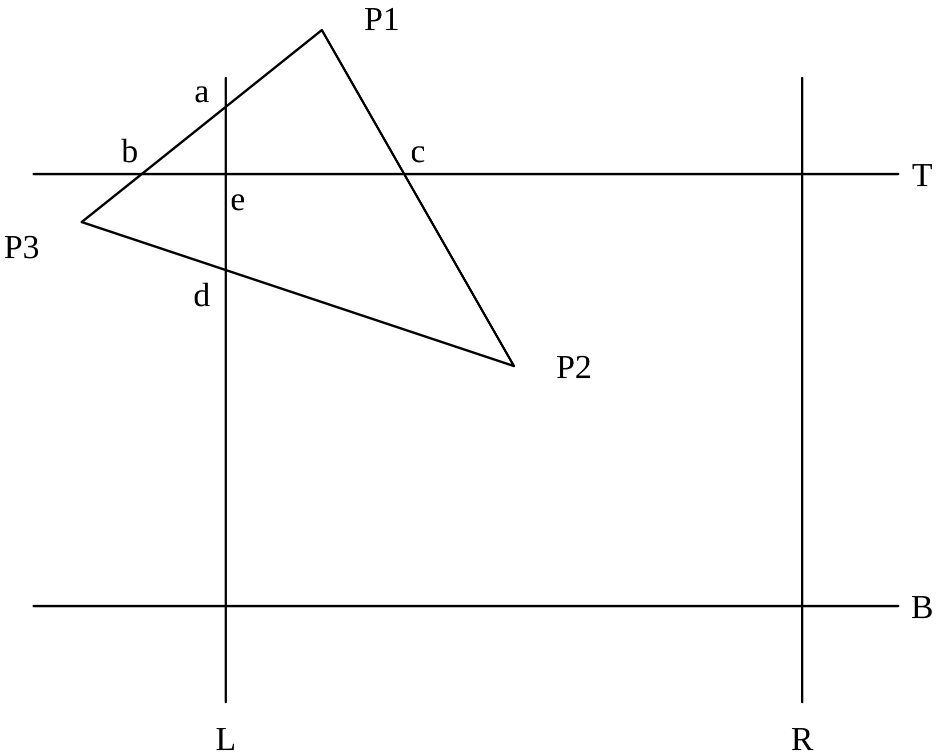


Figure 4.35

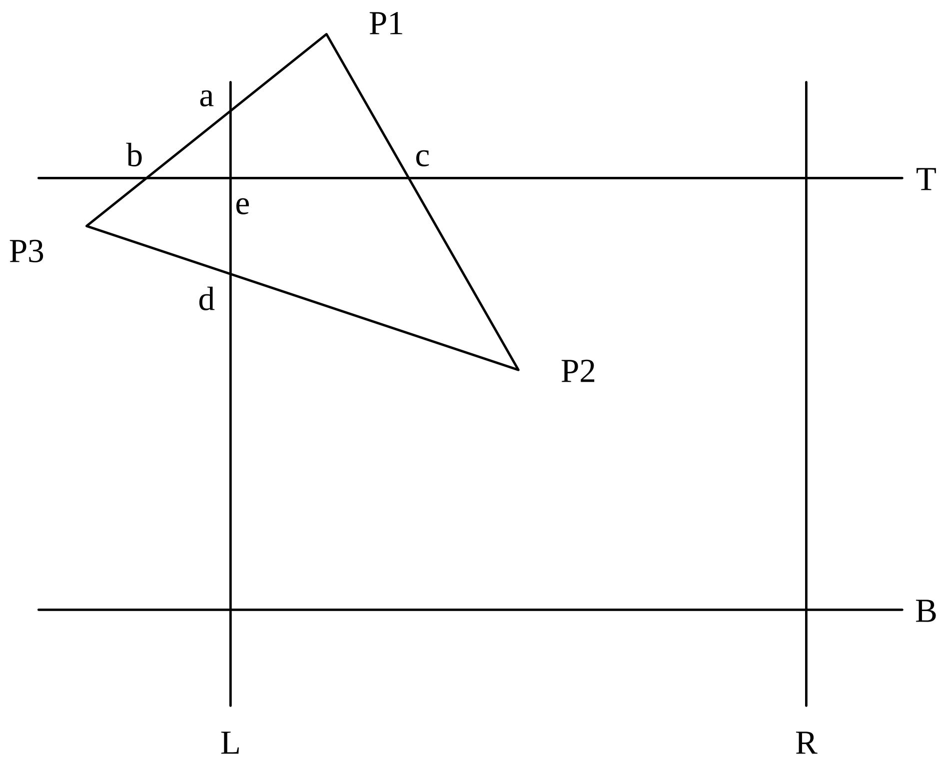
P_3		P_3		P_3		P_3		
P_1	L	a	R	a	B	a	T	a
		P_1	R	P_1	B	P_1	T	b
P_2	L	P_2	R	P_2	B	P_2	T	c
								P_2
P_3	L	d	R	d	d	d	T	d

Initializing the Pipe Properly



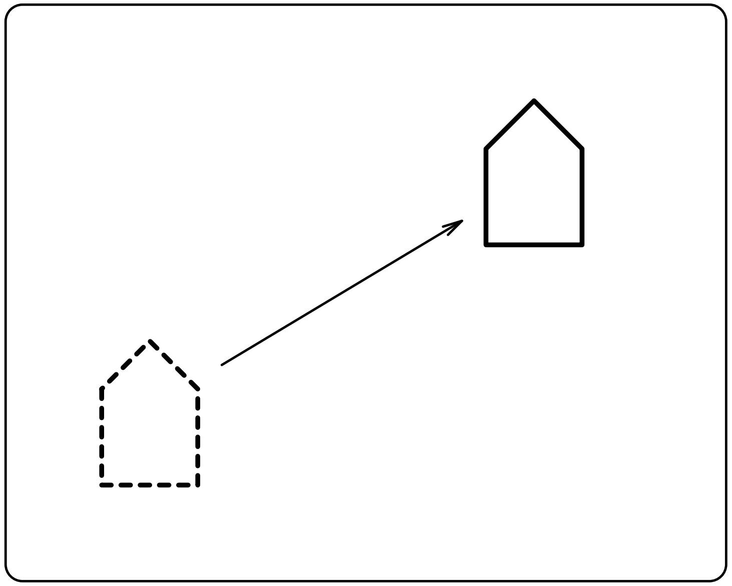
P_3		P_3		P_3		P_3		
P_1	L	a	R	a	B	a	T	b
		P_1	R	P_1	B	P_1	T	
P_2	L	P_2	R	P_2	B	P_2	T	c
								P_2
P_3	L	d	R	d	d	d	T	d

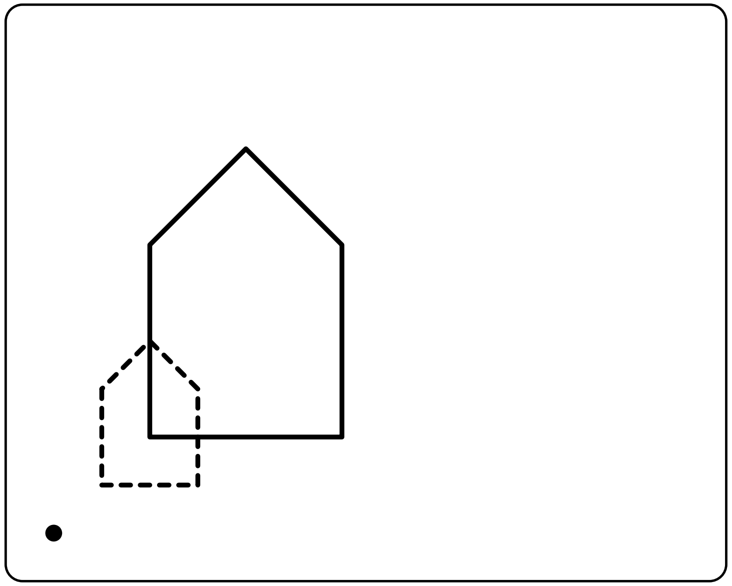
Second Pass

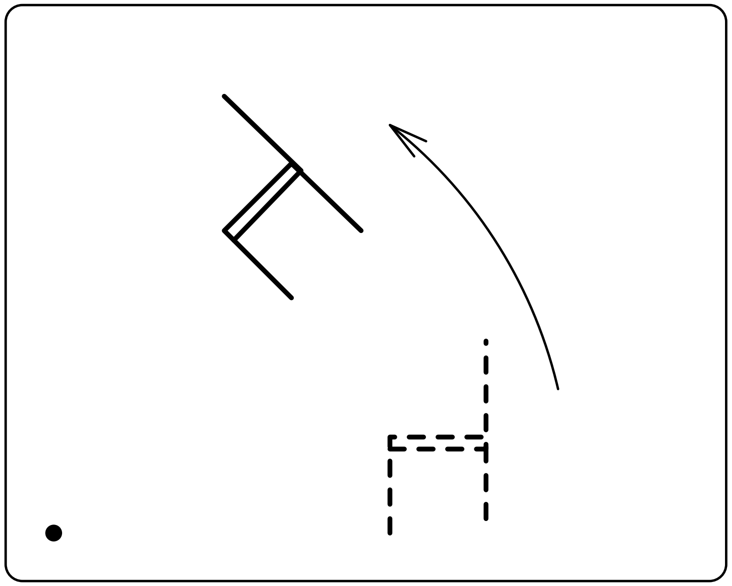


P_3		d		d		d		
P_1	L	a	R	a	B	a	T	e
		P_1	R	P_1	B	P_1	T	
P_2	L	P_2	R	P_2	B	P_2	T	c
								P_2
P_3	L	d	R	d	d	d	T	d

2D Transformations







Linearity

Each of these transformations converts a straight line into a straight line.

Hence, to transform polygons, we need only transform their vertexes.

It also turns out that all of these transformations can be implemented by matrix multiplication (i.e., linear transformations).

2D Translation

To translate a point $P = (x, y)$ by an amount dx in the X direction and dy in the Y direction, we can compute:

$$P_{new} = P + (dx, dy)$$

or

$$(x_{new}, y_{new}) = (x + dx, y + dy)$$

This is simple vector addition.

2D Scaling

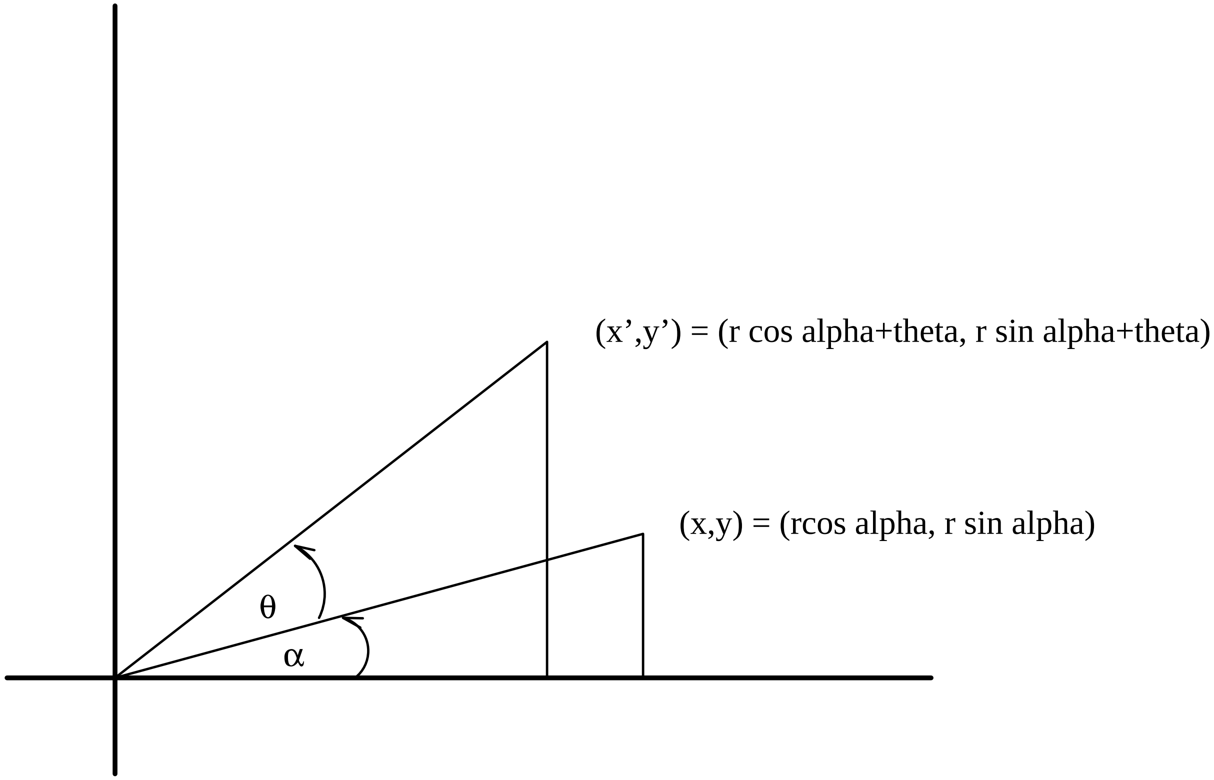
To scale a point $P = (x, y)$ by a scale factor S, we just multiply each coefficient by S:

$$(x_{new}, y_{new}) = (Sx, Sy).$$

This scales the point with respect to the origin. For example, if the point were originally $d = \sqrt{x^2 + y^2}$ away from the origin, it is now

$$Sd = \sqrt{(Sx)^2 + (Sy)^2} = \sqrt{S^2(x^2 + y^2)} = S\sqrt{x^2 + y^2}.$$

2D Rotation



Convert to polar coordinates:

$$(x, y) = (r\cos\alpha, r\sin\alpha)$$

$$(x', y') = (r\cos(\alpha+\theta), r\sin(\alpha+\theta))$$

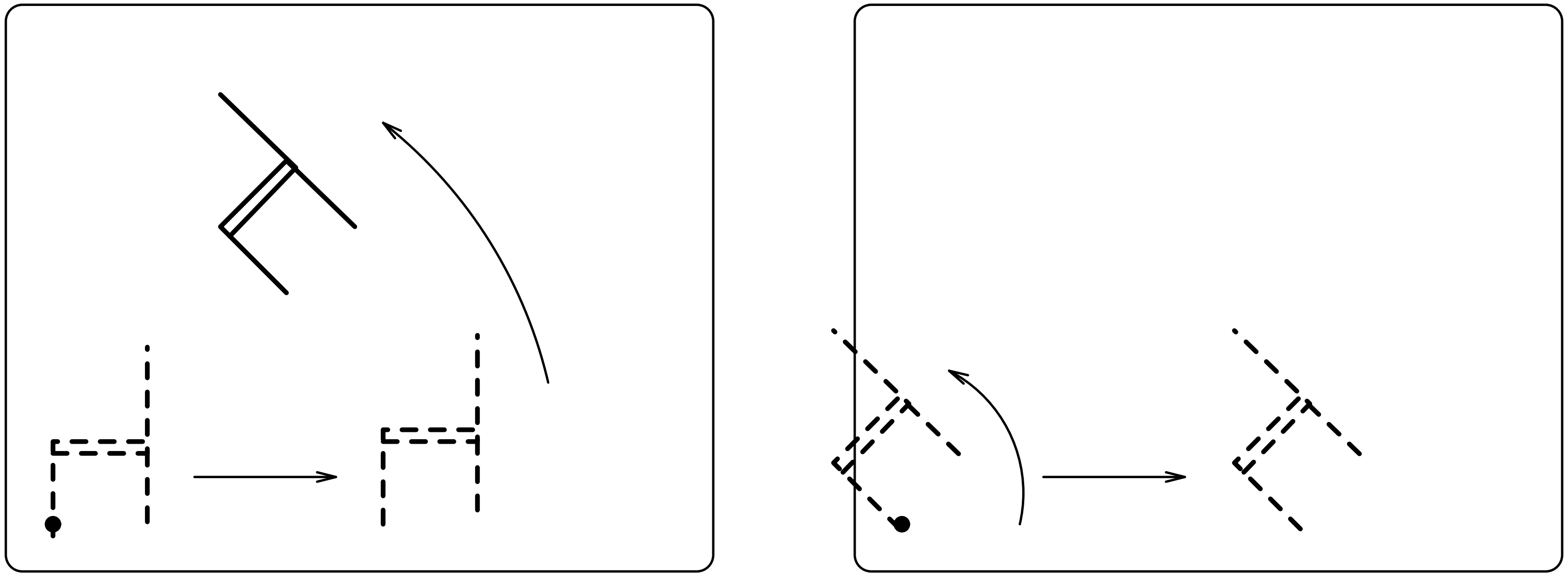
2D Rotation (2)

Apply double angle formulas:

$$\begin{aligned}
x' &= r\left[\cos\alpha\cos\theta - \sin\alpha\sin\theta\right] \\
&= r\left[\frac{x}{r}\cos\theta - \frac{y}{r}\sin\theta\right] \\
&= x\cos\theta - y\sin\theta
\end{aligned}$$

$$\begin{aligned}
y' &= r\left[\sin\alpha\cos\theta + \cos\alpha\sin\theta\right] \\
&= r\left[\frac{y}{r}\cos\theta + \frac{x}{r}\sin\theta\right] \\
&= y\cos\theta + x\sin\theta.
\end{aligned}$$

Transformations are *not* commutative



Translation followed by rotation $\neq$ rotation followed by translation.

Homogeneous Coordinates

All transformations can be expressed as matrix multiplication in homogeneous coordinates.

Homogeneous Coordinates for a point: $(x, y, 1)$.

Homogeneous Coordinates for translation:

$$(x', y', 1) = (x, y, 1) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{pmatrix}$$

Homogeneous Coordinates: Scaling and Rotation

Homogeneous Coordiantes for scaling:

$$(x', y', 1) = (x, y, 1)\begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Homogeneous Coordinates for rotation by θ:

$$(x', y', 1) = (x, y, 1)\begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Pre-computing Complex Transformations

Matrix multiplication is associative:

Suppose we want to scale by a factor of 2, translate by $(1, 5)$ and rotate by 45 degrees:

$$(x', y', 1) = (x, y, 1) \begin{pmatrix} 2.0 & 0 & 0 \\ 0 & 2.0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1.0 & 5.0 & 1 \end{pmatrix} \begin{pmatrix} 0.7071 & 0.7071 & 0 \\ -0.7071 & 0.7071 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$(x', y', 1) = (x, y, 1) \begin{pmatrix} 1.4142 & 1.4142 & 0 \\ -1.4142 & 1.4142 & 0 \\ -2.8284 & 4.2426 & 1 \end{pmatrix}$$

This lets us precompute the transformation as a single matrix and then apply it to all points in a complex image.