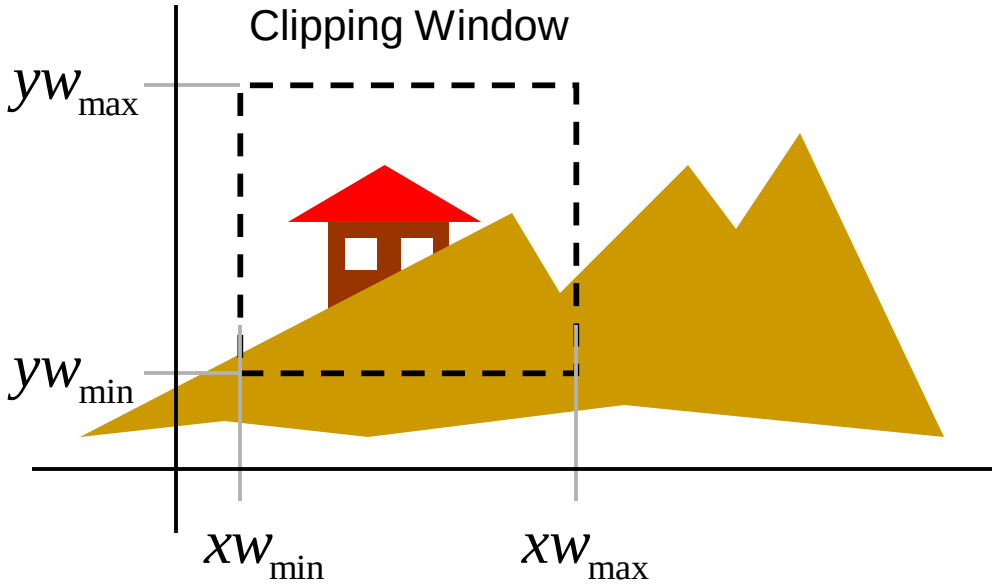
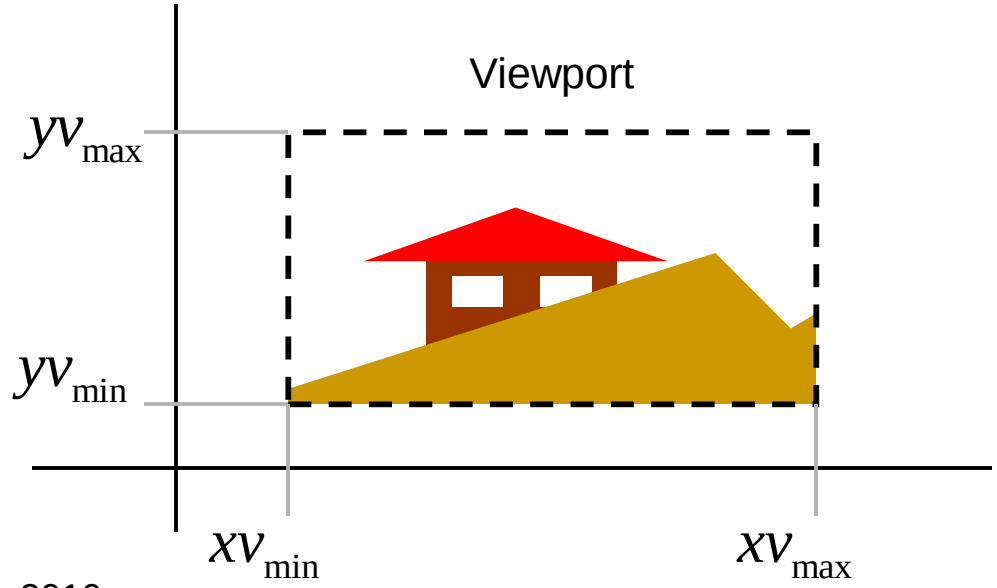
Computer Graphics Viewing

Shmuel Wimer
Bar Ilan Univ., School of Engineering



World Coordinates

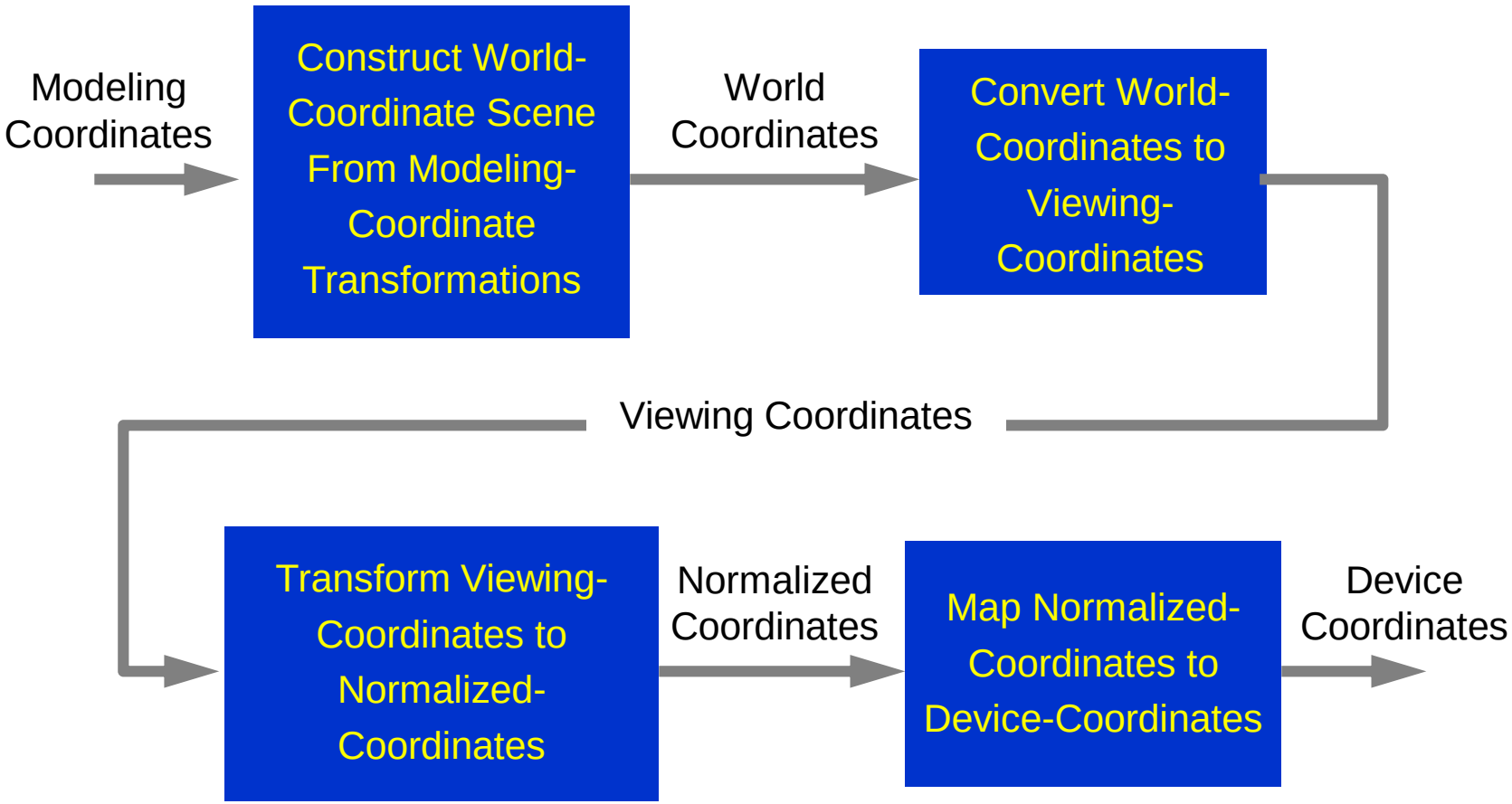
The clipping window is mapped into a viewport.



Viewing world has its own coordinates, which may be a non-uniform scaling of world coordinates.

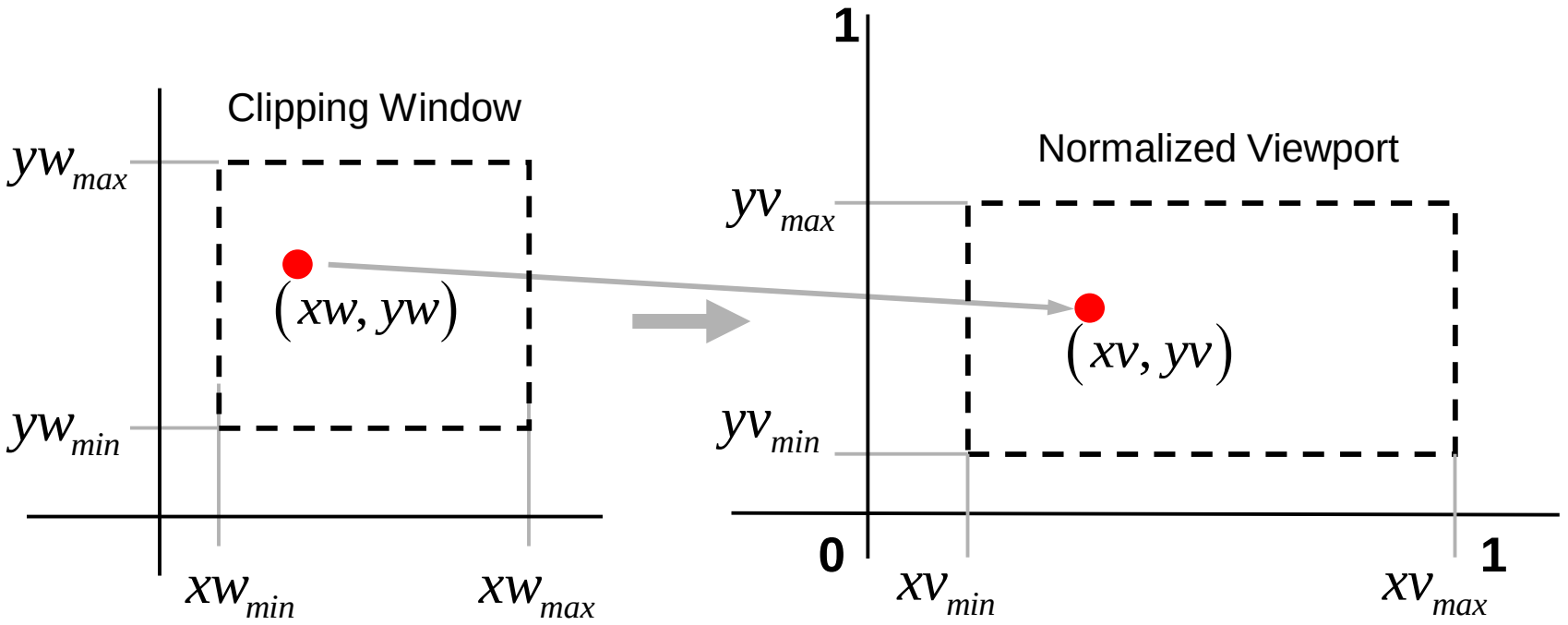
Viewport Coordinates

2D viewing transformation pipeline



Normalization and Viewport Transformations

- First approach:
  - Normalization and window-to-viewport transformations are combined into one operation.
  - Viewport range can be in [0,1] x [0,1].
  - Clipping takes place in [0,1] x [0,1].
  - Viewport is then mapped to display device.
- Second approach:
  - Normalization and clipping take place before viewport transformation.
  - Viewport coordinates are specified in screen coordinates.



Maintain relative size and position between clipping window and viewport.

$$\frac{xv - xv_{\min}}{xv_{\max} - xv_{\min}} = \frac{xw - xw_{\min}}{xw_{\max} - xw_{\min}} \qquad \frac{yv - yv_{\min}}{yv_{\max} - yv_{\min}} = \frac{yw - yw_{\min}}{yw_{\max} - yw_{\min}}$$

Solving for (xv, yv) obtains:

$xv = s_x xw + t_x, \quad yv = s_y yw + t_y,$ where

Scaling factors: $s_x = \dfrac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}}, \quad s_y = \dfrac{yv_{\max} - yv_{\min}}{yw_{\max} - yw_{\min}}$

Translation factors:

$$t_x = \frac{xw_{\max} xv_{\min} - xw_{\min} xv_{\max}}{xw_{\max} - xw_{\min}}, \quad t_y = \frac{yw_{\max} yv_{\min} - yw_{\min} yv_{\max}}{yw_{\max} - yw_{\min}}$$

This can also be obtained by composing transformations:

$\mathbf{M}_{\substack{\text{window,} \\ \text{norm\_viewport}}} =$

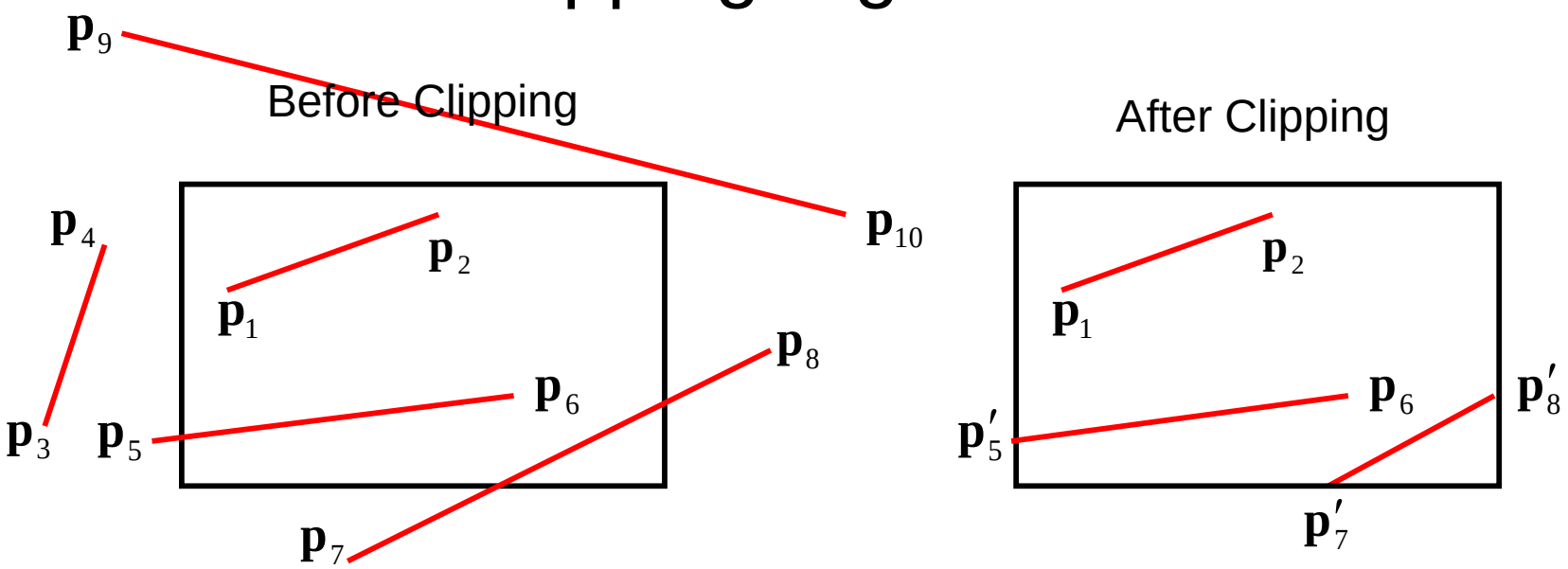
$$\mathbf{T}(xv_{\min}, yv_{\min}) \cdot \mathbf{S}(s_x, s_y) \cdot \mathbf{T}(-xw_{\min}, -yw_{\min}) = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

World clipping window can first be mapped to normalized square between -1 and +1, where clipping algorithm takes place, and then transform the scene into viewport given in display coordinates.

$$\mathbf{M}_{\substack{\text{window,}\\ \text{norm\_square}}} = \begin{bmatrix} \frac{2}{xw_{\max} - xw_{\min}} & 0 & -\frac{xw_{\max} + xw_{\min}}{xw_{\max} - xw_{\min}} \\ 0 & \frac{2}{yw_{\max} - yw_{\min}} & -\frac{yw_{\max} + yw_{\min}}{yw_{\max} - yw_{\min}} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M}_{\substack{\text{norm\_square,}\\ \text{viewport}}} = \begin{bmatrix} \left(xv_{\max} - xv_{\min}\right)/2 & 0 & \left(xv_{\max} + xv_{\min}\right)/2 \\ 0 & \left(yv_{\max} - yv_{\min}\right)/2 & \left(yv_{\max} + yv_{\min}\right)/2 \\ 0 & 0 & 1 \end{bmatrix}$$

Clipping Algorithms



Parametric equations of line segment from (x_0, y_0) to $(x_{\text{end}}, y_{\text{end}})$

$$x = x_0 + u(x_{\text{end}} - x_0), \quad y = y_0 + u(y_{\text{end}} - y_0), \quad 0 \le u \le 1.$$

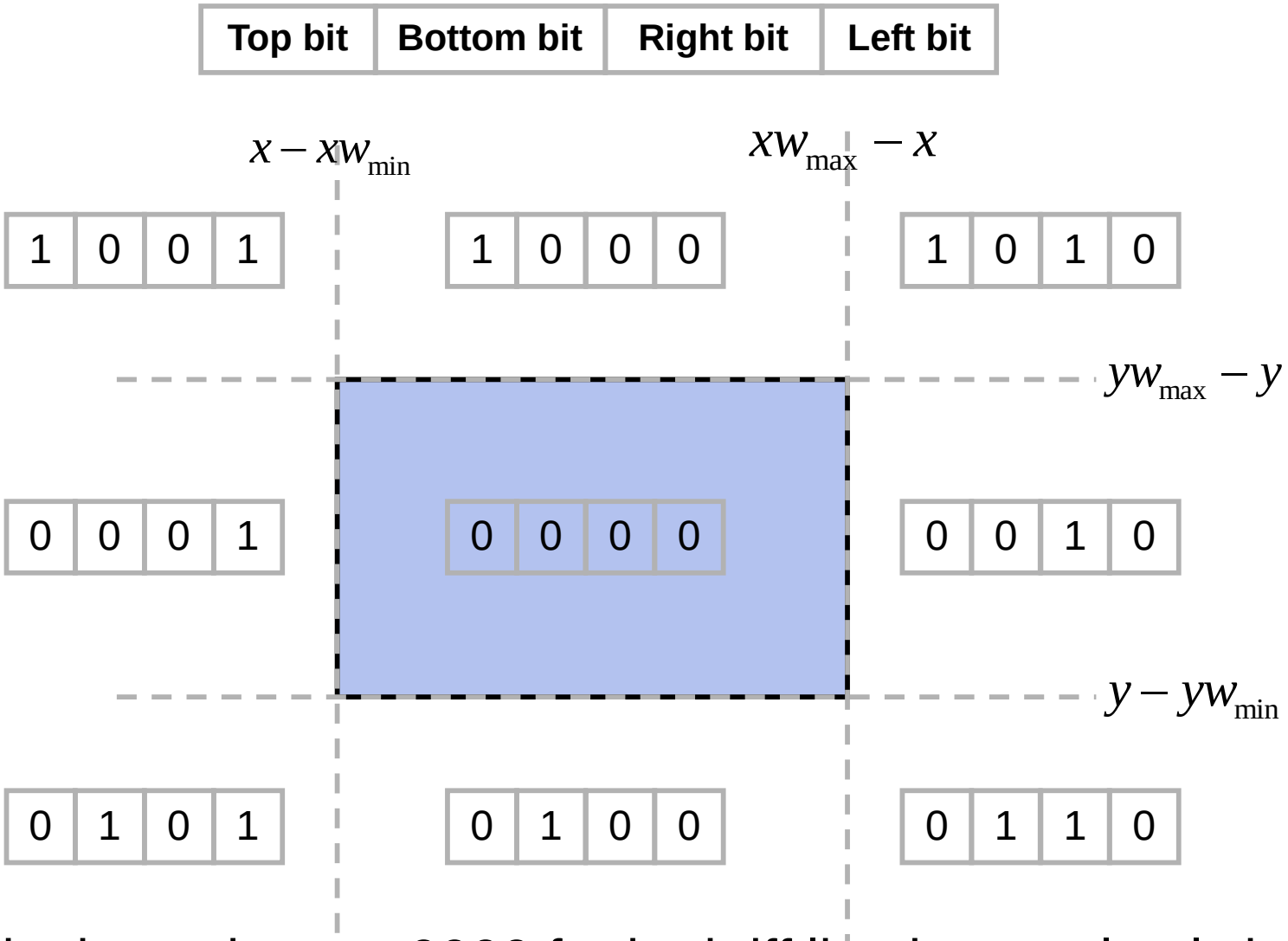
Used to determine the parts contained in clipping window.

Cohen-Sutherland Line Clipping Algorithm

- Intersection calculations are expensive. Find first lines completely inside or certainly outside clipping window. Apply intersection only to undecided lines.
- Perform cheaper tests before proceeding to expensive intersection calculations.

Cohen-Sutherland Line Clipping Algorithm

- Assign code to every endpoint of line segment.
  - Borderlines of clipping window divide the plane into two halves.
  - A point can be characterized by a 4-bit code according to its location in half planes.
  - Location bit is 0 if the point is in the positive half plane, 1 otherwise.
  - Code assignment involves comparisons or subtractions.
- Completely inside / certainly outside tests involve only logic operations of bits.



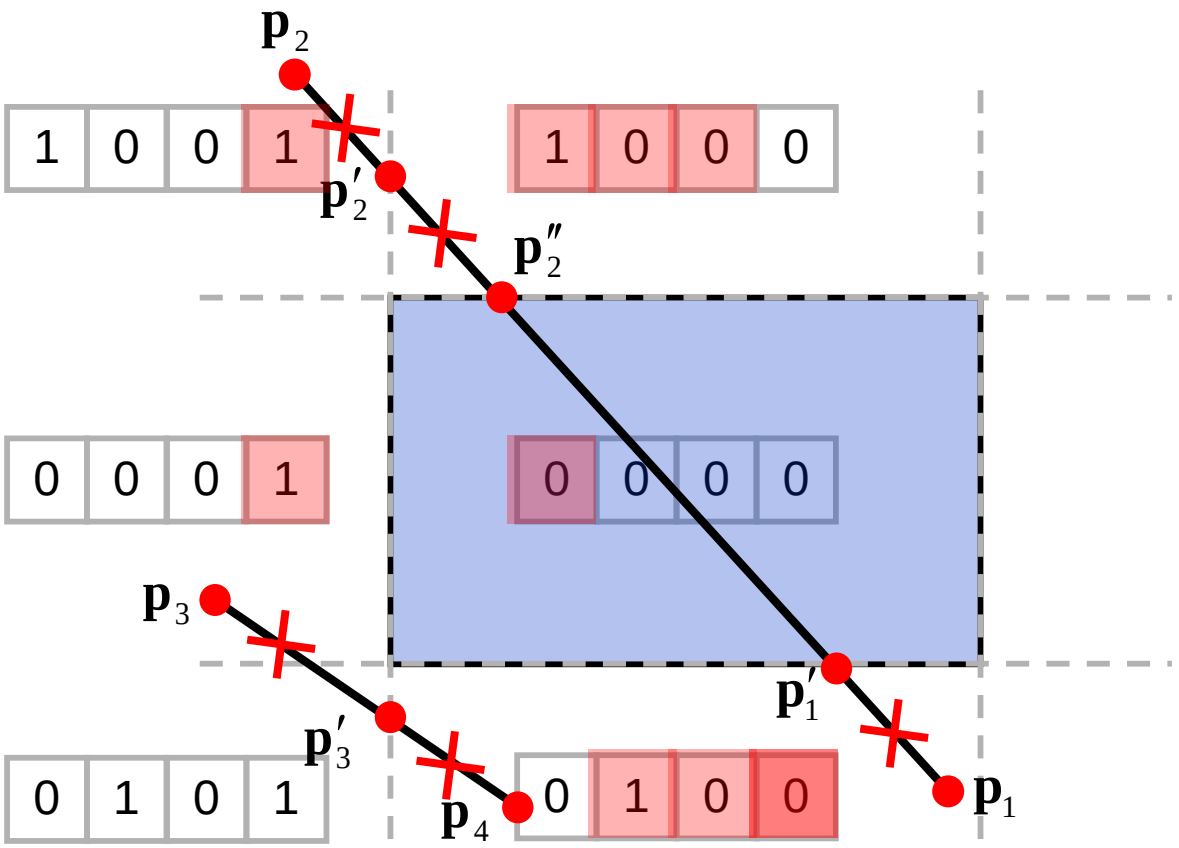
Endpoint codes are 0000 for both iff line is completely inside.

If endpoint codes has 1 in same bit, line is certainly outside.

Lines that cannot be decided are intersected with window border lines.

Each test clips the line and the remaining is tested again for full inclusion or certain exclusion, until remaining is either empty or fully contained.

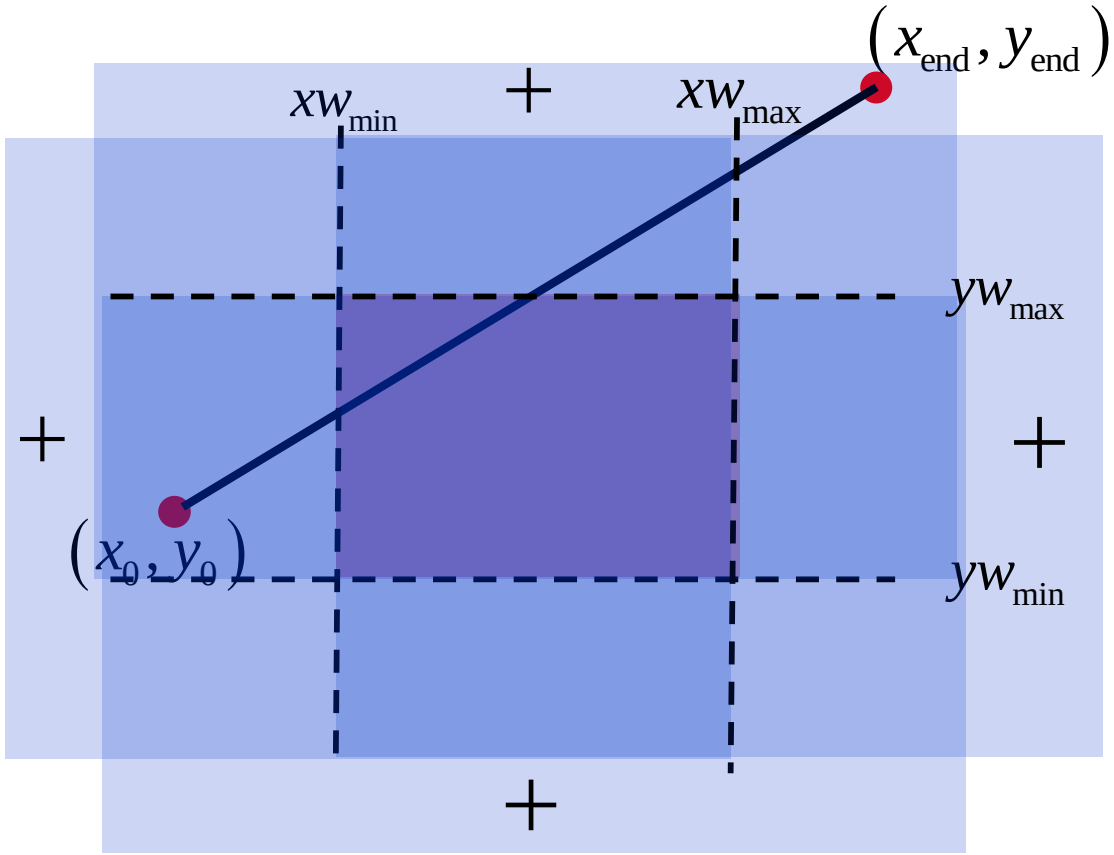
Endpoints of lines are examined against left, right, bottom and top borders (can be any order).


$\mathbf{p}_2$
1 0 0 1
1 0 0 0
$\mathbf{p}'_2$
$\mathbf{p}''_2$
0 0 0 1
0 0 0 0
$\mathbf{p}_3$
$\mathbf{p}'_1$
$\mathbf{p}'_3$
0 1 0 1
0 1 0 0
$\mathbf{p}_4$
$\mathbf{p}_1$

Liang-Barsky Line Clipping Algorithm

Treat undecided lines in Cohen-Sutherland more efficiently.

Define clipping window by intersections of four half-planes.



Parametric presentation:

$$x = x_0 + u\left(x_{\text{end}} - x_0\right),\ \ y = y_0 + u\left(y_{\text{end}} - y_0\right),\ \ 0 \le u \le 1.$$

A point on the line is cotained in the clipping window iff:

$$xw_{\min} \le x_0 + u\left(x_{\text{end}} - x_0\right) \le xw_{\max},$$

$$yw_{\min} \le y_0 + u\left(y_{\text{end}} - y_0\right) \le yw_{\max}.$$

It can be expressed by: $up_k \le q_k,\ k = 1,2,3,4,$ where

$$p_1 = x_0 - x_{\text{end}},\ \ q_1 = x_0 - xw_{\min}\ ;$$

$$p_2 = x_{\text{end}} - x_0,\ \ q_2 = xw_{\max} - x_0.$$

$$p_3 = y_0 - y_{\text{end}},\ \ q_3 = y_0 - yw_{\min}\ ;$$

$$p_4 = y_{\text{end}} - y_0,\ \ q_4 = yw_{\max} - y_0.$$

In the inequality $up_k \le q_k$ if $p_k < 0$ $(p_k > 0)$, the traversal from (x_0, y_0) to $(x_{\text{end}}, y_{\text{end}})$ by increasing u from $-\infty$ to $+\infty$ proceeds the line from the $-(+)$ half-plane to $+(-)$ one (with respect to the k-th border).

Intersection of $(-\infty, +\infty)$ extension with k-th border occurs at $u = q_k / p_k$.

We calculate and update u_0 and u_{end} progressively for $k = 1, 2, 3, 4$ borders (left, right, bottom, top).

If $p_k < 0$ u_0 is calculated since progression is from $-$ to $+$ half planes. Similarly, if $p_k > 0$ u_{end} is calculated.

u_0 is the maximum among 0 and all q_k / p_k . u_{end} is the minimum among 1 and all q_k / p_k . The feasibility condition $u_{\text{end}} > u_0$ is progressively checked. The line is completely outside if $u_{\text{end}} < u_0$.

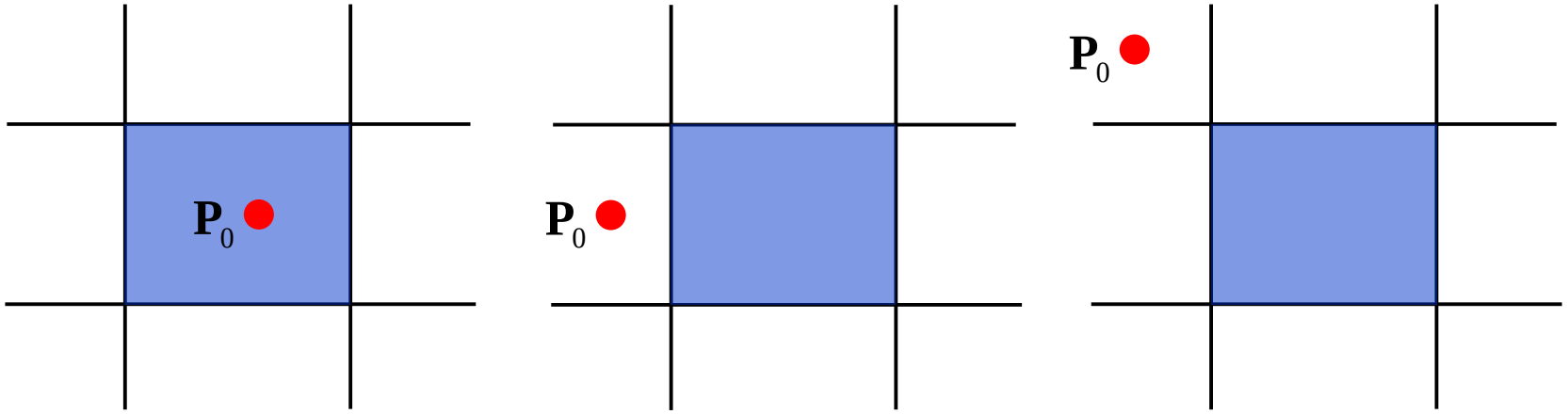
Notice that q_k / p_k doesn't need actual division since comparison of q'/p' with q''/p'' can be done by comparison of $q'p''$ with $q''p'$, and the quotient q/p can be stored as a pair (q, p)

Only if $u_{\text{end}} > u_0$, given by (q_0, p_0) and $(q_{\text{end}}, p_{\text{end}})$, the actual ends of the clipped portion are calculated.

This is more efficient than Cohen-Sutherland Alg, which computes intersection with clipping window borders for each undecided line, as a part of the feasibility tests.

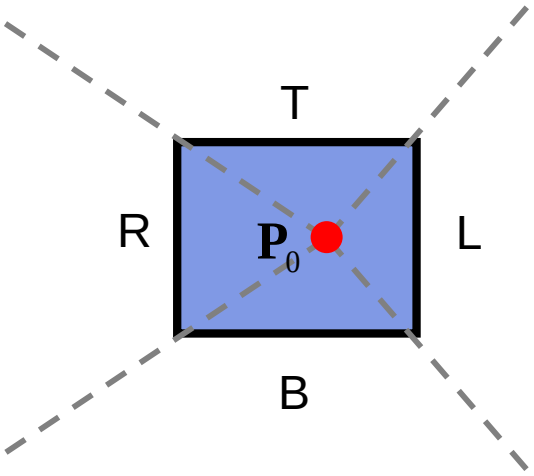
Nicholl-Lee-Nicholl Line Clipping Algorithm

- Creates more regions around clipping window to avoid multiple line intersection calculations.
- Performs fewer comparisons and divisions than Cohen-Sutherland and Liang-Barsky, but cannot be extended to 3D, while they can.
- For complete inclusion in clipping window or certain exclusion we'll use Cohen-Sutherland.

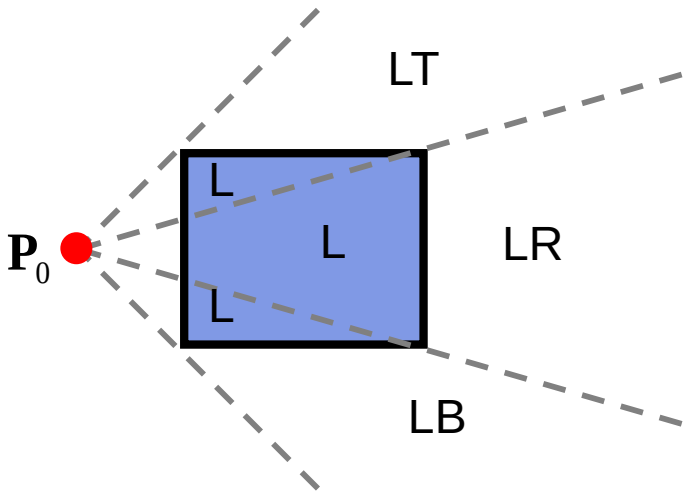


Examine first where the starting point $\mathbf{P}_0$ is located.

Only three regionsare considered. Location in any of the other six regions can be handled by symmety transformation.

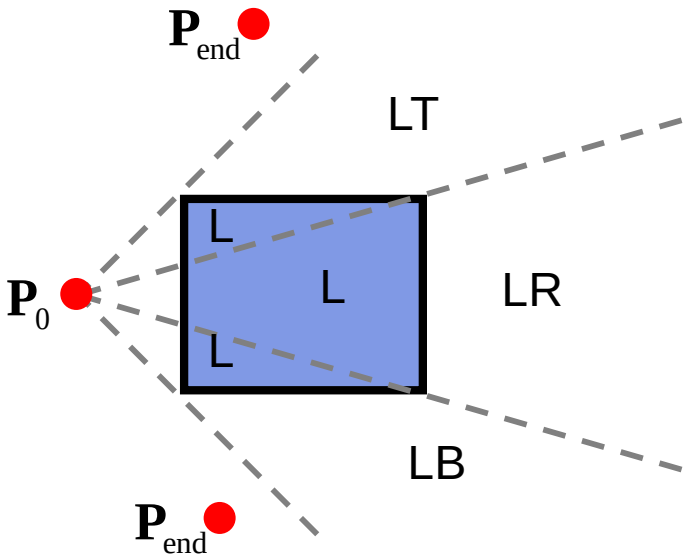


The location of $\mathbf{P}_{\text{end}}$ in each region defines what edge the line $[\mathbf{P}_0, \mathbf{P}_{\text{end}}]$ is intersecting.



Detecting whether $\mathbf{P}_{end}$ is in any of regions L is immediate.

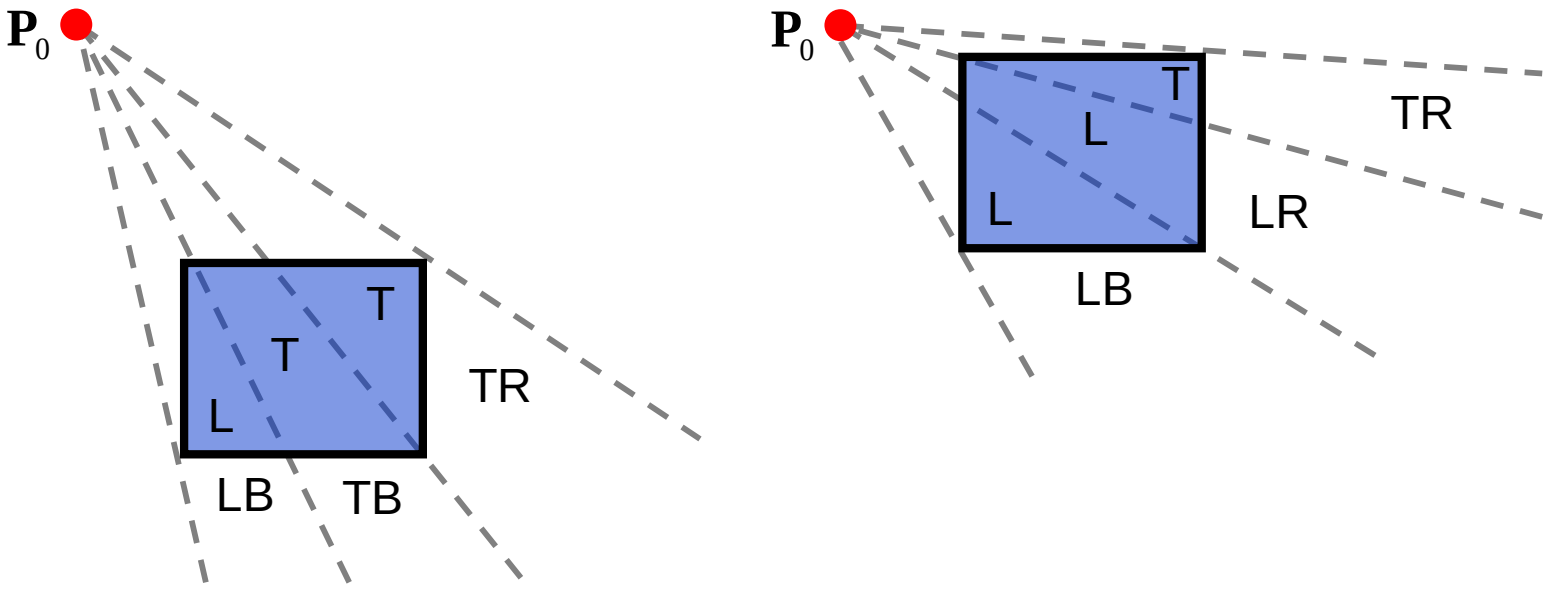
Else, $\mathbf{P}_{end}$ is detected for being positioned in any of LB, LR or LT, case where $[\mathbf{P}_0, \mathbf{P}_{end}]$ is clipped with left border and bottom, right or top border, resp.



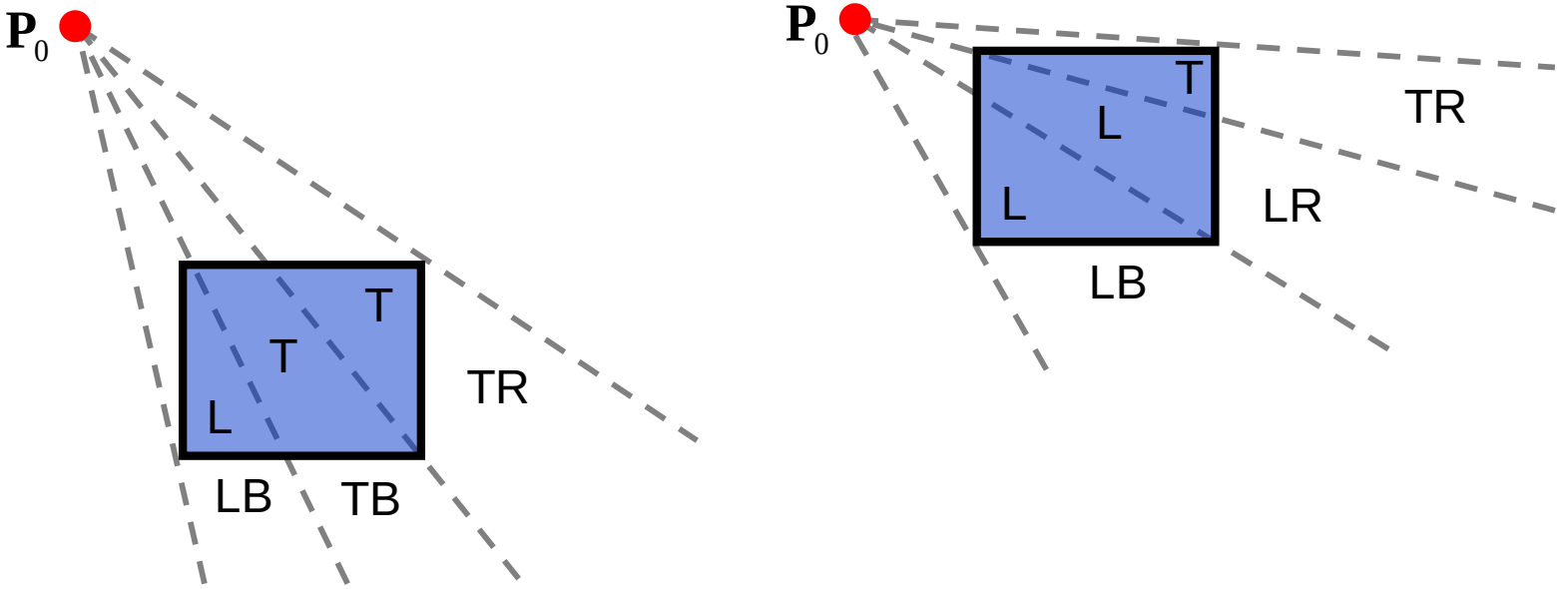
The slope of $[\mathbf{P}_0, \mathbf{P}_{end}]$ is compared to $[\mathbf{P}_0, \mathbf{P}_{corner}]$ for each corner to find the region of $\mathbf{P}_{end}$.

Once one of LT, LR or LB regions is found, intersection point with appropriate border is calculated.

$[\mathbf{P}_0, \mathbf{P}_{end}]$ is entierely clipped if $\mathbf{P}_{end}$ is positioned outside the regions.

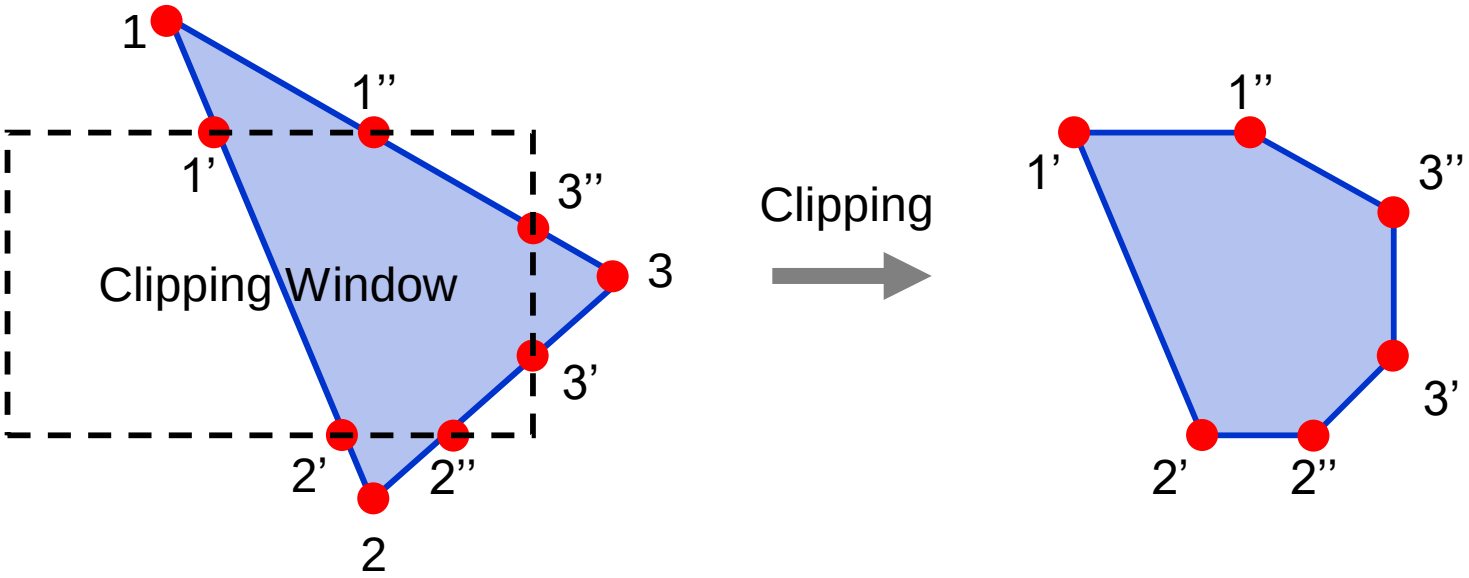


There are two cases, depending on whether $\mathbf{P}_0$ is closer to left or top borders.



Notice that simple inclusion test of P_{end} in clipping rectangle is not enough since there are both T and L labels for the regions inside. Testing of the angular regions is a must.

Sutherland-Hodgman Polygon Clipping

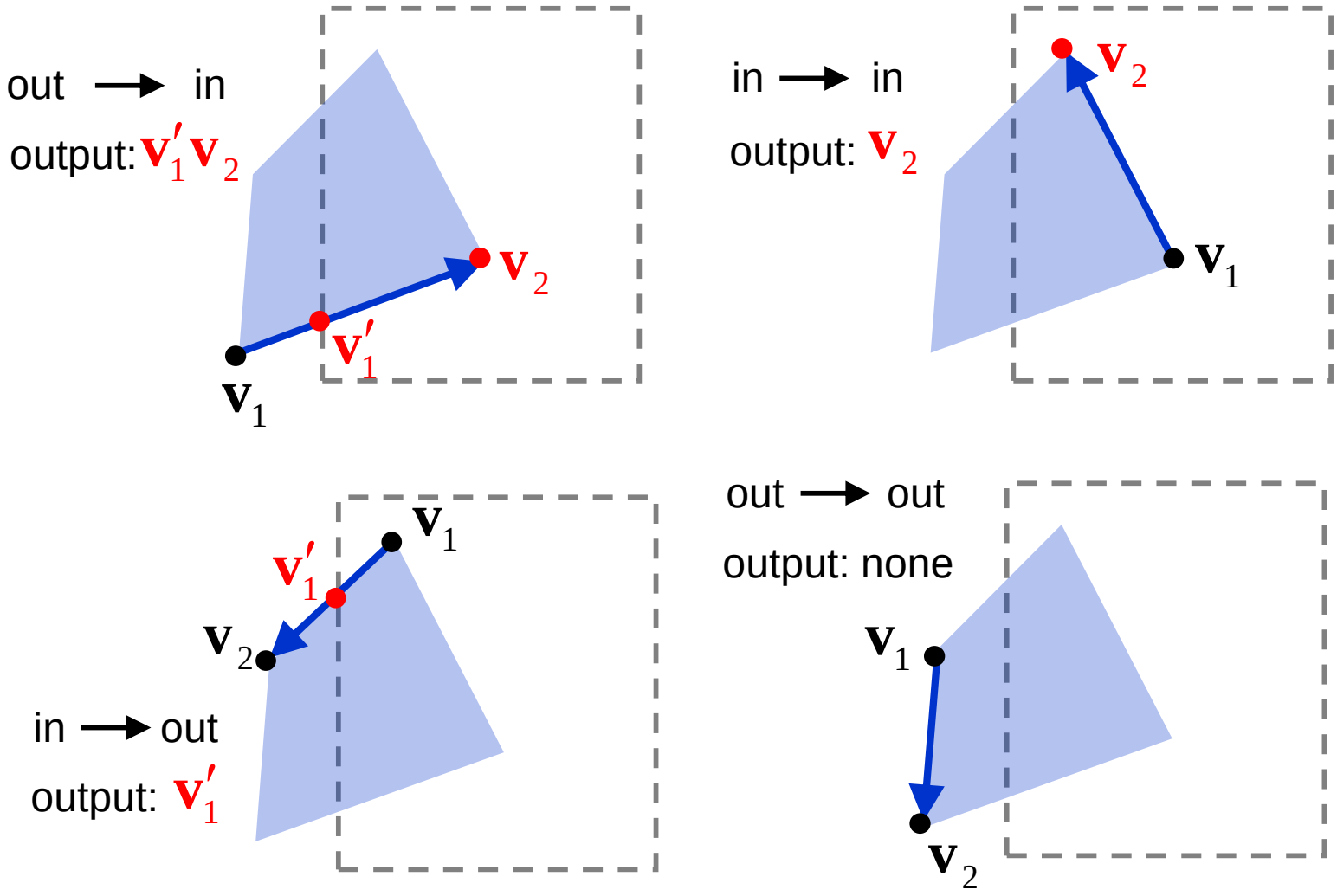


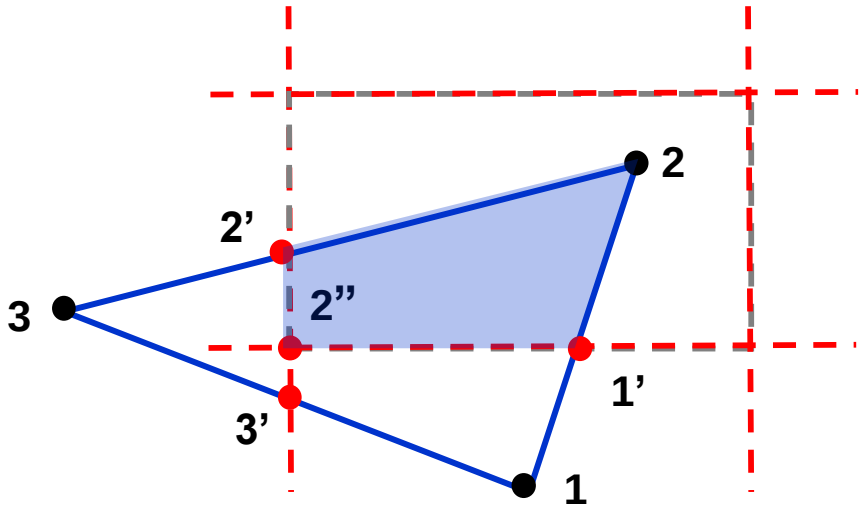
Efficient algorithm for clipping convex polygons.

Edges are clipped against every border line of clipping window. Edges are processed successively.

Allows pipelining of edge clipping of polygons, as well as pipelining of different polygons.

The four possible outputs generated by the left clipper, depending on the relative position of pair of edge endpoints.

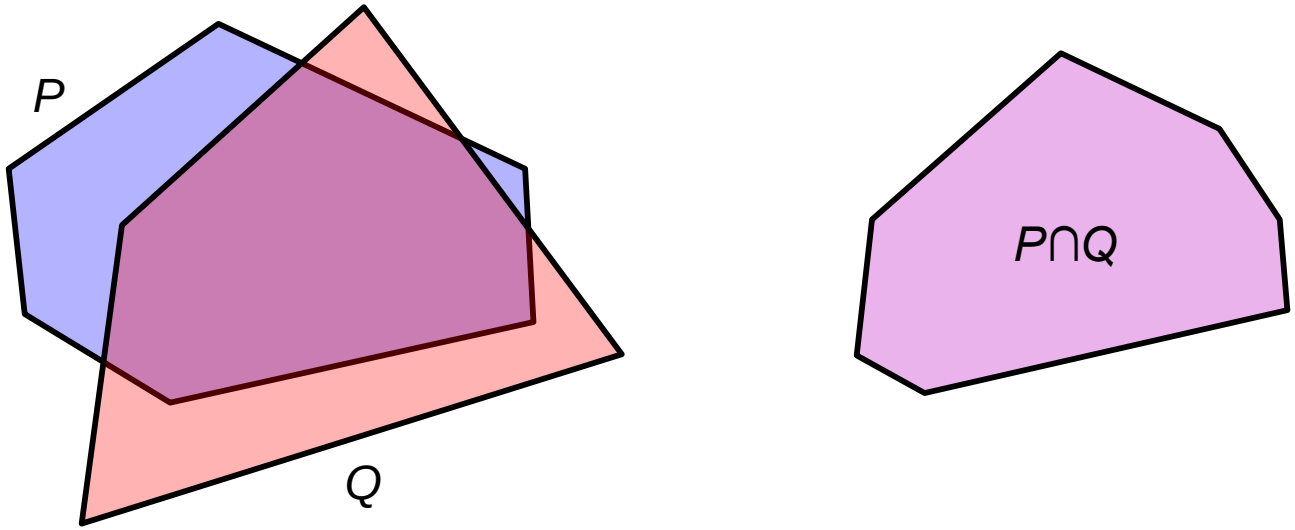




Input	Left Clipper	Right Clipper	Bottom Clipper	Top Clipper
[1,2]:	(in-in)>{2}			
[2,3]:	(in-out)>{2’}	[2,2’]:(in-in)>{2’}		
[3,1]:	(out-in)>{3’,1}	[2’,3’]:(in-in)>{3’}	[2’,3’]:(in-out)>{2”}	
		[3’,1]:(in-in)>{1}	[3’,1]:(out-out)>{}	
		[1,2]:(in-in)>{2}	[1,2]:(out-in)>{1’,2}	[2”,1’]:(in-in)>1’}
			[2,2’]:(in-in)>{2’}	[1’,2]:(in-in)>{2}
				[2,2’]:(in-in)>{2’}
				[2’,2”]:(in-in)>{2”}

- The four clippers can work in parallel.
  - Once a pair of endpoints it output by the first clipper, the second clipper can start working.
  - The more edges in a polygon, the more effective parallelism is.
- Processing of a new polygon can start once first clipper finished processing.
  - No need to wait for polygon completion.

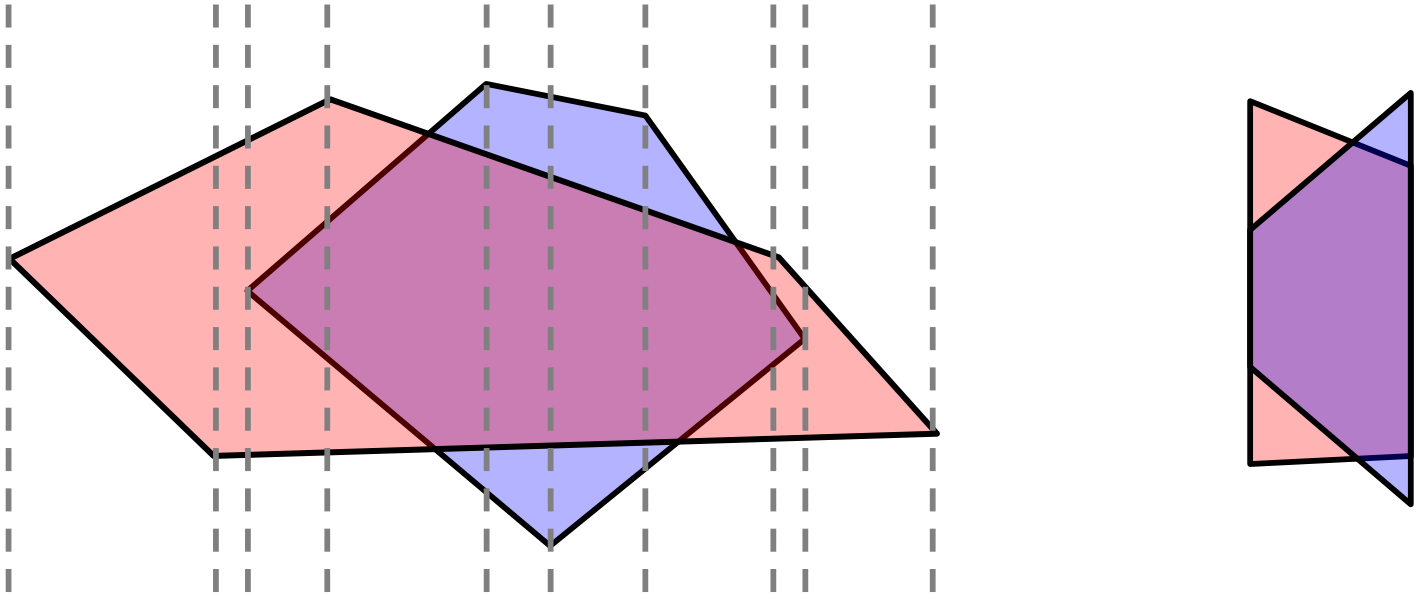
Convex Polygon Intersection



Theorem: The intersection of an *L*-vertex convex polygon and an *M*-vertex convex polygon is a convex polygon of *L+M* vertices at most.

Proof: *P∩Q* is an intersection of *L+M* interior half planes determined by the two polygons.

Intersection of convex polygons can answer the question of whether two sets of points can be separated by a line.

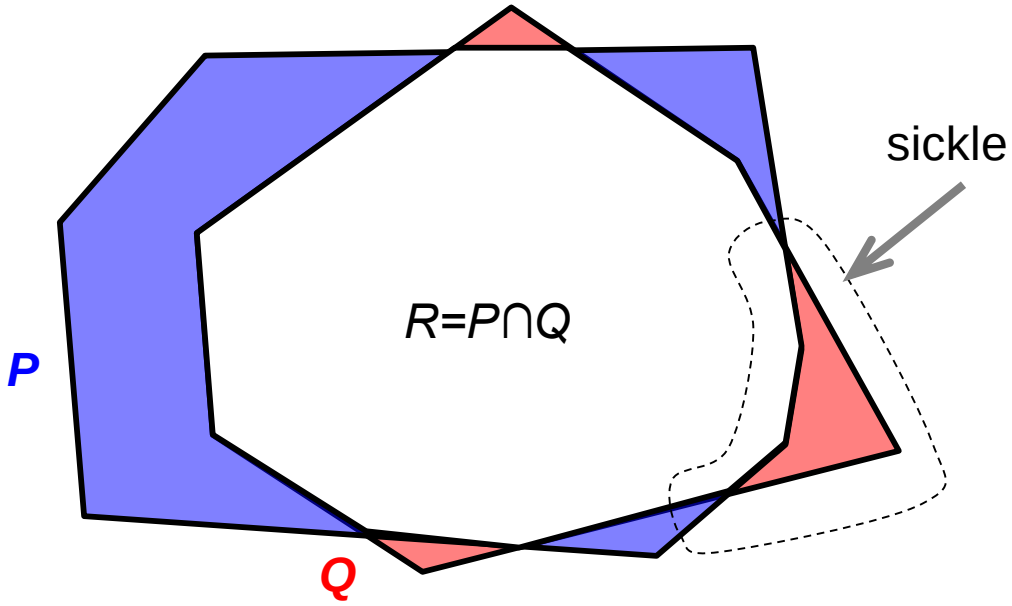


Theorem: The intersection of an L-vertex convex polygon and an M-vertex convex polygon can be found in $\theta(L+M)$ time.

Proof: Polygons are given in cyclic order of vertices. We start from the leftmost vertex of both polygons and progress along the border of both in a left-to-right fashion, defining $O(L+M)$ slabs.

Inside a slab each polygon forms a trapezoid. The intersection of two trapezoids can be calculated in constant time.

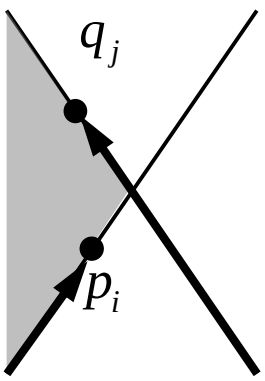
Another solution: The non empty intersection of two convex polygons forms a sequence of "sickles" enclosing the intersection. The border of a sickle comprises internal and external sequence of vertices originating from *P* and *Q*, which are alternating in every sickle.



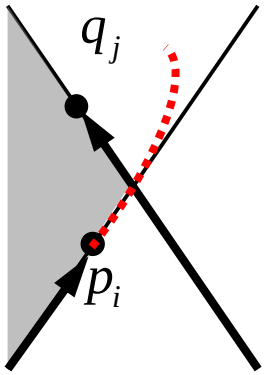
Let P: $(p_1,p_2,\ldots,p_L)$ and Q: $(q_1,q_2,\ldots,q_M)$ be counterclockwise cyclically ordered. The algorithm advances such that ∂P and ∂Q are "chasing" one another, adjusting their speeds so that they meet at every intersection.

Let p_i and q_j be the vertices where the traversal is. $\overline{p_{i-1}p_i}$ and $\overline{q_{j-1}q_j}$ are the current edges, defining the half planes $h(p_i)$ and $h(q_j)$, containing P and Q, respectively.

Clearly $R \subset h(p_i) \cap h(q_j)$.

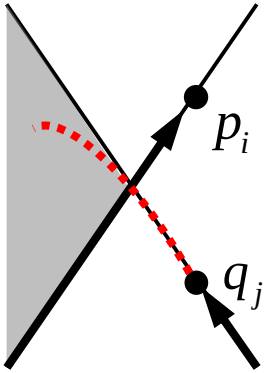


There are four possible situations of p_i and q_j with respect to R. For each an advancing rule is in order. The idea is to progress along boundary of the "lagging" polygon.



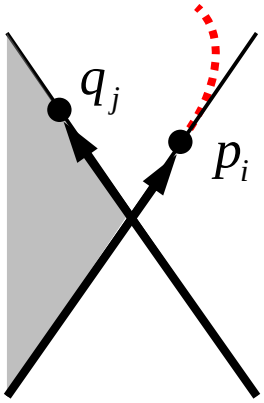
(a)

Advance is from p_i since there's no chance for a future intersection point in the edge $\overline{p_{i-1}p_i}$. Edge $\overline{q_{j-1}q_j}$ may have a future intersection with the boundary of P.

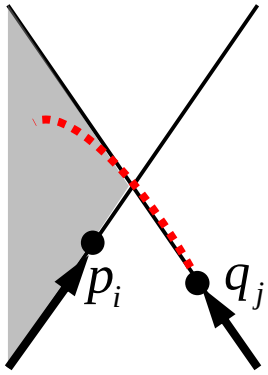


(b)

Edge $\overline{p_{i-1}p_i}$ may still be intersected later by Q, while edge $\overline{q_{j-1}q_j}$ have already exahusted its intersections. Hence advance is from q_j.

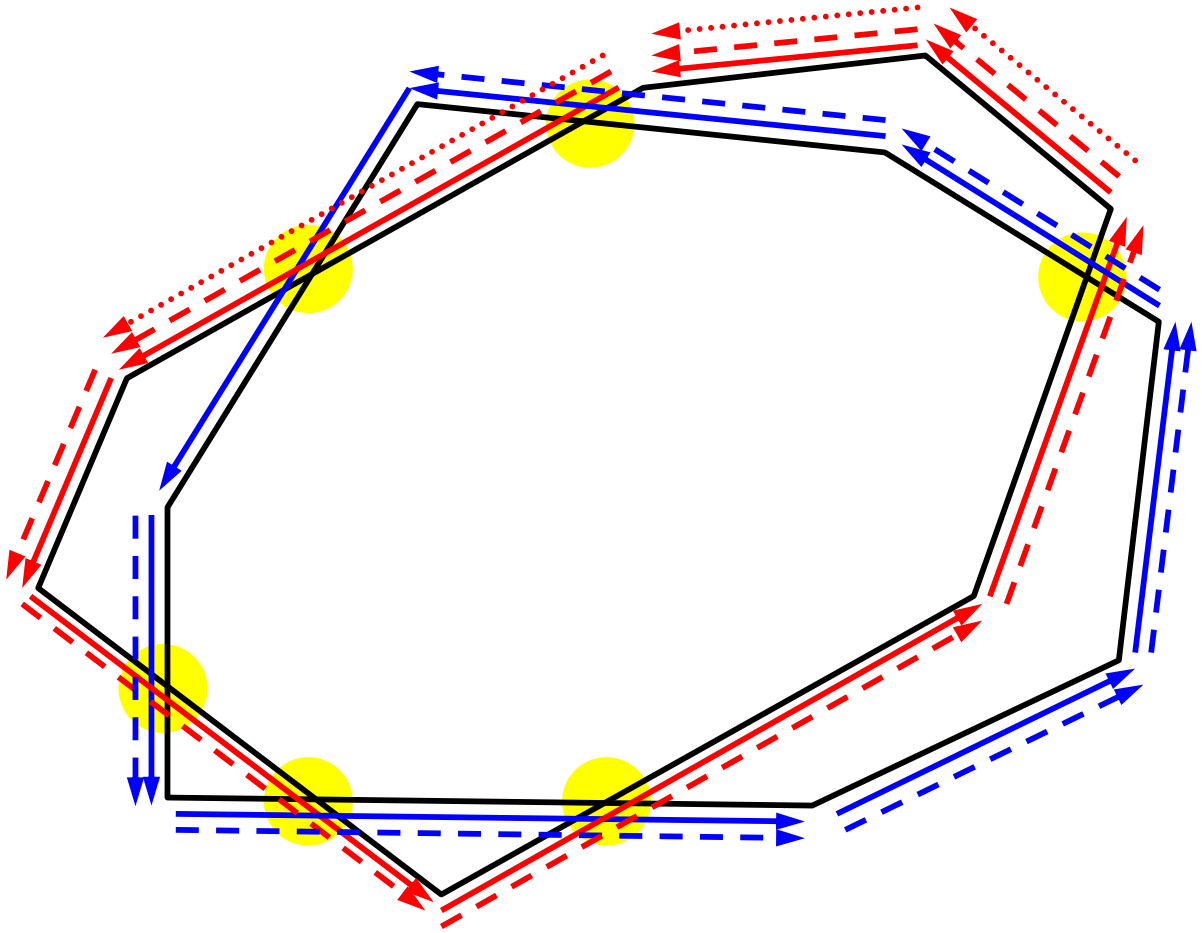


(c) Edge $\overline{q_{j-1}q_j}$ may still be intersected later by P, while edge $\overline{p_{i-1}p_i}$ have already exahusted its intersections. Hence advance is from p_i.



(d) Arbitrary choice. Advancing from q_j yields case (a), while advancing from p_i yields case (b).

```
void CONVEX_POLYGON_INTERSECT (polygon P, polygon Q) {
  i = 1; j = 1;  k = 1; // initialization
  while ( k < 2(L + M) ) {
    if ( p_{i-1}p_i and q_{j-1}q_j intersect) { report intersection }
    ADVANCE; // apply one of (a), (b), (c) or (d) cases
    ++k;
  }
  if (no intersection reported) {
    if ( p_i ∈ Q ) { P ⊆ Q }
    elseif ( q_j ∈ P ) { Q ⊆ P }
    else { Q ⋂ P = ∅ }
  }
}
```

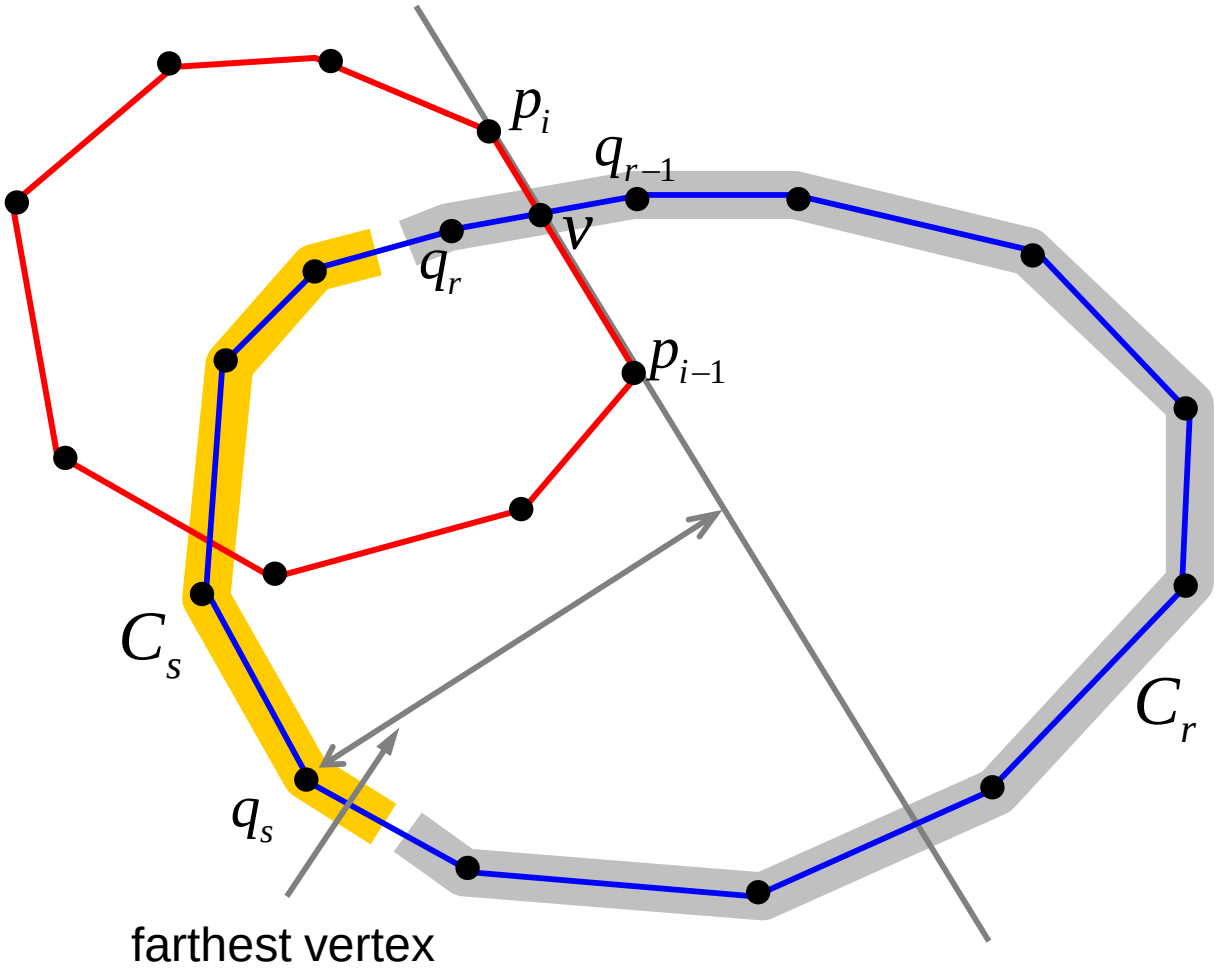


Correcness of algorithm : If p_i and q_j belong to the same sickle, the intersection point terminating the sickle must be found since algorithm never ADVANCE along the polygon whose current edge may contain a sought intersection point.

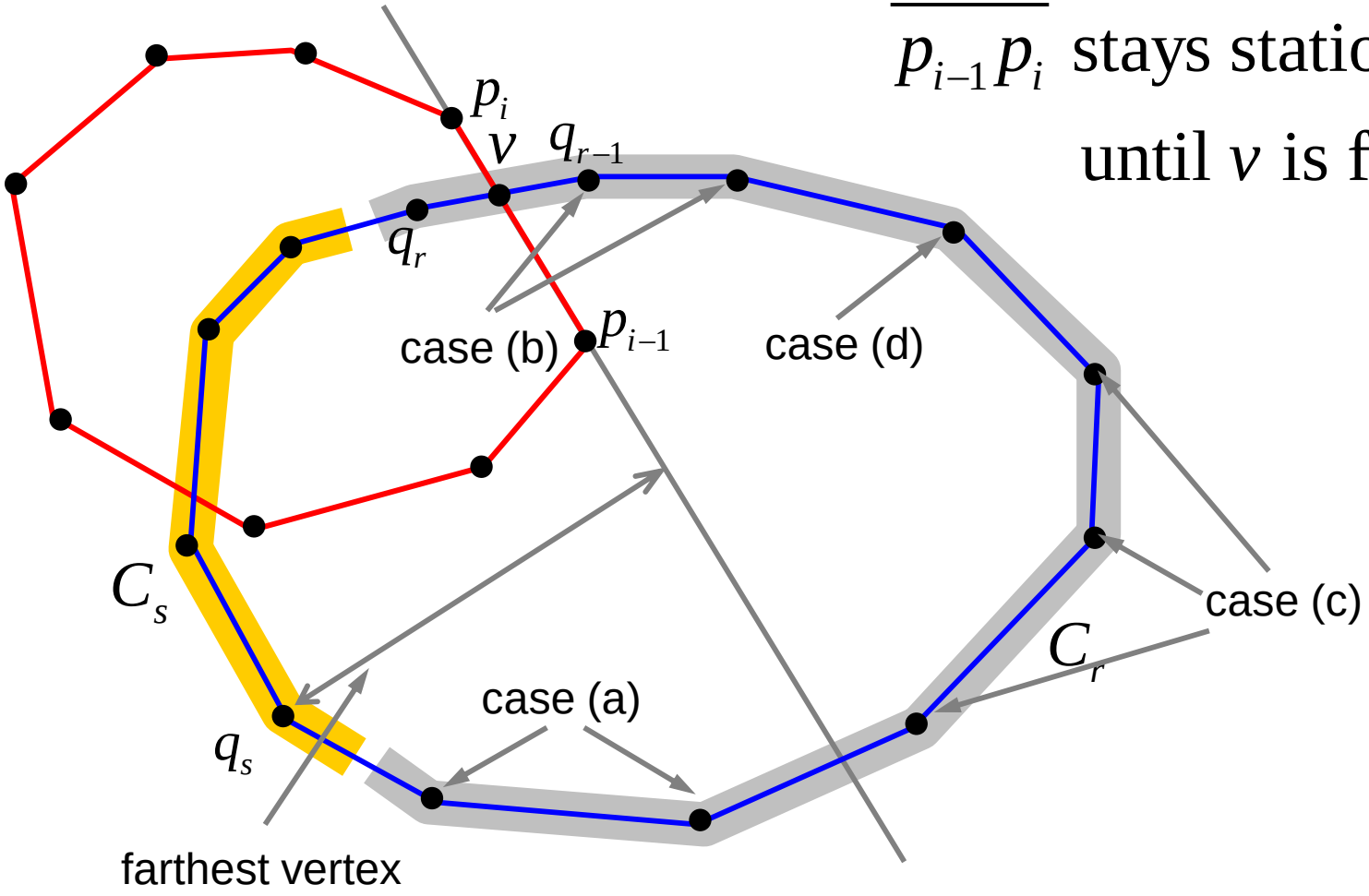
This in turn guarantees that once an intersection point of a sickle is found, all the others will be constructed successively.

To complete the proof we must show that an intersection point must be found if $P \bigcap Q \neq \varnothing$.

Let the current edge $\overline{p_{i-1}p_i}$ on P, contain an intersection point v with edge $\overline{q_{r-1}q_r}$ on Q, so $v \in h(p_i)$.

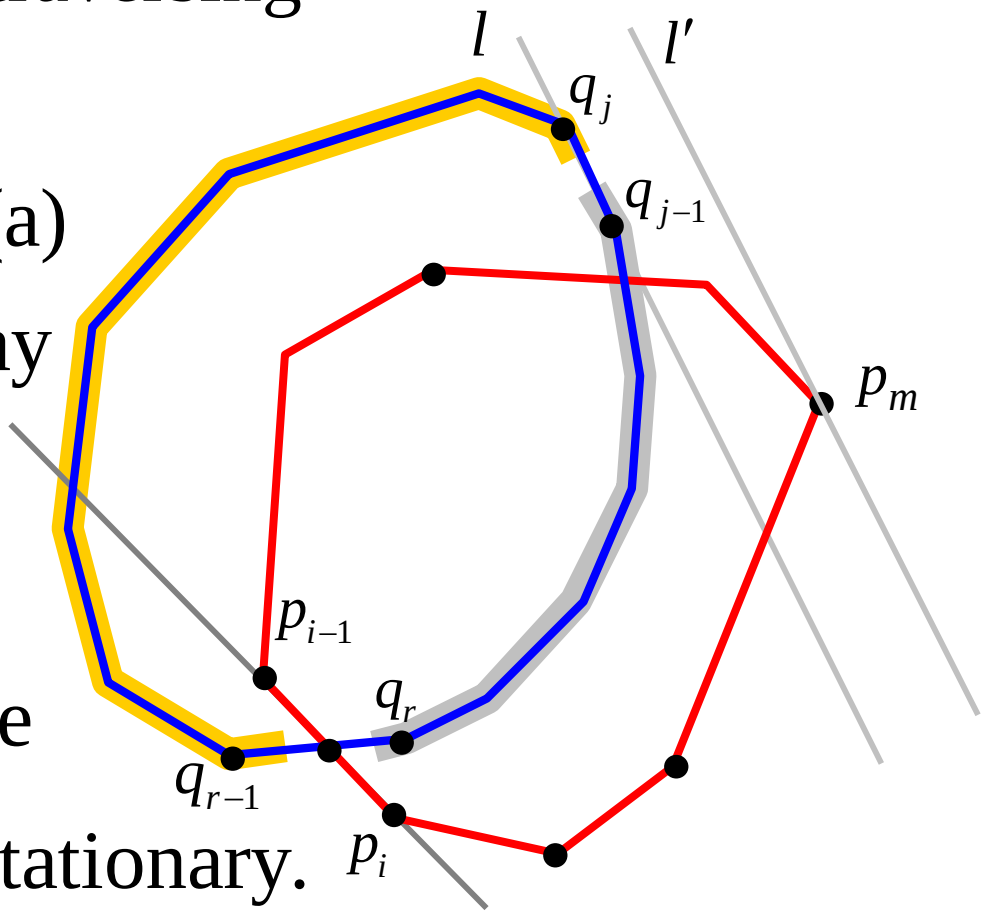

p_i
q_{r-1}
v
q_r
p_{i-1}
C_s
C_r
q_s
farthest vertex

Let $\overline{q_{j-1}q_j}$ be current edge and $q_j \in C_r$. Sequence of cases (a), (c), (d) and (b) (possibly empty) occurs, while $\overline{p_{i-1}p_i}$ stays stationary, until v is found.

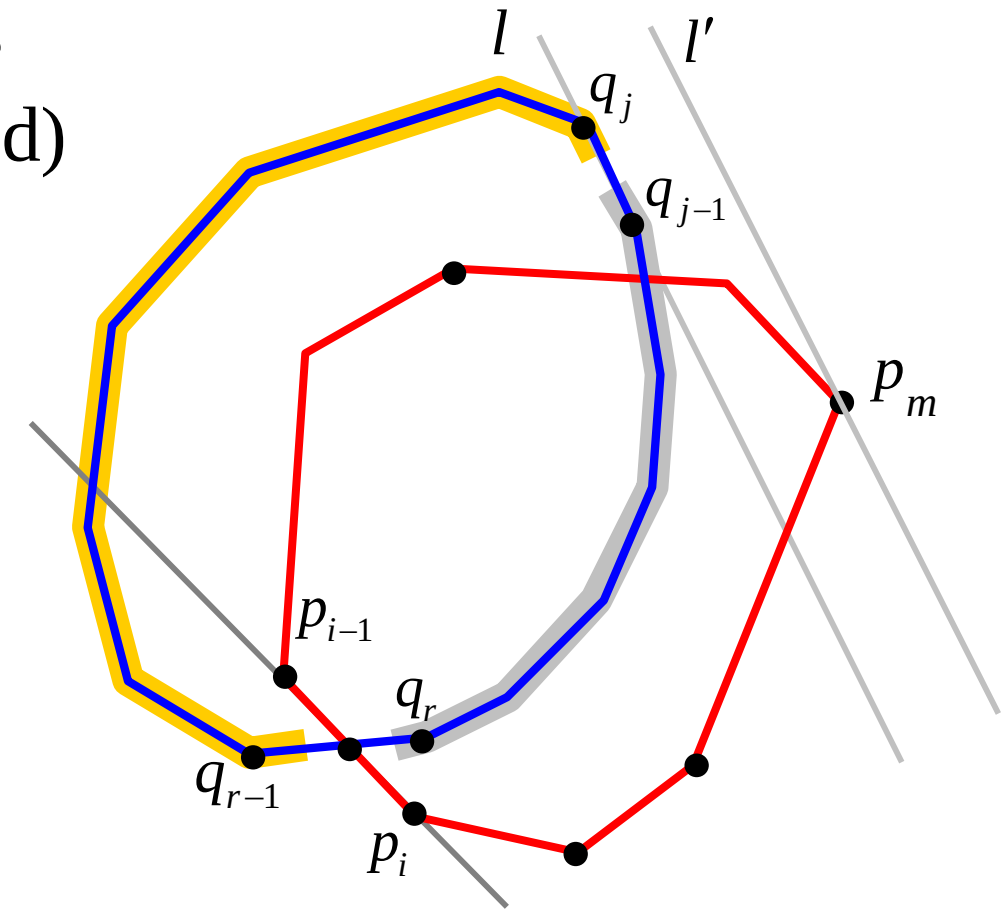


Let $q_j \in C_s$, l be a line determined by $\overline{q_{j-1}q_j}$ and $l' \parallel l$ supporting P at p_m, the first support reached when traversing from p_i along P.

A sequence of cases (a) starts at p_i and will stay so until crossing l, where case (c) holds up to p_m (may be empty). $\overline{q_{j-1}q_j}$ stays stationary.



If $p_m \notin h(q_j)$ ADVANCE continues marching along P with cases (d) and (b) until first point $p_t \in h(q_j)$ is found.



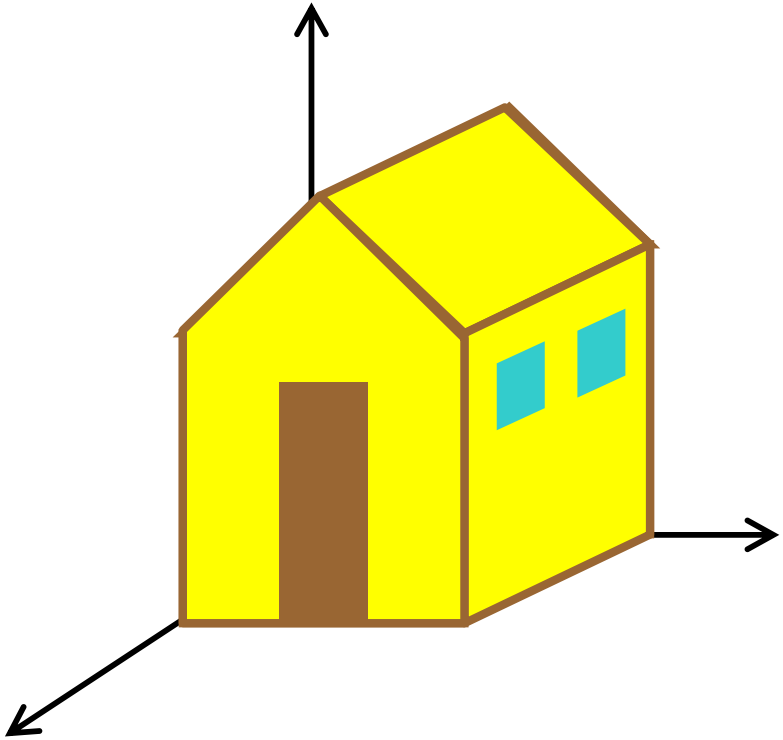
If an edge like $\overline{p_{i-1}p_i}$ exists, $L+M$ steps must reach it since the boundary of at least one polygon must be consumed.

Additional $L+M$ ADVANCE steps suffice to obtain all the intersection points in cyclic order, yielding $2(L+M)$ total steps of ADVANCE.

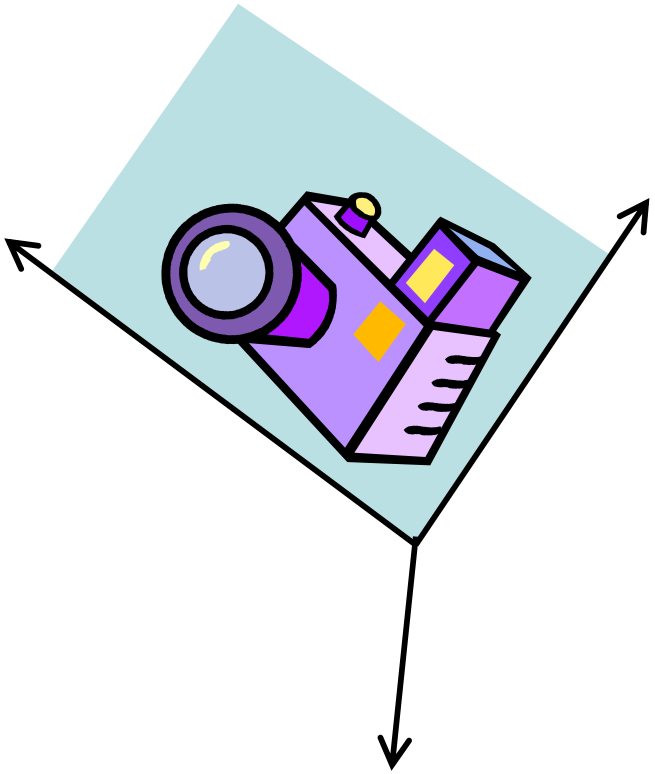
If no intersection was found then the relations $P \subseteq Q,\ Q \subseteq P$ or $P \bigcap Q = \varnothing$ can be resolved in $O(L+M)$ time.

3D Viewing Concepts

World Coordinate System



Viewing Coordinate System



2D Reminder

Choose viewing position, direction and orientation of the camera in the world.

A clipping window is defined by the size of the aperture and the lens.

Viewing by computing offers many more options which camera cannot, e.g., parallel or perspective projections, hiding parts of the scene, viewing behind obstacles, etc.

Clipping window: Selects ***what*** we want to see.

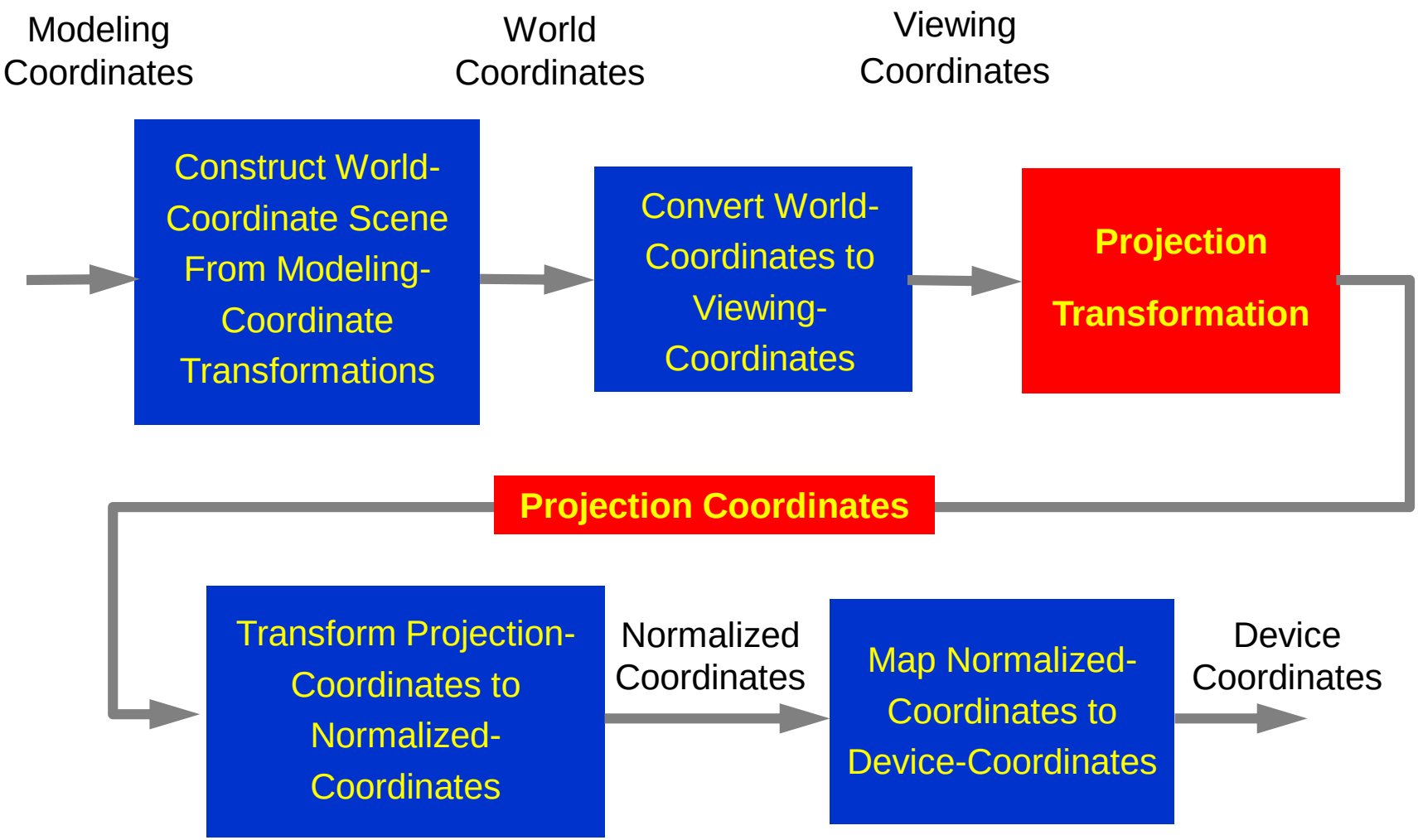
Viewport: Indicates ***where*** it is to be viewed on the output device (still in world coordinates).

Display window: Setting into screen coordinates.

In 3D the clipping is displayed on the view plane, but clipping of the scene takes place in the space by a clipping volume.

3D transformation pipeline is similar to 2D with addition of projection transformation.

3D Viewing Transformation Pipeline



Model is given in model (self) coordinates.

Conversion to world coordinates takes place.

Viewing coordinate system which defines the position and orientation of the projection plane (film plane in camera) is selected, to which scene is converted.

2D clipping window (lens of camera) is defined on the projection plane (film plane) and a 3D clipping, called ***view volume***, is established.

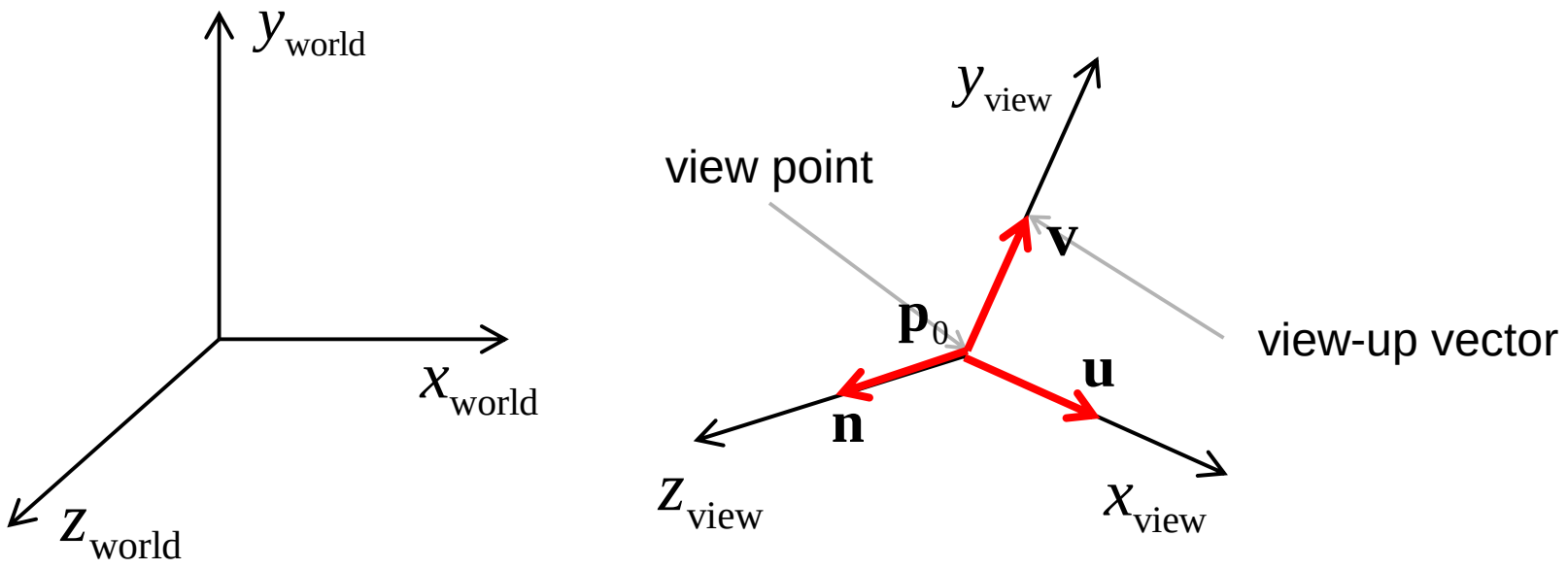
The shape and size of view volume is defined by the dimensions of clipping window, the type of projection and the limiting positions along the viewing direction.

Objects are mapped to normalized coordinated and all parts of the scene out of the view volume are clipped off.

The clipping is applied after all device independent transformation are completed, so efficient transformation concatenation is possible.

Few other tasks such as hidden surface removal and surface rendering take place along the pipeline.

World to Viewing 3D Transformation



$\mathbf{n}$: viewing directionon

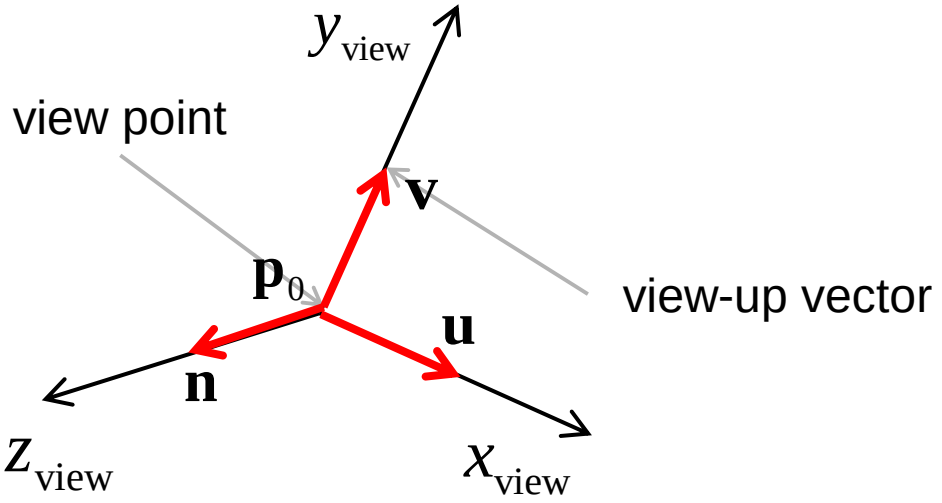
$\mathbf{u}-\mathbf{v}$: viewing plane

$\mathbf{u}=(u_x,u_y,u_z)$

$\mathbf{v}=(v_x,v_y,v_z)$

$\mathbf{n}=(n_x,n_y,n_z)$

$\mathbf{p}_0=(x_0,y_0,z_0)$



world to viewing transformation

$$\mathbf{M}_{WC,VC}=\begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\cdot\begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

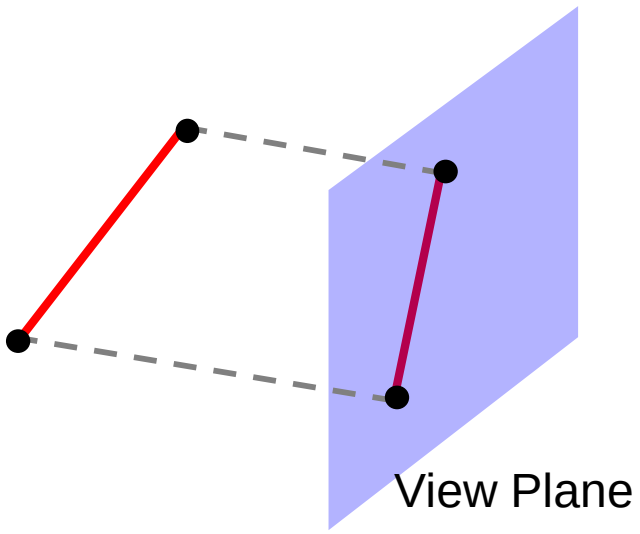
rotation translation

Projection Transformations

Next step in 3D viewing pipeline is projection of object to viewing plane

Parallel Projection

Coordinate are transferred to viewing plane along parallel lines.

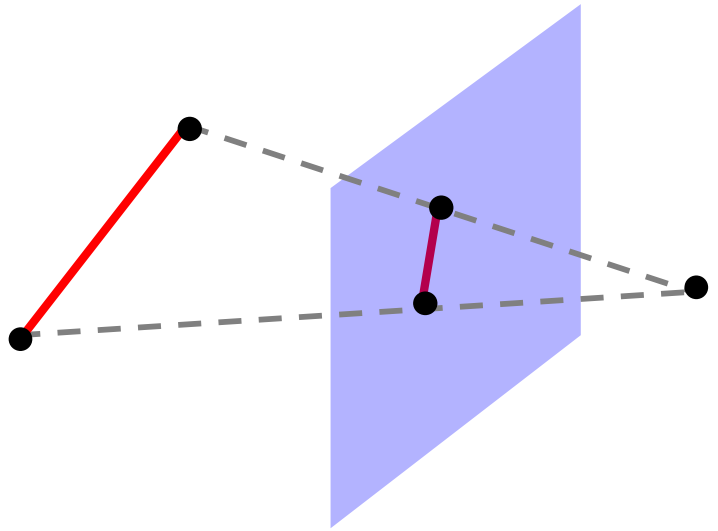


Preserves relative size of object's portions.

Projection can be perpendicular or oblique to viewing plane.

Perspective Projection

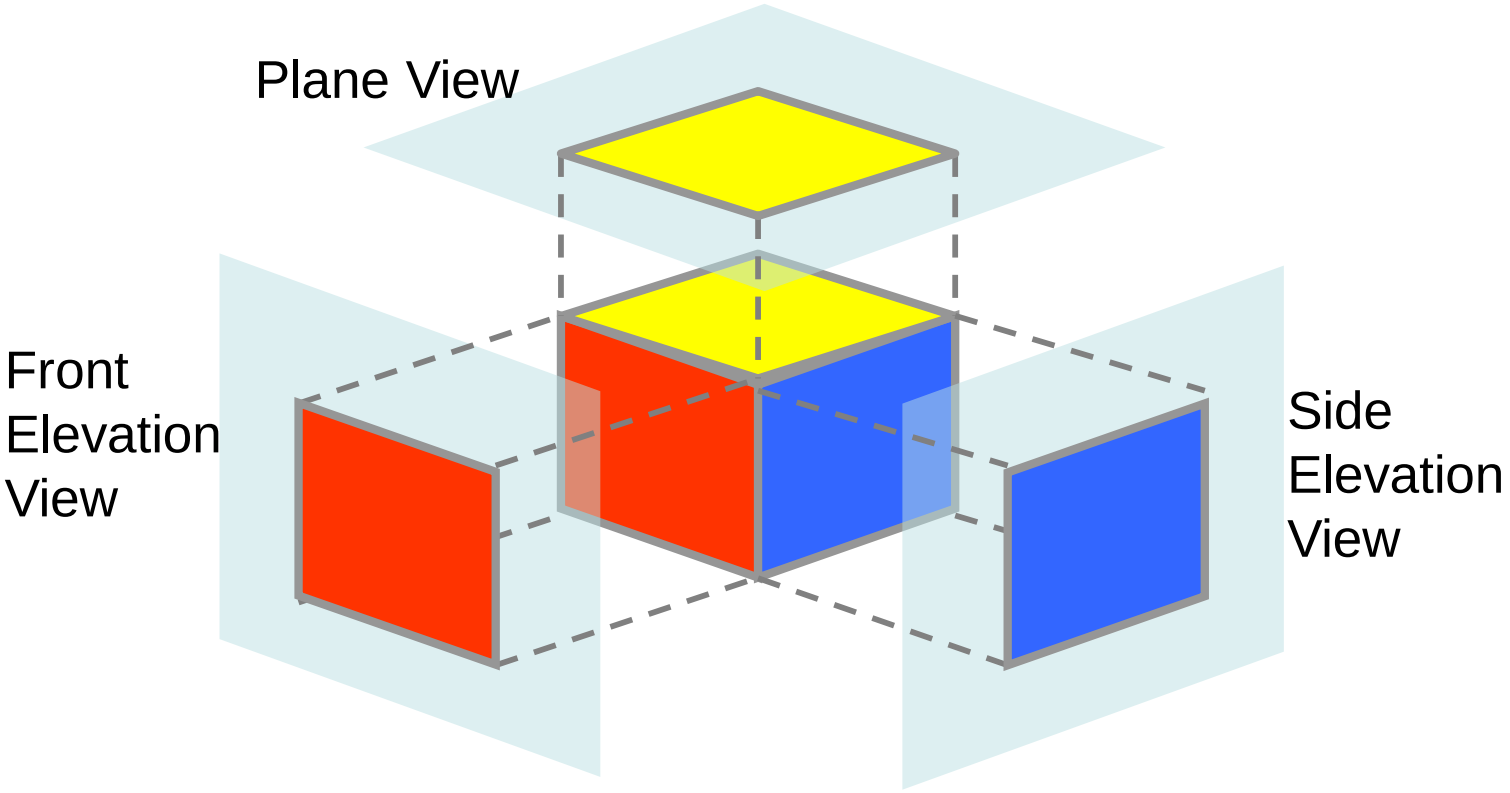
Projection lines converge in a point behind viewing plane.



Doesn’t preserve relative size but looks more realistic.

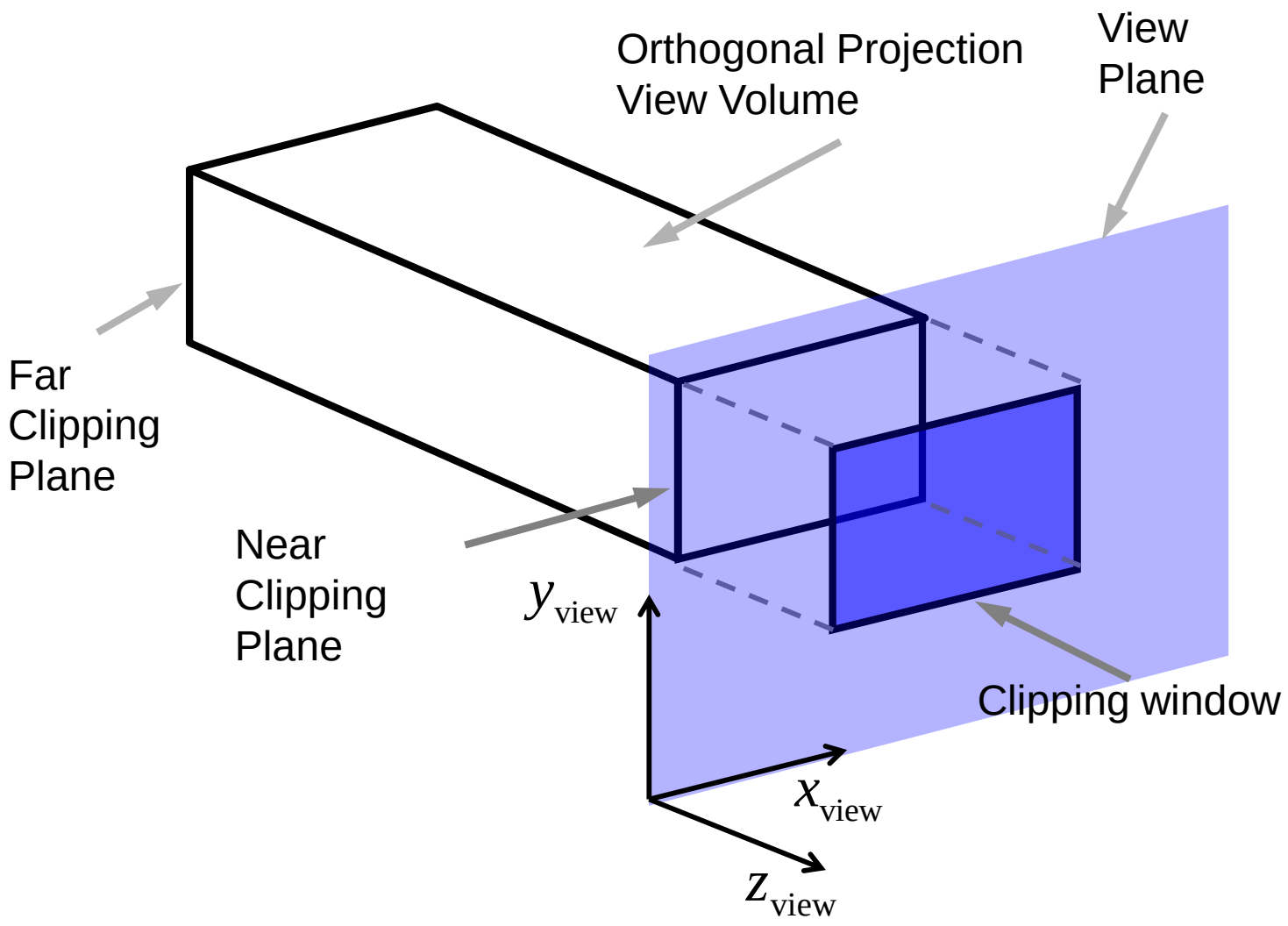
Orthogonal (orthographic) projections

Projection lines are parallel to normal.

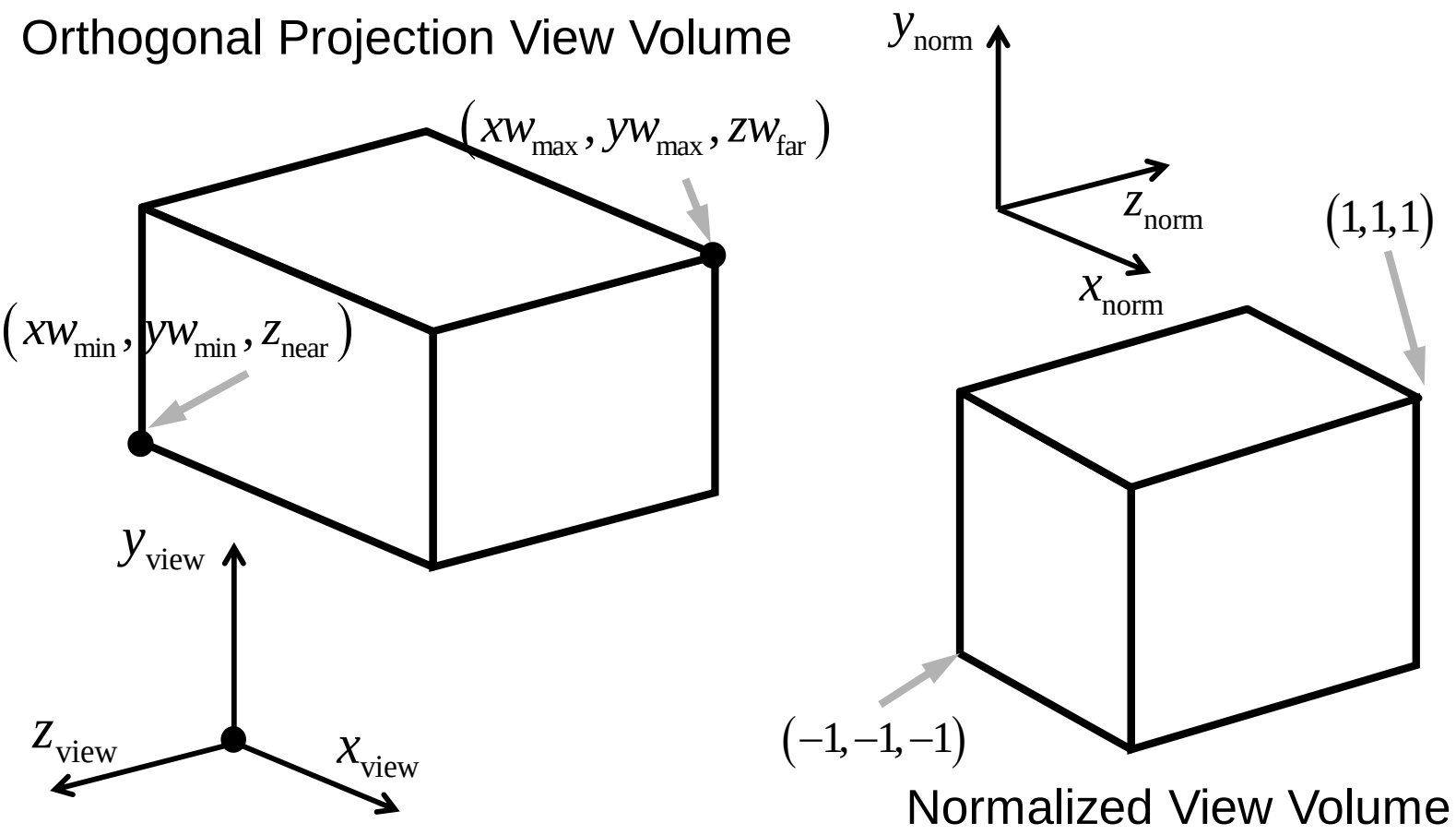


Used in engineering and architecture. Length and angles can be measured directly from drawings.

Clipping Window and View Volume



Normalizing Orthogonal Projection



Display coordinate system is usually left-handed.

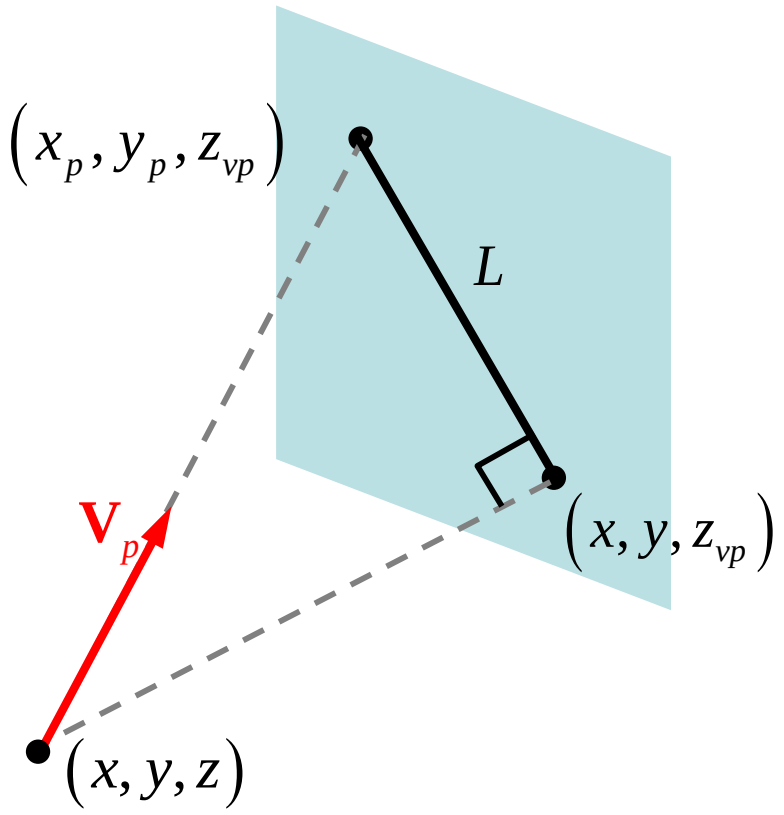
$$\mathbf{M}_{\text{ortho,norm}} =$$

$$\begin{bmatrix} \frac{2}{xw_{\max} - xw_{\min}} & 0 & 0 & -\frac{xw_{\max} + xw_{\min}}{xw_{\max} - xw_{\min}} \\ 0 & \frac{2}{yw_{\max} - yw_{\min}} & 0 & -\frac{yw_{\max} + yw_{\min}}{yw_{\max} - yw_{\min}} \\ 0 & 0 & \frac{-2}{z_{\text{near}} - z_{\text{far}}} & \frac{z_{\text{near}} + z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The complete transformation from world coordinated to normalized orthogonal-projection coordinates is obtained by $\mathbf{M}_{\text{ortho,norm}} \cdot \mathbf{M}_{\text{WC,VC}} \cdot$

Oblique Parallel Projections

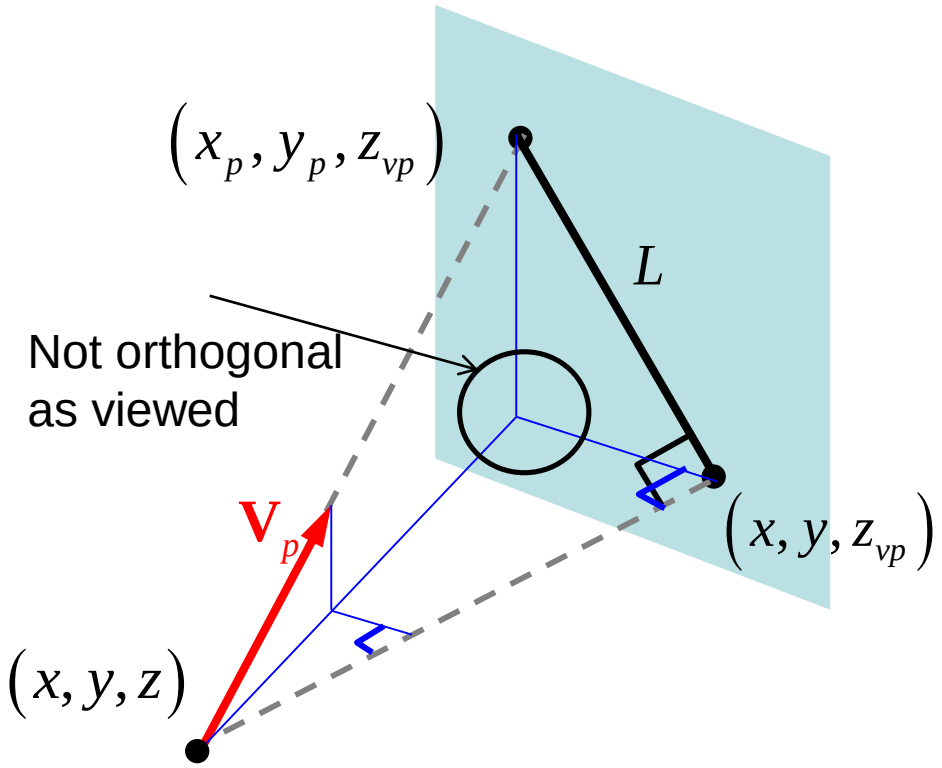
Projection is defined by a viewing vector $\mathbf{V}_p$.



Projection lines are parallel to vector $\mathbf{V}_p$.

$$\frac{x_p - x}{z_{vp} - z} = \frac{V_{px}}{V_{pz}}$$

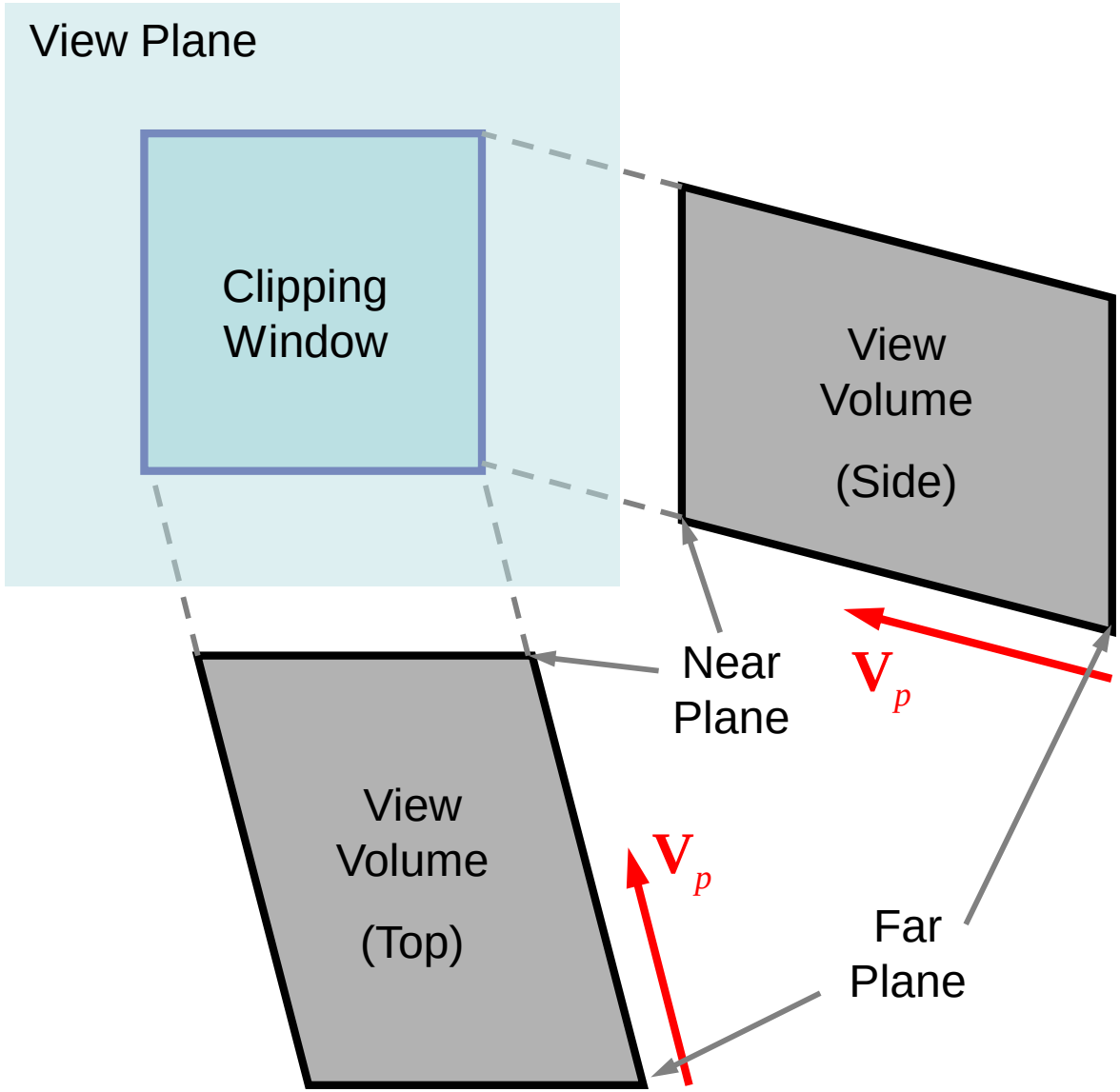
$$\frac{y_p - y}{z_{vp} - z} = \frac{V_{py}}{V_{pz}}$$



$$x_p = x + (z_{vp} - z)(V_{px}/V_{pz}) \;,\; y_p = y + (z_{vp} - z)(V_{py}/V_{pz})$$

The above is called shear transformation, where the displacement in x and y linearly increases with z.

View Volume of Parallel Oblique Projection



$$\mathbf{M}_{\text{oblique}} = \begin{bmatrix} 1 & 0 & -\frac{V_{px}}{V_{pz}} & z_{vp}\frac{V_{px}}{V_{pz}} \\ 0 & 1 & -\frac{V_{py}}{V_{pz}} & z_{vp}\frac{V_{py}}{V_{pz}} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This is a 3D shear transformation.

x and y are displaced by amount proportional to z.

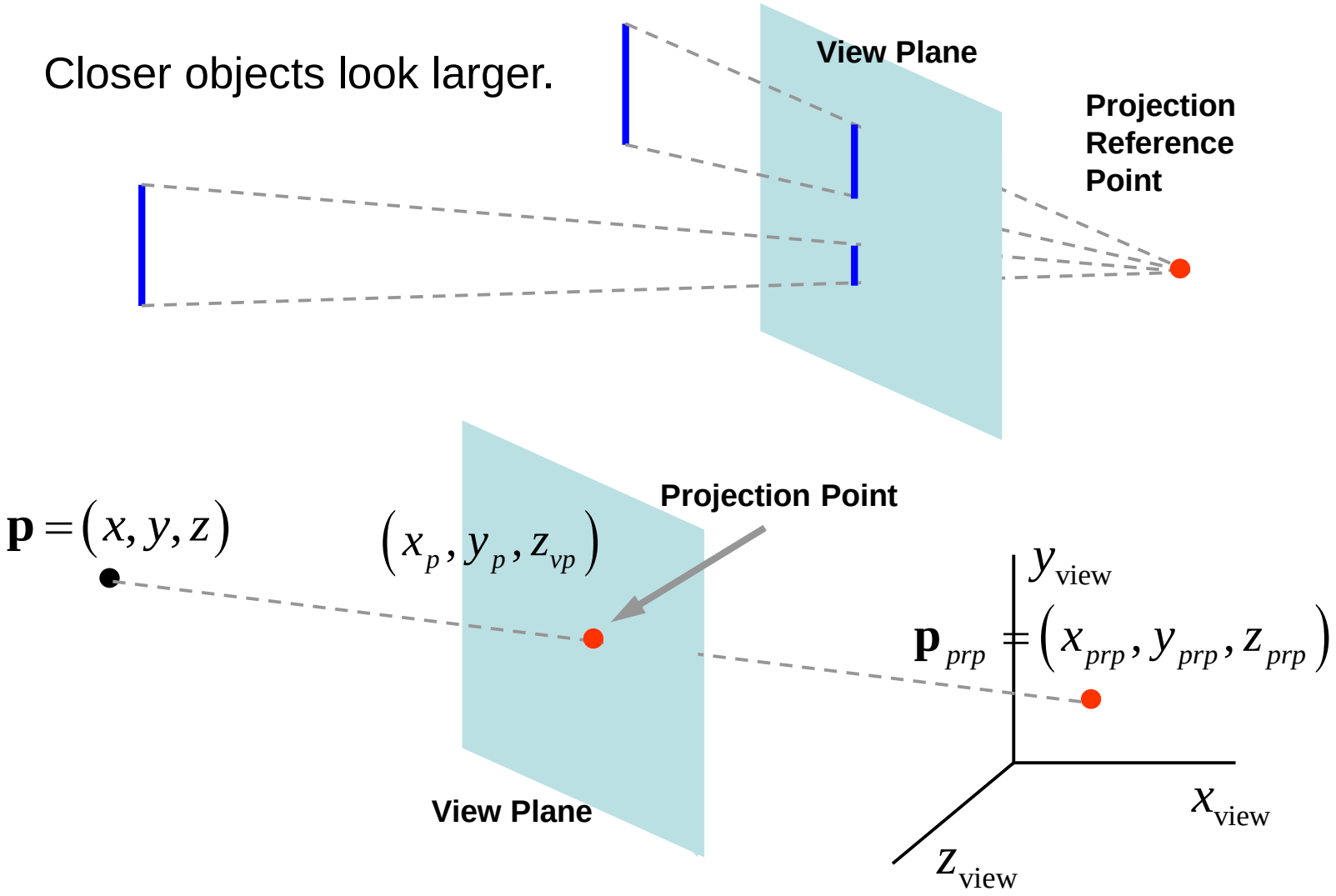
Normalization oblique projection is similar to orthogonal projection.

The composite transformation is obtained by the product of the two.

$$\mathbf{M}_{\text{oblique,norm}} = \mathbf{M}_{\text{ortho,norm}} \cdot \mathbf{M}_{\text{oblique}}$$

Perspective Projections

Closer objects look larger.



Parametric representation of a point on the line connecting $\mathbf{P} = (x, y, z)$ with $\mathbf{P}_{prp} = (x_{prp}, y_{prp}, z_{prp})$:

$$x' = x - (x - x_{prp})u \ , \ y' = y - (y - y_{prp})u \ ,$$

$$z' = z - (z - z_{prp})u \ , \ 0 \le u \le 1.$$

At viewing plane: $u = (z_{vp} - z) / (z_{prp} - z)$.

Substitution for the point on viewing plane:

$$x_p = x(z_{prp} - z_{vp}) / (z_{prp} - z) + x_{prp}(z_{vp} - z) / (z_{prp} - z)$$

$$y_p = y(z_{prp} - z_{vp}) / (z_{prp} - z) + y_{prp}(z_{vp} - z) / (z_{prp} - z)$$

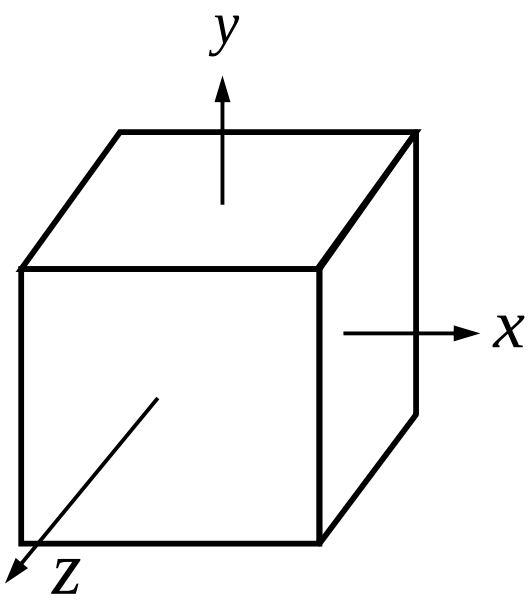
The problem with the above representation is that *Z* appears in denominator, so matrix multiplication representation of *X* and *Y* on view plane as a function of *Z* is not straight forward.

Z is point specific, hence division will be computation killer.

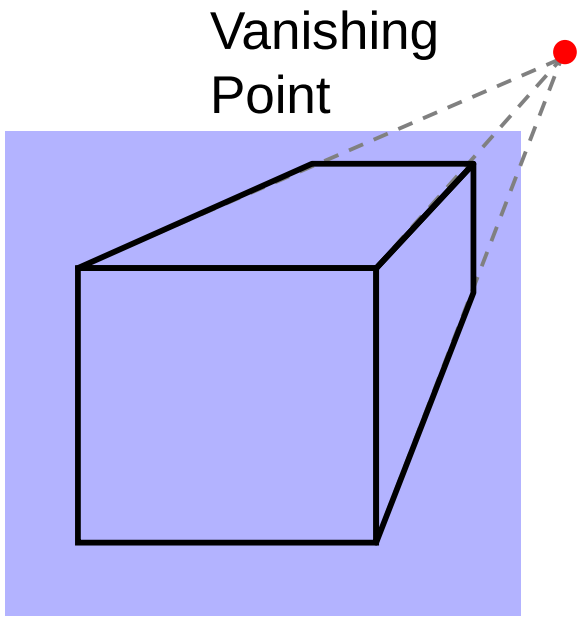
Different representation is in order, so transformations can be concatenated

Vanishing Points

Vanishing points occur when the viewing plane intersects with the axes of viewing coordinate system.



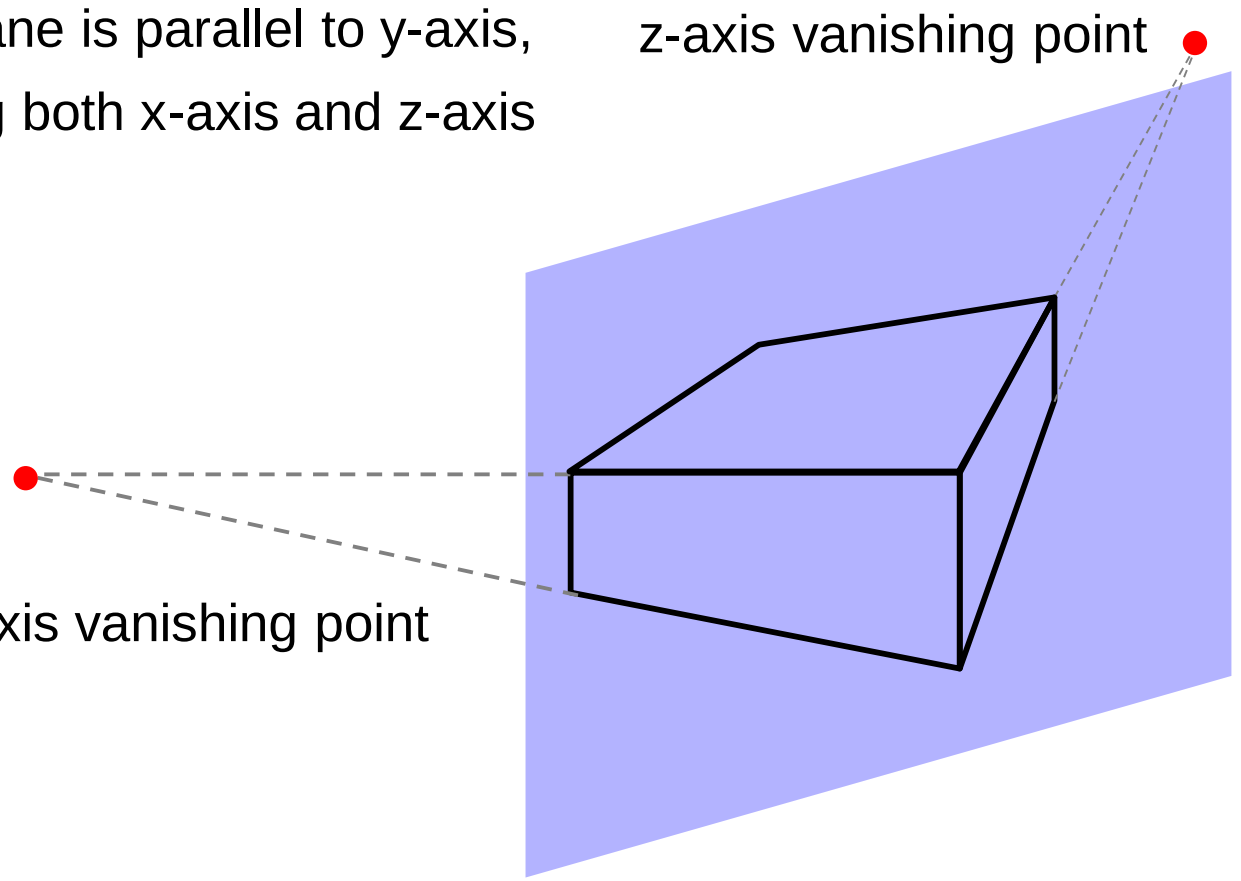
Principle axes for cube



One-Point perspective Projection

Parallel to Z lines of XZ plane and parallel lines to Z in YZ plane will vanish. Vanishing point on viewing plane correspond to infinity in world.

Viewing plane is parallel to y-axis, intersecting both x-axis and z-axis



Vanishing points of all three axes occur when viewing plane intersects all three axes.

Transformation Matrix

To get rid of z in denominator we define parameter $h = z_{prp} - z$ and new homogeneous coordinates (x_h, y_h, z_h, h).

$x_p = x_h / h,\ y_p = y_h / h,$ yielding:

$$x_h = x\left(z_{prp} - z_{vp}\right) + x_{prp}\left(z_{vp} - z\right),$$
$$y_h = y\left(z_{prp} - z_{vp}\right) + y_{prp}\left(z_{vp} - z\right).$$

$$\mathbf{P} = (x, y, z, 1), \quad \mathbf{P}_h = (x_h, y_h, z_h, h), \quad \mathbf{P}_h = \mathbf{M}_{\text{pers}} \cdot \mathbf{P},$$

Matrix Representation:

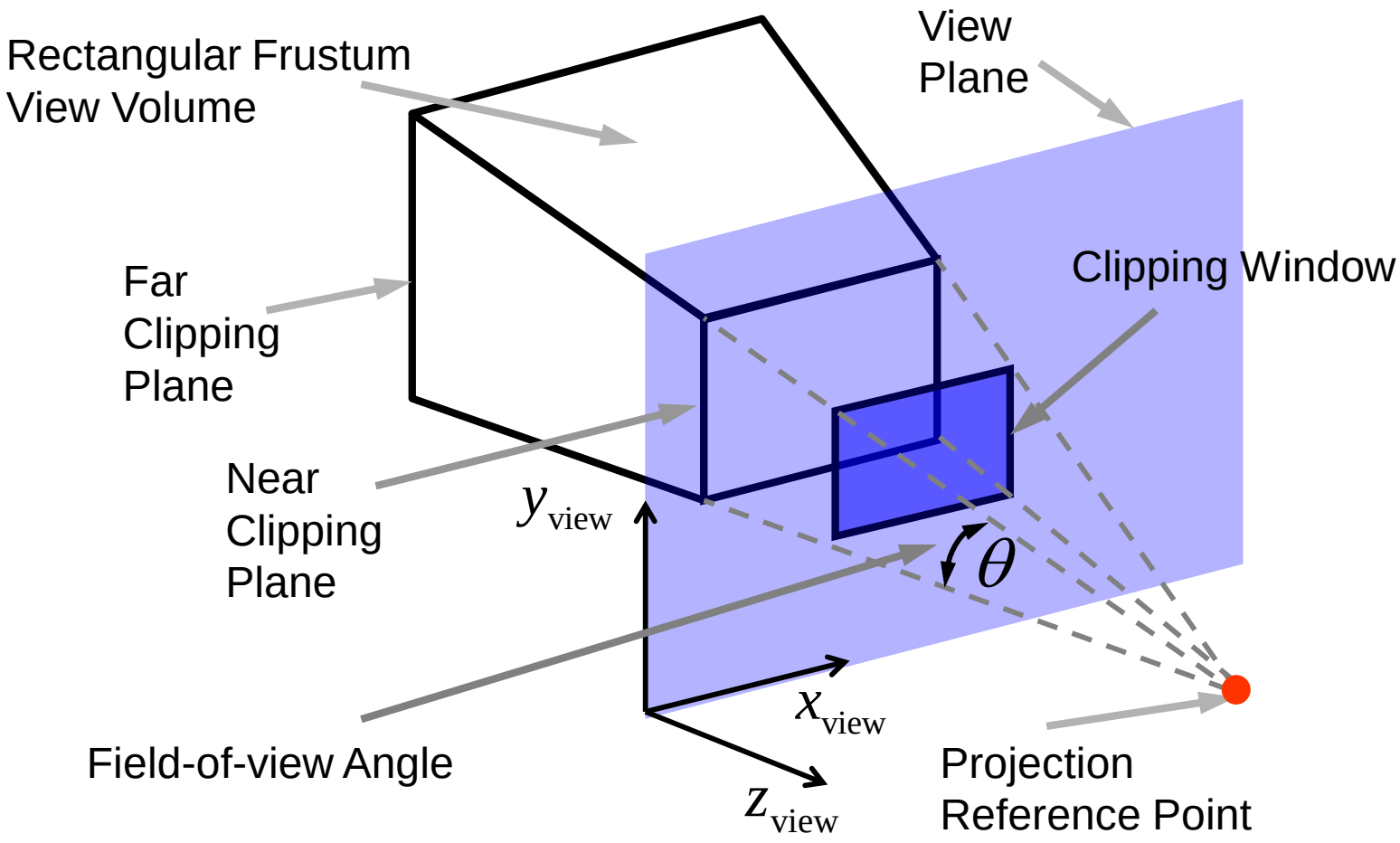
$$\mathbf{M}_{\text{pers}} = \begin{bmatrix} z_{prp} - z_{vp} & 0 & -x_{prp} & x_{prp} z_{prp} \\ 0 & z_{prp} - z_{vp} & -y_{prp} & y_{prp} z_{prp} \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & z_{prp} \end{bmatrix} \quad z_h = z s_z + t_z$$

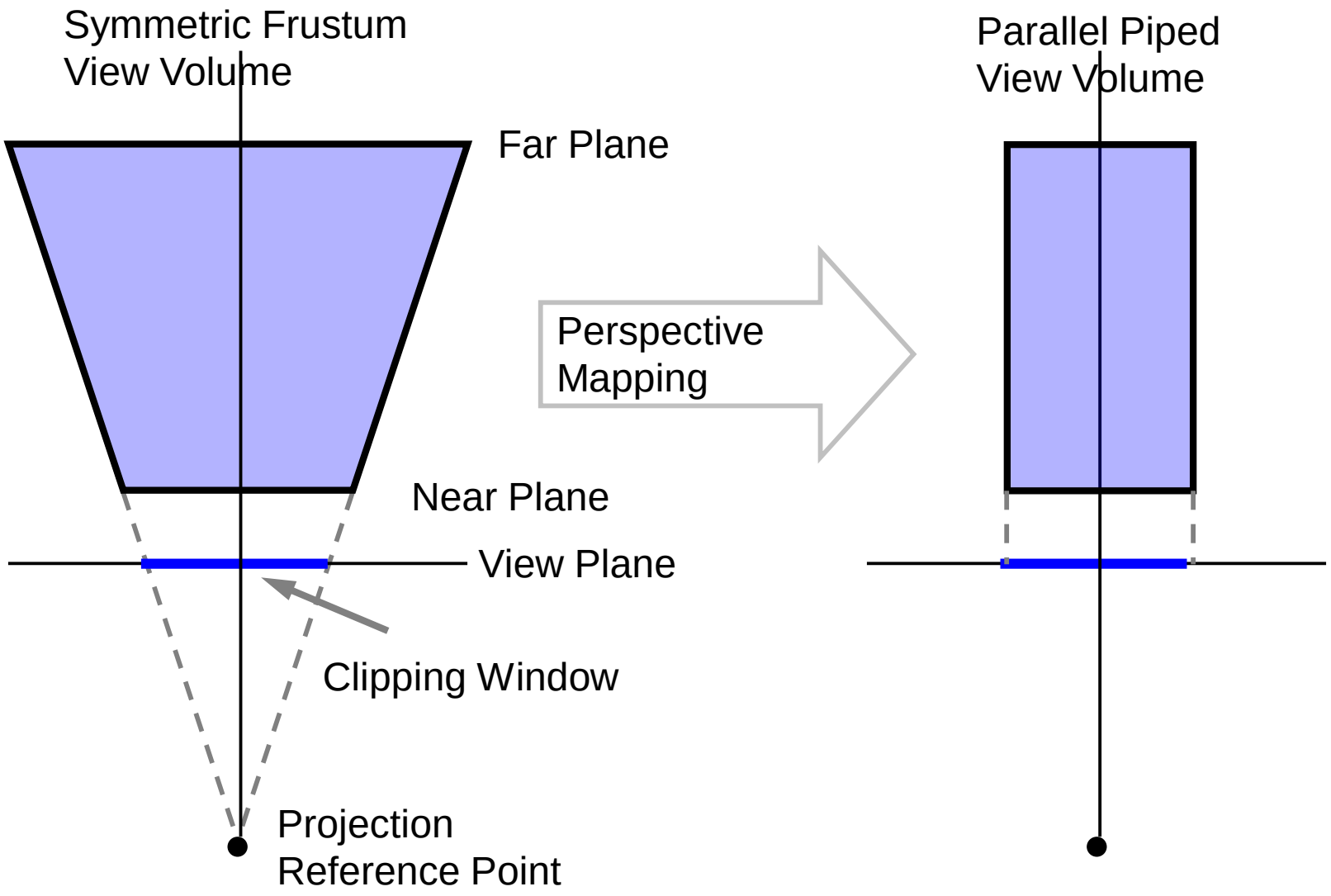
$z_h = z s_z + t_z$ is using a scaling factor s_z and translation t_z. s_z and t_z can be set arbitrarily. We'll set those to satisfy desired normalization.

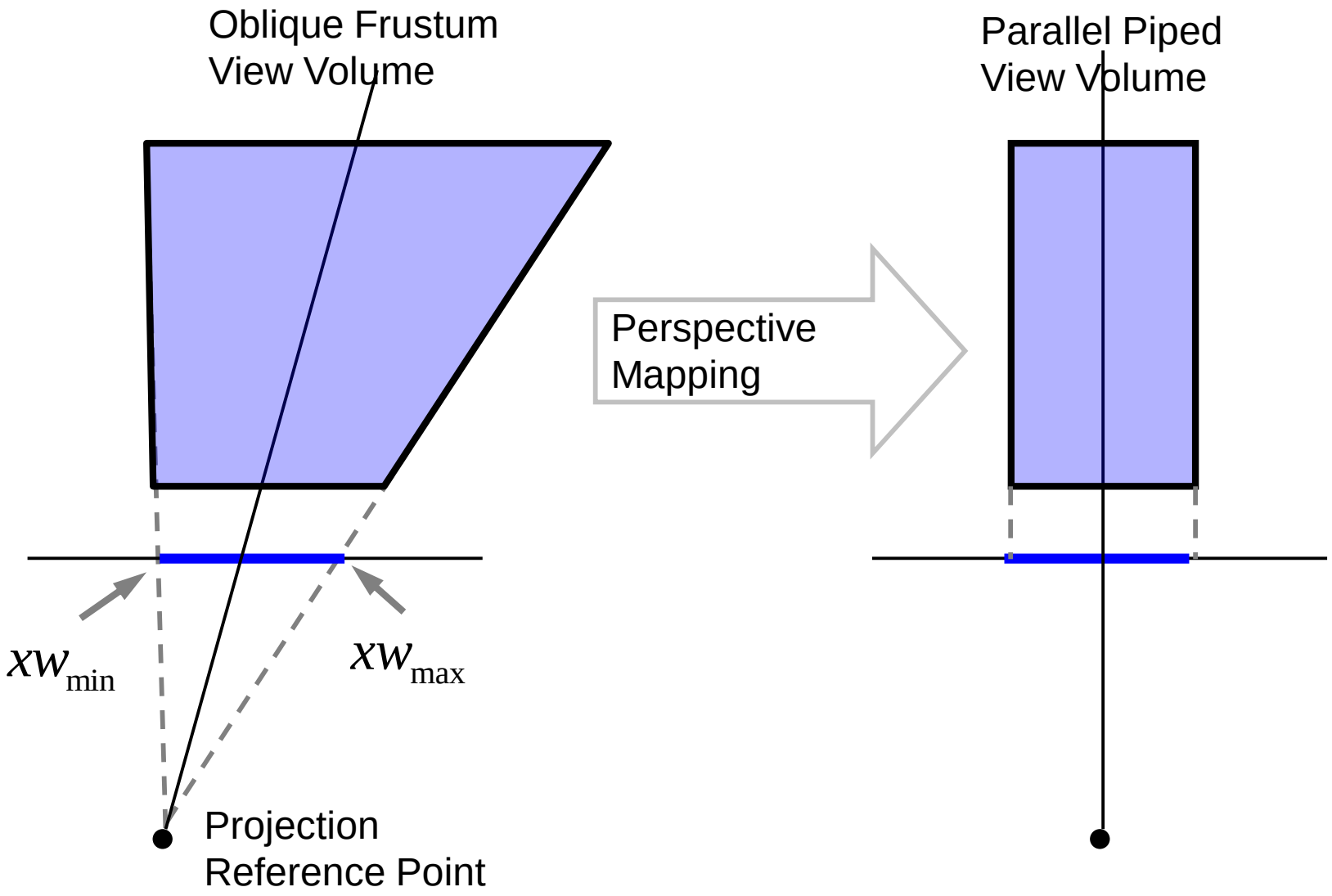
Transformed z coordinate is very useful for deciding later on the hidden parts of the scene.

Notice that drawing the perspective point requires a division by h. h depends on z coordinate, hence evey point must be divided by a different number, which is very expensive!

Perspective-Projection View Volume




Symmetric Frustum
View Volume
Far Plane
Perspective
Mapping
Parallel Piped
View Volume
Near Plane
View Plane
Clipping Window
Projection
Reference Point


Oblique Frustum
View Volume
Parallel Piped
View Volume
Perspective
Mapping
xw_{min}
xw_{max}
Projection
Reference Point

Oblique Perspective-Projection Frustum

We'll take the projection reference point to be the origin of viewing coordinate system $(x_{prp}, y_{prp}, z_{prp}) = (0,0,0)$ and viewing plane at near clipping plane. Oblique projection results shear transformation. It transforms the intersection point of center-line with clipping window $\left(\frac{xw_{\min} + xw_{\max}}{2}, \frac{yw_{\min} + yw_{\max}}{2}, z_{\text{near}}\right)$ to $(0,0,z_{\text{near}})$.

$$\mathbf{M}_{\text{z shear}} = \begin{bmatrix} 1 & 0 & \text{sh}_{zx} & 0 \\ 0 & 1 & \text{sh}_{zy} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ Z_{\text{naer}} \\ 1 \end{bmatrix} = \mathbf{M}_{\text{z shear}} \cdot \begin{bmatrix} (xw_{\min} + xw_{\max})/2 \\ (yw_{\min} + yw_{\max})/2 \\ Z_{\text{naer}} \\ 1 \end{bmatrix}$$

which solves to $\text{sh}_{zx} = -\dfrac{xw_{\min} + xw_{\max}}{2}$ and $\text{sh}_{zy} = -\dfrac{yw_{\min} + yw_{\max}}{2}$.

Substituting $\left(x_{prp}, y_{prp}, z_{prp}\right) = (0,0,0)$ in the perspective-projection matrix and positioning the viewing near z clipping plane at the view plane simplifies it to

$$\mathbf{M}_{\text{pers}} = \begin{bmatrix} -z_{\text{near}} & 0 & 0 & 0 \\ 0 & -z_{\text{near}} & 0 & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

The z coordinate scaling factor s_z and the translation t_z will be determined by the normalization requirements.

The complete oblique perspective-projection is obtained by concatenating the perspective and shear matrices.

$$\mathbf{M}_{\text{obliquepers}} = \mathbf{M}_{\text{pers}} \cdot \mathbf{M}_{\text{z shear}} = \begin{bmatrix} -z_{\text{near}} & 0 & \left(xw_{\min} + xw_{\max}\right)/2 & 0 \\ 0 & -z_{\text{near}} & \left(yw_{\min} + yw_{\max}\right)/2 & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

For symmetric viewing volume $\left(xw_{\min} = -xw_{\max}\ ,\ yw_{\min} = -yw_{\max}\right)$.

Transformation is simplified to:

$$\mathbf{M}_{\text{symmetricpers}} = \mathbf{M}_{\text{pers}} \cdot \mathbf{M}_{\text{z shear}} = \begin{bmatrix} -z_{\text{near}} & 0 & 0 & 0 \\ 0 & -z_{\text{near}} & 0 & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Normalized Perspective-Projection

Normalization is obtained by multiplying with standard scaling matrix.

$$\mathbf{M}_{\text{normpers}} = \mathbf{M}_{xy\ \text{scale}} \cdot \mathbf{M}_{\text{obliquepers}}$$

$$= \begin{bmatrix} -z_{\text{near}} s_x & 0 & s_x \left(xw_{\min} + xw_{\max} \right)/2 & 0 \\ 0 & -z_{\text{near}} s_y & s_y \left(yw_{\min} + yw_{\max} \right)/2 & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Homogenous coordinates are obtained as follows:

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \mathbf{M}_{\text{normpers}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Projection coordinates are:

$$x_p = x_h / h = \left[-z_{\text{near}} s_x x + s_x \left(x w_{\min} + x w_{\max} \right) z / 2 \right] / -z,$$

$$y_p = y_h / h = \left[-z_{\text{near}} s_y y + s_y \left(y w_{\min} + y w_{\max} \right) z / 2 \right] / -z,$$

$$z_p = z_h / h = \left(s_z z + t_z \right) / -z.$$

We'd like normalization transform to result:

$$\left(x w_{\min}, y w_{\min}, z_{\text{near}} \right) \Rightarrow \left(-1, -1, -1 \right) \text{ and } \left(x w_{\max}, y w_{\max}, z_{\text{far}} \right) \Rightarrow \left(1, 1, 1 \right).$$

Substitution in above equations yields:

$$s_x = 2 / \left(x w_{\max} - x w_{\min} \right), \; s_y = 2 / \left(y w_{\max} - y w_{\min} \right),$$

$$s_z = \left(z_{\text{near}} + z_{\text{far}} \right) / \left(z_{\text{near}} - z_{\text{far}} \right), \; \mathrm{t}_z = -2 z_{\text{near}} z_{\text{far}} / \left(z_{\text{near}} - z_{\text{far}} \right).$$

Back substitution in the normalized perspective transformation yields:

$$\mathbf{M}_{\text{normpers}} = \begin{bmatrix} \frac{-2z_{\text{near}}}{xw_{\max} - xw_{\min}} & 0 & \frac{xw_{\min} + xv_{\max}}{xw_{\max} - xw_{\min}} & 0 \\ 0 & \frac{-2z_{\text{near}}}{yw_{\max} - yw_{\min}} & \frac{yw_{\min} + yw_{\max}}{yw_{\max} - yw_{\min}} & 0 \\ 0 & 0 & \frac{z_{\text{near}} + z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}} & -\frac{2z_{\text{near}} z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

No division by z!

3D Viewport Transformation

The normalized view volume cube extending from $(-1,-1,-1)$ to $(1,1,1)$ is mapped to a screen viewport, extending from $(xv_{\min}, yv_{\min})$ to $(xv_{\max}, yv_{\max})$. z information is stored for depth calculations. z is often renormalized to the range from 0 to 1.0, yielding:

$$\mathbf{M}_{\substack{\text{normviewvol,}\\ \text{3D screen}}} = \begin{bmatrix} \frac{xv_{\max} - xv_{\min}}{2} & 0 & 0 & \frac{xv_{\max} + xv_{\min}}{2} \\ 0 & \frac{yv_{\max} - yv_{\min}}{2} & 0 & \frac{yv_{\max} + yv_{\min}}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Settings of Perspective Projection

- Perspective projection point
  - Where the viewer (camera, eye) is positioned in the world.
- Positioning viewing plane with respect to viewing coordinates
  - Results vanishing points, one, two or three.
- Clipping window on viewing plane
  - Defines the infinite pyramid view volume.
- Near and far clipping planes (parallel to view plane)
  - Define the rectangular frustum view volume.
- Scale and translation parameters of perspective matrix
  - Define the normalization range.

3D Clipping

Clipping can take place on normalized cube:

$$xw_{\min} = -1,\ yw_{\min} = -1,\ zw_{\min} = -1,\ xw_{\max} = 1,\ yw_{\max} = 1,\ zw_{\max} = 1.$$

Similar to 2D, we add two bits to code the far and near planes.

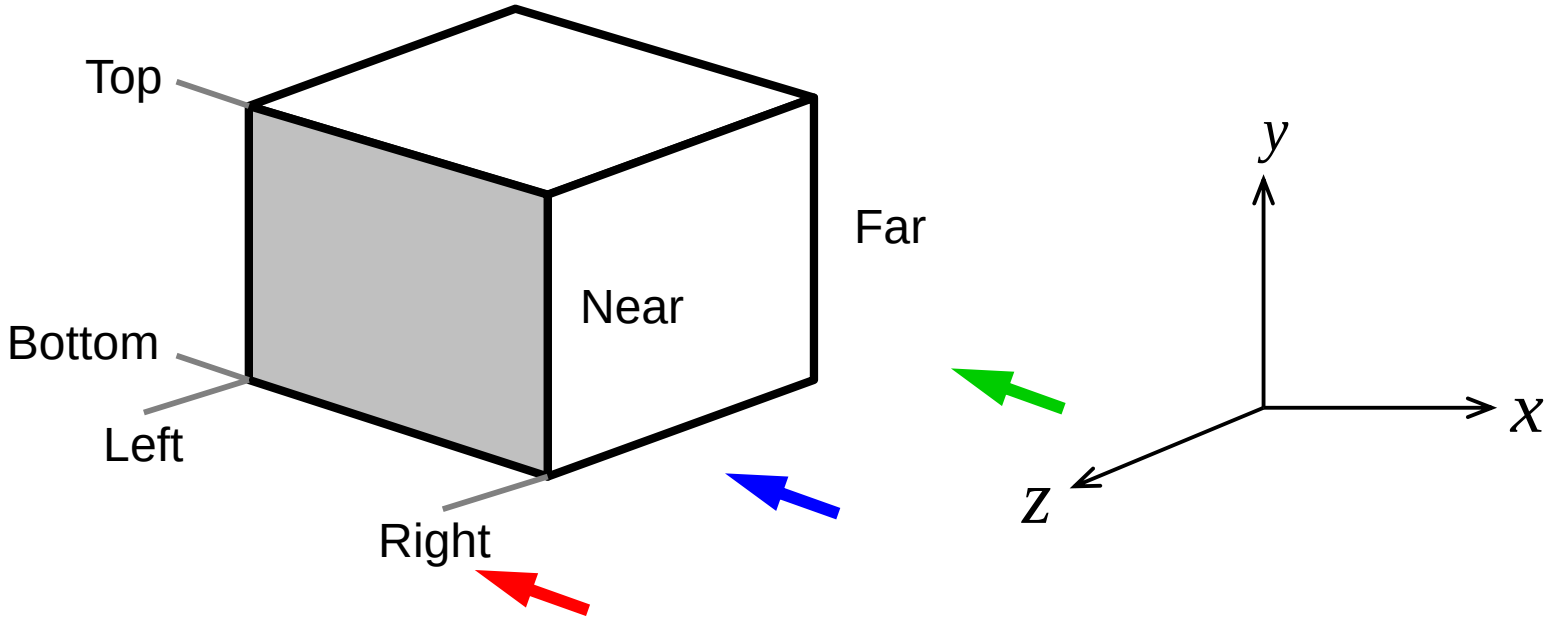
Far bit	Near bit	Top bit	Bottom bit	Right bit	Left bit

Recall that in 3D point is represented in homogeneous form. Hence

$\mathbf{P} = (x_h, y_h, z_h, h)$ is inside the normalized cube iff

$$-h \le x_h \le h, \quad -h \le y_h \le h, \quad -h \le z_h \le h \quad \text{if } h > 0$$

$$h \le x_h \le -h, \quad h \le y_h \le -h, \quad h \le z_h \le -h \quad \text{if } h < 0$$



011001	011000	011010
010001	010000	010010
010101	010100	010110

001001	001000	001010
000001	000000	000010
000101	000100	000110

101001	101000	101010
100001	100000	100010
100101	100100	100110

A line is completely accepted if the codes of its ends are both 000000, or equivalently, if the logical OR between codes is zero.

A line is rejected if the codes of its ends has at least one 1 in same bit, or equivalently, if the logical AND between codes is nonzero.

Otherwise, the line is tested against each of the planes and the 2D Liang-Barsky algorithm can be extended to 3D.

A point $\mathbf{P}$ of a line segment $\overline{\mathbf{P}_1\mathbf{P}_2}$ extending from $\mathbf{P}_1 = \left(x_{h_1}, y_{h_1}, z_{h_1}, h_1\right)$ to $\mathbf{P}_2 = \left(x_{h_2}, y_{h_2}, z_{h_2}, h_2\right)$ is given by $\mathbf{P} = \mathbf{P}_1 + \left(\mathbf{P}_2 - \mathbf{P}_1\right)u, \quad 0 \le u \le 1.$

If for instance the codes of the two end points of $\overline{\mathbf{P}_1\mathbf{P}_2}$ w.r.t the right clipping plane $x_{\max} = 1$ are different, the intersection point is derived from $x_p = x_h / h = \left[x_{h_1} + \left(x_{h_2} - x_{h_1} \right) u \right] \Big/ \left[h_1 + \left(h_2 - h_1 \right) u \right] = 1.$

Solving for u yields $u = \left[x_{h_1} - h_1 \right] \Big/ \left[\left(x_{h_1} - h_1 \right) - \left(x_{h_2} - h_2 \right) \right].$

Such calculation proceeds for each $up_k \le q_k,\ 1 \le k \le 6,$ while updating u_0 and u_1. If at some iteration $u_1 \le u_0,$ line segment is completely clipped. If upon termination $u_1 > u_0,$ the end points of the clipped line segment are obtained.