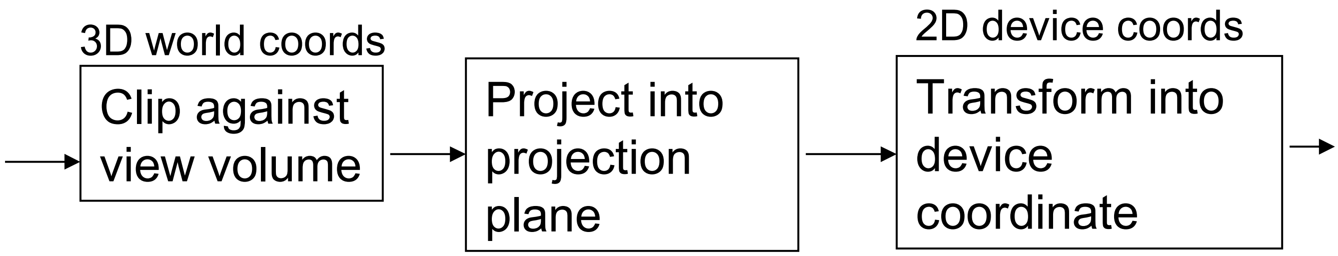
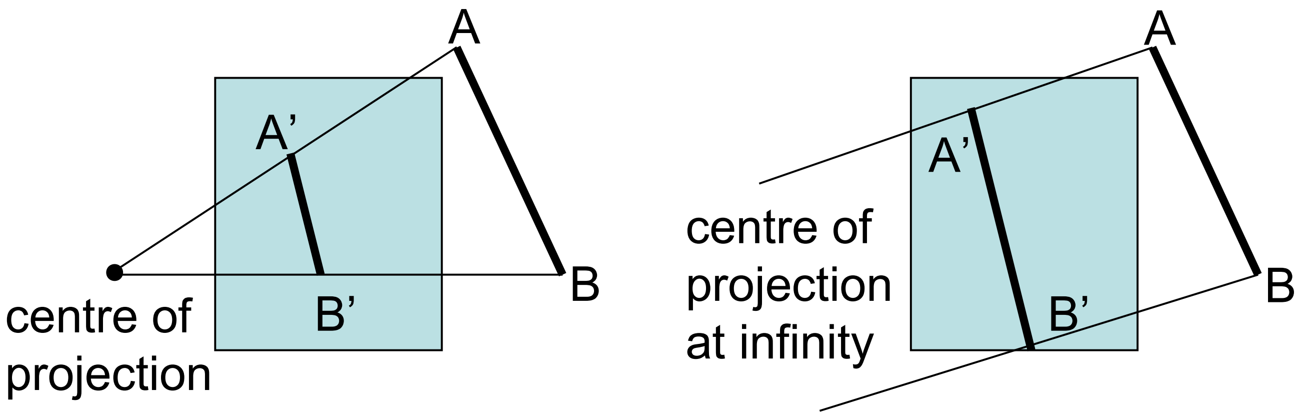
Projection

- Conceptually the 3D viewing process is:



- The projection is defined by projection rays (*projectors*) that come from a *centre of projection* and pass through each point of the object, and intersect with a *projection plane*.
  - If the centre of projection is at infinity we have ***parallel projection***.
  - If the centre of projection is at a point we have ***perspective projection***.

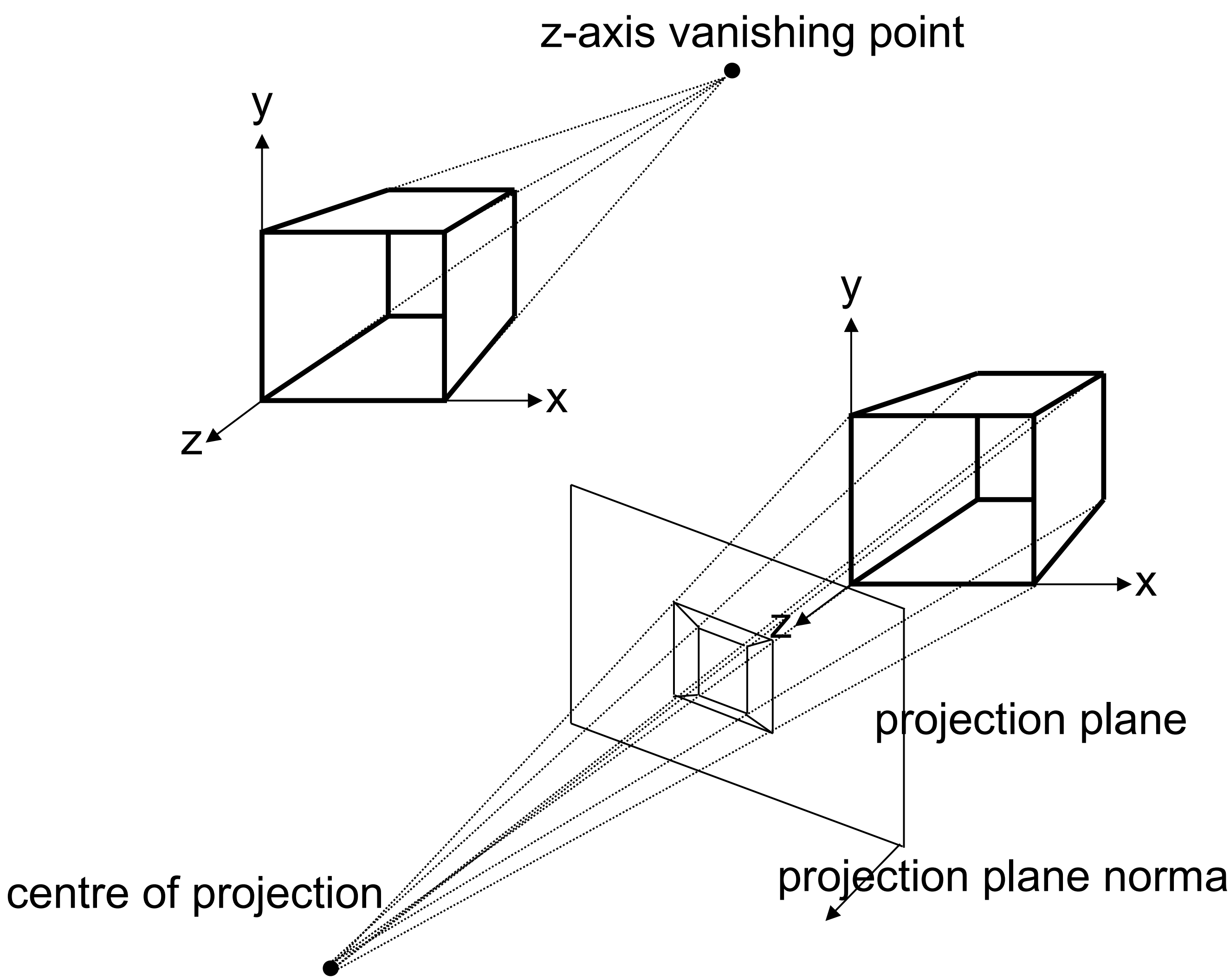
Projections (contd.)



- Since the direction of projection is a vector, and a vector is the difference between two points, it can be calculated as
- $(x,y,z,1)^T - (x',y',z',1)^T = (a,b,c,0)^T$.
- The above diagrams show an effect of perspective projection, namely *perspective foreshortening*; the size of an object is inversely proportional to the distance to the centre of projection.

Perspective Projection

- Perspective projection does not preserve parallel lines. Parallel lines in a scene converge on a ***vanishing point***. If the parallel lines are parallel to one of the principal axes the vanishing point is called an ***axis vanishing point***.

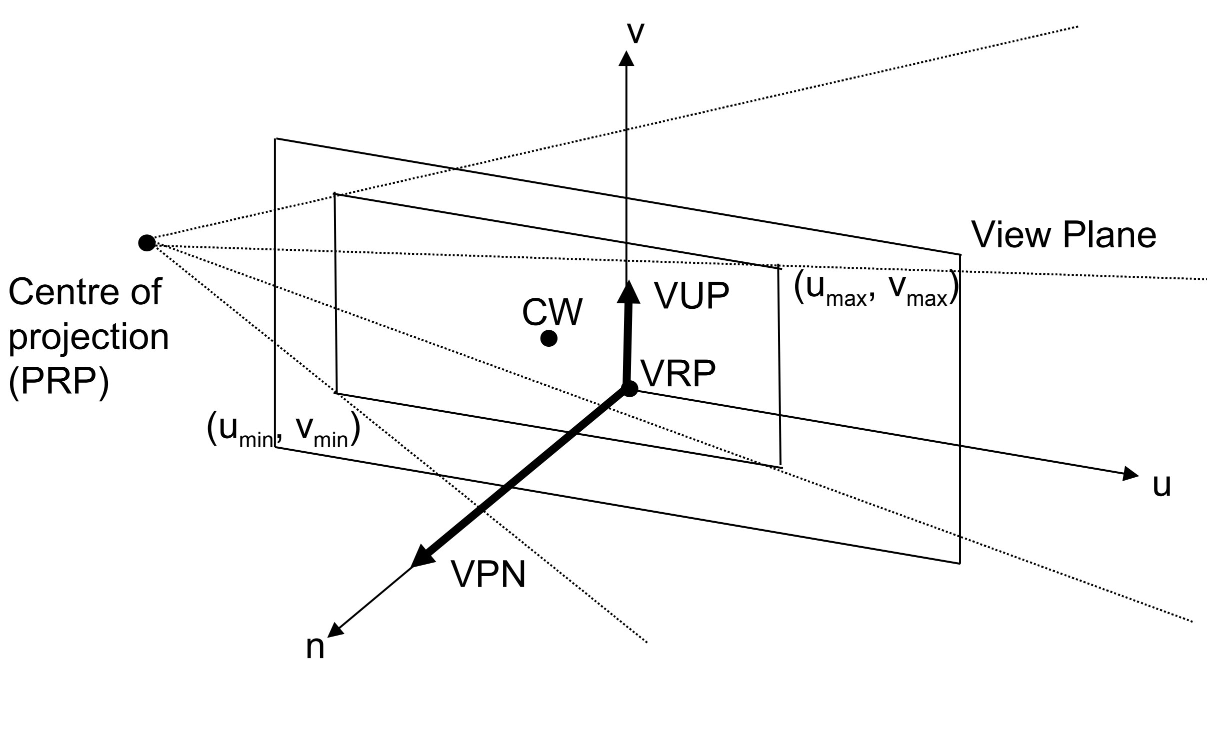


Parallel Projection

- There are different types of parallel projection.
  - Orthographic
    - Direction of projection and projection plane normal are the same.
      - Projection plane normal coincident with a principal axis.
        - » Top (plan)
        - » Front elevation
        - » Side elevation
      - Projection plane normal not coincident with a principal axis. (Axonometric)
        - » Isometric – all principal axes equally foreshortened.
  - Oblique
    - Cabinet
      - Angle between direction of projection and projection plane normal is arctan(2) = 63.4°. This make lines perpendicular to the projection plane project to half their length. More natural look.
    - Cavalier
      - Angle between direction of projection and projection plane normal is 45°.

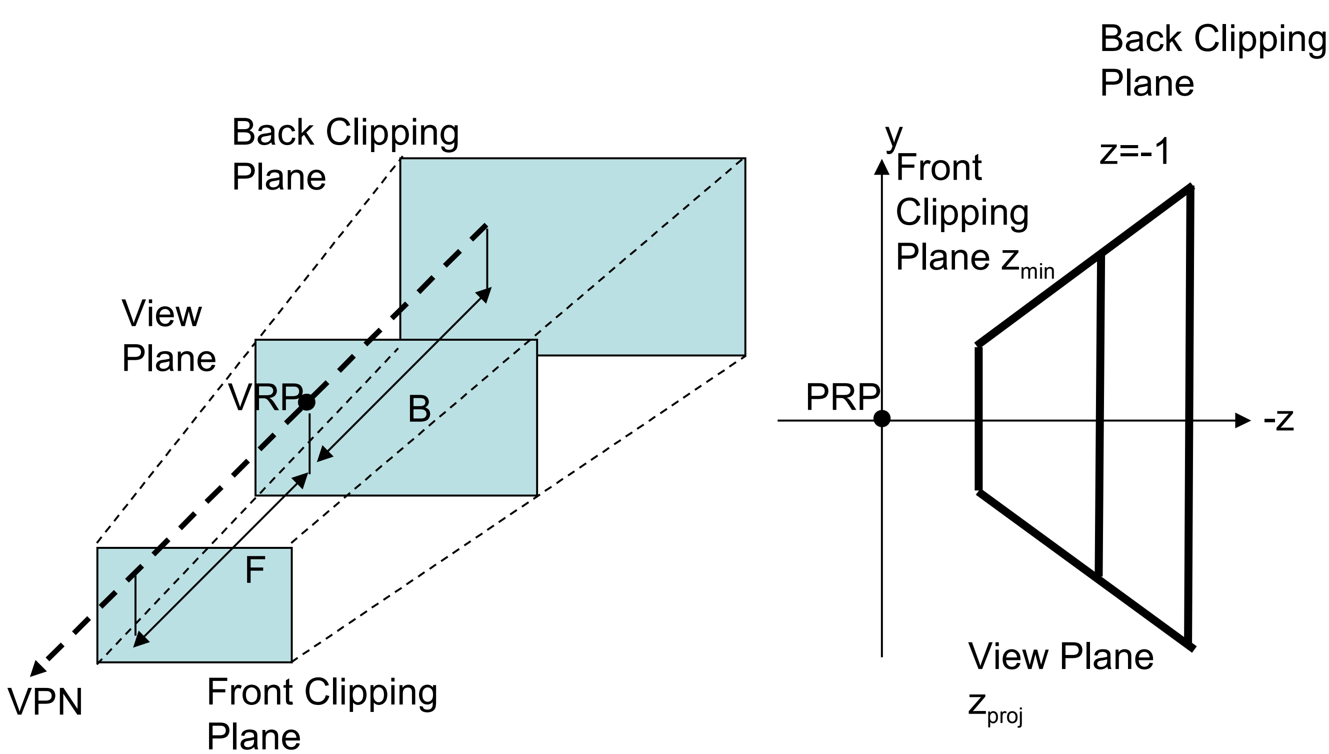
Specifying a 3D view

- The ***projection plane***, also called the ***view plane***, is defined by a ***view reference point*** **(*VRP*)** and the ***view plane normal*** **(*VPN*)**.
- The ***3D viewing reference coordinate*** (***VRC***) system is formed from the ***VPN*** and the ***view up vector*** (***VUP***). The ***VPN*** define the *n*-axis, the projection of the ***VUP*** on the view plane define the *v*-axis and the *u*-axis makes up the right-handed coordinate system.
- The viewing window in the view plane is defined by a ***centre of window*** **(*CW*)** and minimum and maximum *u* and *v* values.



Specifying a 3D view (contd.)

- For a parallel projection the ***PRP*** and the ***direction of projection*** (***DOP***) define the view. The ***DOP*** is a vector from the ***PRP*** to the ***CW***.
- The projectors from the PRP through the minimum and maximum window coordinate define a semi-infinite pyramid view volume (perspective projection) or semi-infinite parallelepiped view volume (parallel projection).
- We usually define a front and back clipping plane.
- This is then mapped to ***normalised projection coordinates*** (***NPC***) with the ***PRP*** at the origin and the back clipping plane mapped to 1.



Implementing Parallel Projection

- We can create a transformation matrix that maps a scene into ***NPC***.
  - Translate ***VRP*** to the origin.
  - Rotate ***VRC*** so that *n*-axis (***VPN***) becomes the *z*-axis, *u*-axis become the *x*-axis and *v*-axis becomes the *y*-axis.
  - Shear such that ***DOP*** is parallel to *z*-axis.
  - Translate and scale to ***NPC***.
- Step 1 is a simple translation

  T(-VRP)

$$T(-VRP)=\begin{bmatrix} 1 & 0 & 0 & -vrp_x \\ 0 & 1 & 0 & -vrp_y \\ 0 & 0 & 1 & -vrp_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Implementing Parallel Projection (contd.)

- Step 2 is the rotation to align axes.
  - Remember that we can think of a rotation matrix as:

$$\begin{bmatrix} R_x & R_y & R_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - ***VPN*** is rotated onto *z*-axis.

$$R_z = \frac{VPN}{\left\| VPN \right\|}$$

  - The *u*-axis which is perpendicular to the *z*-axis and ***VUP*** is rotated on the *y*-axis.

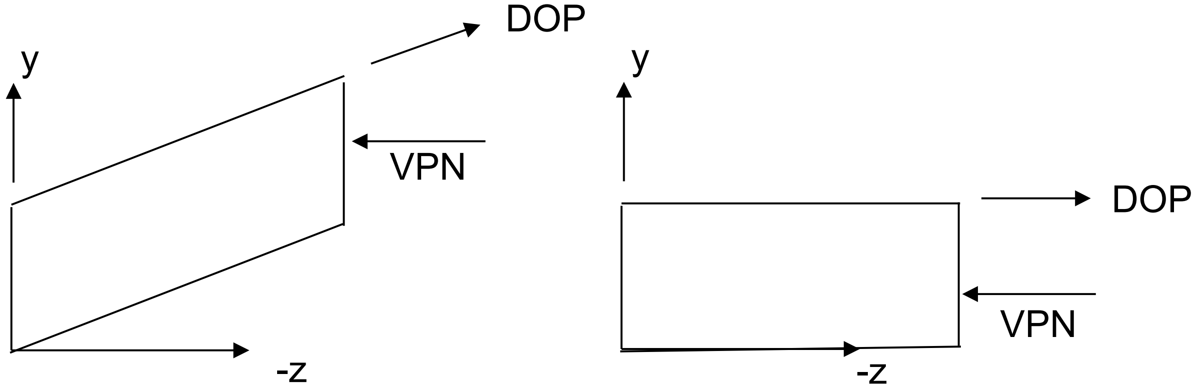
$$R_x = \frac{VUP \times R_z}{\left\| VUP \times R_z \right\|}$$

  - The *v*-axis, which perpendicular to R_x and R_z , is rotated on to the *x*-axis.

$$R_y = R_z \times R_x$$

Implementing Parallel Projection (contd.)

- Step 3 is to shear along the *z*-axis to align the ***DOP*** with *z*-axis while maintaining the ***VPN***.



- Shear Matrix is defined as:

$$SH_{par} = SH\left(shx_{par}, shy_{par}\right) = \begin{bmatrix} 1 & 0 & shx_{par} & 0 \\ 0 & 1 & shy_{par} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- DOP is defined as:

$$\vec{DOP} = \vec{CW} - \vec{PRP}$$

$$\vec{DOP} = \begin{bmatrix} \frac{u_{max}+u_{min}}{2} & \frac{v_{max}+v_{min}}{2} & 0 & 1 \end{bmatrix}^T - \begin{bmatrix} prp_x & prp_y & prp_z & 1 \end{bmatrix}^T$$

Implementing Parallel Projection (contd.)

- After the shearing transform

$$D\vec{O}P' = \begin{bmatrix} 0 & 0 & dop_z & 0 \end{bmatrix} = SH_{par} \cdot D\vec{O}P$$

$$shx_{par} = -\frac{dop_x}{dop_z}, \qquad shy_{par} = -\frac{dop_y}{dop_z}$$

- Now the bounds of the view volume are:

$$u_{min} \le x \le u_{max}, \quad v_{min} \le y \le v_{max}, \quad B \le z \le F$$

- We want translate and scale this volume to ***NPC*** so that the centre of the front clipping plane is at the origin, the ***NPC*** *x* and *y* values range over [-1,1] and the *z* value ranges over [0,1].

$$T_{par} = T\left(-\frac{u_{max} - u_{min}}{2}, -\frac{v_{max} - v_{min}}{2}, -F\right)$$

$$S_{par} = S\left(\frac{2}{u_{max} - u_{min}}, \frac{2}{v_{max} - v_{min}}, \frac{1}{F - B}\right)$$

Implementing Parallel Projection (contd.)

- The resulting transform when all the steps are composed will take an arbitrary view volume and transform it into a canonical parallel perspective view volume.
  - This canonical view is easier to clip against and map to 2D device coordinates.

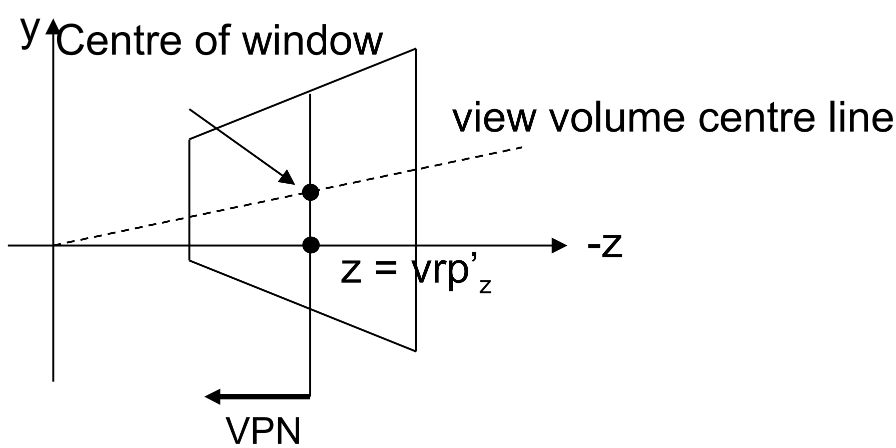
$$N_{par} = S_{par} \cdot T_{par} \cdot SH_{par} \cdot R \cdot T(-VRP)$$

Implementing Perspective Projection

- We can create a transformation matrix that maps a scene into ***NPC*** using perspective project.
  - Translate ***VRP*** to the origin.
  - Rotate ***VRC*** so that *n*-axis (***VPN***) becomes the *z*-axis, *u*-axis become the *x*-axis and *v*-axis becomes the *y*-axis.
  - Translate the centre of projection, given by the ***PRP*** to the origin.
  - Shear such that the centre line of the view volume becomes the *z*-axis.
  - Scale to ***NPC*** so that the back clipping plane is at $z = -1$ and the x and y values range over [-1,1].
- This is similar to the parallel projection implementation.
  - Steps 1 and 2 are identical.

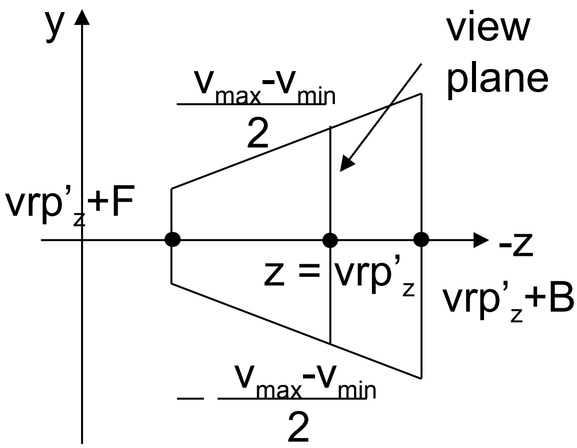
Implementing Perspective Projection (contd.)

- Step 3 translates the centre of projection (***COP***) to the origin. The ***COP*** is defined with respect to the ***VRC*** by the ***PRP***. Since steps 1 and 2 effectively converted the viewing-coordinates into world coordinates, the required translation is simply T(-***PRP***).

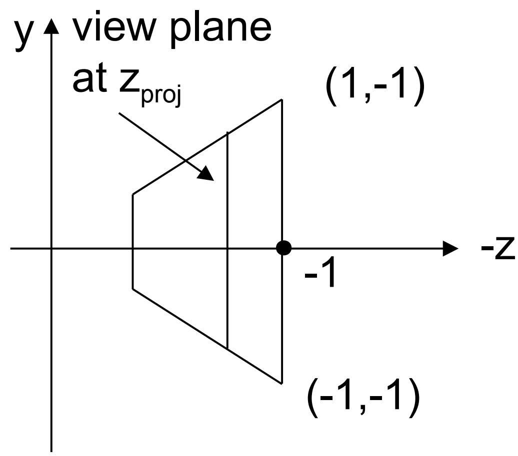


- Step 4 shears the view volume centre line onto the *z*-axis. The direction of the shear is the origin to the centre of the window. Since ***PRP*** is mapped to the origin, the direction of the shear is ***CW*** – ***PRP***, the exact same as for parallel projection.

Implementing Perspective Projection (contd.)



Before scaling



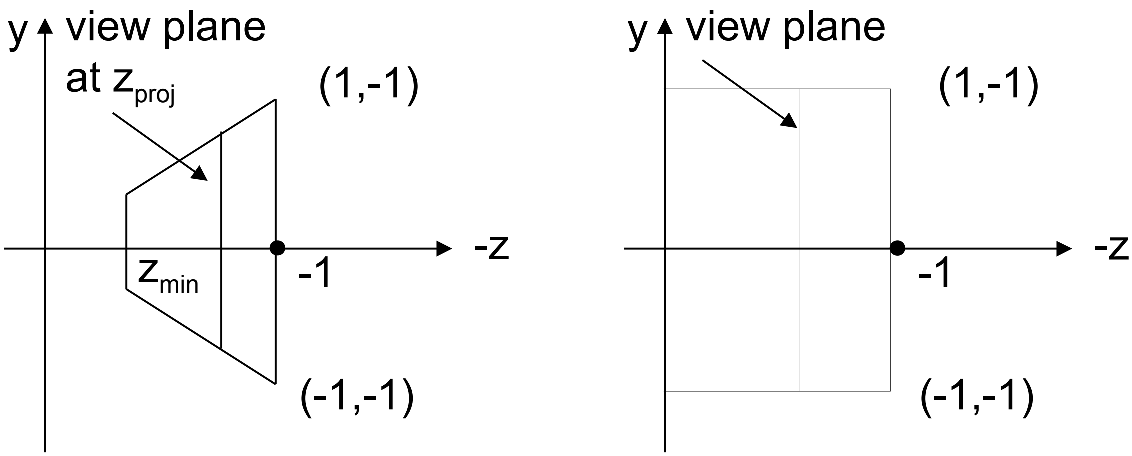
After scaling

- Since the shear operation did not affect the *z* axis, vpr'_z, the result of steps 1 to 4, must be equal to $-prp_n$.
- Scaling the *x* and *y* axes so that the slope of the view volume is unity; and then scaling all axes equally (to preserve slope) so that the back clipping plane is at -1 yields:

$$s_y^1 \cdot \frac{\frac{v_{max}-v_{min}}{2}}{vpr'_z} = 1 \Rightarrow s_y^1 = \frac{2\text{vpr}'_z}{v_{max}-v_{min}}$$

$$s_y = s_y^1 \cdot \frac{1}{vpr'_z + B}$$

Implementing Perspective Projection (contd.)



Finally we have to transform the truncated pyramid on the left to the rectangular view volume on the right.

This is carried out by the matrix below

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{1}{(1+z_{min})} & \dfrac{-z_{min}}{(1+z_{min})} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Implementing Perspective Projection (contd.)

$$S_{per} = S\left(\frac{2\,prp_n}{(u_{max}-u_{min})(prp_n+B)}, \frac{2\,prp_n}{(v_{max}-v_{min})(prp_n+B)}, \frac{-1}{-prp_n}\right)$$

- Hence the normalising perspective projection transform is:

$$N_{per} = S_{per} \cdot SH_{par} \cdot T(-PRP) \cdot R \cdot T(-VRP)$$

- Now that the view volume has be transformed to a canonical space we can clip to the boundaries of the canonical space.

Implementing Perspective Projection (contd.)

$$S_{per} = S\left(\frac{2\,prp_n}{(u_{max}-u_{min})(prp_n+B)}, \frac{2\,prp_n}{(v_{max}-v_{min})(prp_n+B)}, \frac{-1}{-prp_n}\right)$$

- Hence the normalising perspective projection transform is:

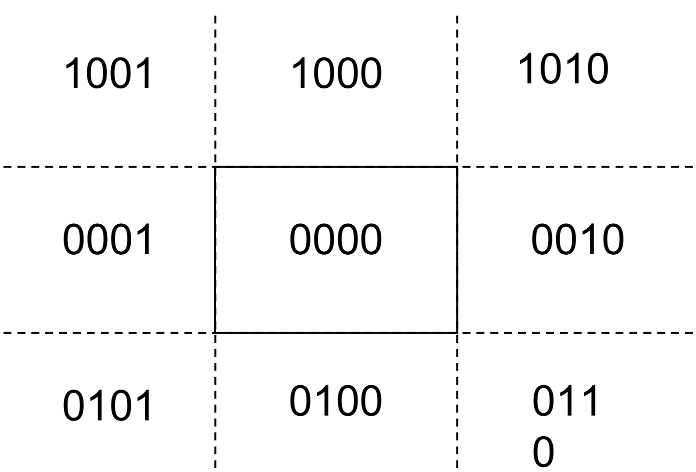
$$N_{per} = S_{per} \cdot SH_{par} \cdot T(-PRP) \cdot R \cdot T(-VRP)$$

- Now that the view volume has be transformed to a canonical space we can clip to the boundaries of the canonical space.

2D Clipping

- The Cohen-Sutherland line clipping algorithm clips lines to rectangular clipping regions.
- The clipping region is defined by 4 clipping planes.
- The 2D plane is divided into regions, each with its own ***outcode***.
- The endpoints of each line to be clipped are assigned a region outcode depending on their location.

Region outcodes



1st bit set if $y > y_{max}$

2nd bit set if $y < y_{min}$

3rd bit set if $x > x_{max}$

4th bit set if $x < x_{min}$

Cohen-Sutherland line clipping algorithm

- If both endpoints' outcodes are 0000 then the line is trivially accepted and rendered as is.
- If a **bitwise and** of both endpoints is not zero then the line can be trivially rejected and not rendered at all.
- If the line cannot be trivially accepted or rejected it may enter the clipping region.
  - Divide the line in 2 based on where it intersects with one of the clipping planes.
    - The segment outside the clipping plane can be discarded.
    - The remaining segment's new endpoint is assigned an outcode and it is tested.
  - The clipping planes can be tested in any order, but the order must be the same each time the algorithm is used.
    - We will use the order of our outcode bits: top, bottom, right, left.

Cohen-Sutherland line clipping algorithm (contd.)

```
Class CohenSutherland
{
  public static final byte TOP = 0x1;
  public static final byte BOTTOM = 0x2;
  public static final byte RIGHT = 0x4;
  public static final byte LEFT = 0x8;

  void clip2D (float x0, float y0, float x1, float y1, float xmin, float xmax,
                    float ymin, float ymax)
  {
    byte outcode0, outcode1, outcodeOut;
    boolean accept = false, done = false;
    float x, y;

    outcode0 = computeOutcode (x0, y0, xmin, xmax, ymin, ymax);
    outcode1 = computeOutcode (x1, y1, xmin, xmax, ymin, ymax);

    do
    {
      if (!(outcode0 | outcode1))
        {
        accept = true; done = true;                         // trivial accept
        }
      else if (outcode0 & outcode1)
        {
        done = true;                                         // trivial reject
        }
```

Cohen-Sutherland line clipping algorithm (contd.)

```
else
  {
  // calculate intersection with a clipping plane using
  // y = y0 +slope*(x-x0) and x = x0 + (1/slope)*(y-y0)

  outcodeOut = outcode0 ? Outcode0 : outcode1;

  if (outcodeOut & TOP)
    {
    x = x0 + (x1-x0)*(ymax-y0)/(y1-y0);
    y = ymax;
    }
  else if (outcodeOut & BOTTOM)
    {
    x = x0 + (x1-x0)*(ymin-y0)/(y1-y0);
    y = ymin;
    }
  else if (outcodeOut & RIGHT)
    {
    y = y0 + (y1-y0)*(xmax-x0)/(x1-x0);
    x = xmax;
    }
  else if (outcodeOut & LEFT)
    {
    y = y0 + (y1-y0)*(xmin-x0)/(x1-x0);
    x = xmin;
    }
  }
```

Cohen-Sutherland line clipping algorithm (contd.)

```
    if (outcodeOut == outcode0)
      {
      x0 = x; y0 = y; outcode0 = ComputeOutcode (x0, y0, xmin, xmax,
     ymin, ymax);
      }
    else
      {
      x1 = x; y1 = y;
      outcode1 = ComputeOutcode (x1, y1, xmin, xmax,
                                                ymin, ymax);
      }
  } while (done == false);

  if (accept)
    {
    DrawLine (x0, y0, x1, y1);
    }
}


byte outcode ()
{
  byte outcode = 0;

  if (y > ymax)
    outcode = outcode | TOP;
  else if (y < ymin)
    outcode = outcode | BOTTOM;

  if (x > xmax)
    outcode = outcode | RIGHT;
  else if (x < xmin)
    outcode = outcode | LEFT;

  return outcode;
}

}
```

3D Clipping

- The Cohen-Sutherland algorithm is easily extended to 3D.
  - Normalised parallel projection
    - The outcode uses 6 bits which indicate:
      - Bit 1: point above view volume: y > 1
      - Bit 2: point below view volume: y < -1
      - Bit 3: point right of view volume: x > 1
      - Bit 4: point left of view volume: x <-1
      - Bit 5: point is behind view volume: z< -1
      - Bit 6: point is in front of view volume: z > 0
    - The calculation of the intersections between the clipped lines and clipping planes uses parametric line equations.

$$\left.\begin{aligned} x &= x_0 + t(x_1 - x_0) \\ y &= y_0 + t(y_1 - y_0) \\ z &= z_0 + t(z_1 - z_0) \end{aligned}\right\} \quad 0 \le t \le 1$$

- For a given clipping plane substitute the appropriate *x*, *y* or *z* value, solve for *t* and use the remaining formulae to find the missing values.

3D Clipping (contd.)

- Normalised perspective projection
  - The outcode uses 6 bits which indicate:
    - Bit 1: point above view volume: y > -z
    - Bit 2: point below view volume: y < z
    - Bit 3: point right of view volume: x > -z
    - Bit 4: point left of view volume: x < z
    - Bit 5: point is behind view volume: z< -1
    - Bit 6: point is in front of view volume: z > z_{min}
  - The calculation of the intersections between the clipped lines and clipping planes is similar to parallel projection. For example the intersection with the *y* = *z* plane

$$y = z \Rightarrow y_0 + t(y_1 - y_0) = z_0 + t(z_1 - z_0)$$

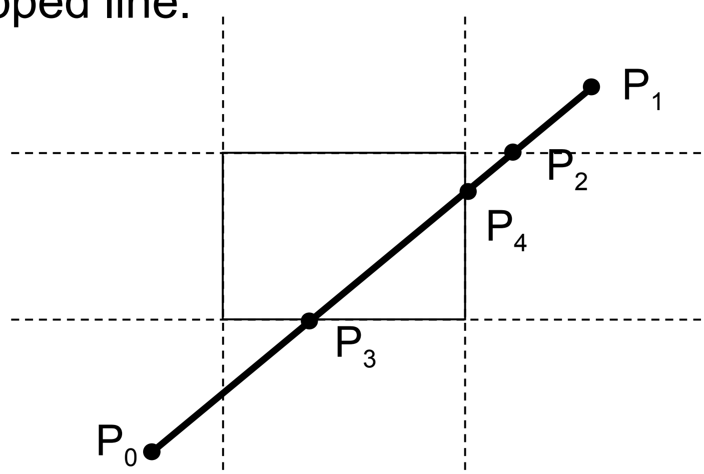
$$t = \frac{z_0 - y_0}{(y_1 - y_0) - (z_1 - z_0)}$$

$$x = x_0 + \frac{(x_1 - x_0)(z_0 - y_0)}{(y_1 - y_0) - (z_1 - z_0)}, \quad y = y_0 + \frac{(y_1 - y_0)(z_0 - y_0)}{(y_1 - y_0) - (z_1 - z_0)}$$

- The intersections with other planes can be calculated similarly.

Other 3D clipping algorithms

- Other algorithms, such as Cyrus-Beck and Liang-Barsky, are more efficient the Cohen-Sutherland.
  - Cohen-Sutherland calculates the *x*, *y* and *z* values at each intersection with a clipping plane.
    - Only 2 of these values are required to draw the clipped line.



- The other 2 algorithms use the parametric line equation and the angle the line to clipped makes with each clipping plane.
  - At each clipping plane only *t* needs to be calculated.
  - The *t* values can identify the endpoints of the clipped line.
  - Liang-Barsky has a more efficient trivial rejection test.