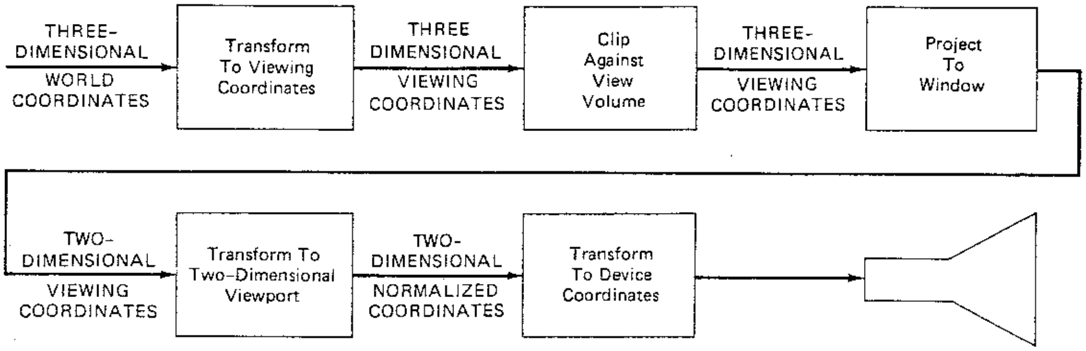
IMPLEMENTATION OF VIEWING
(Sections 12-3 to 12-6 in *Computer Graphics*)

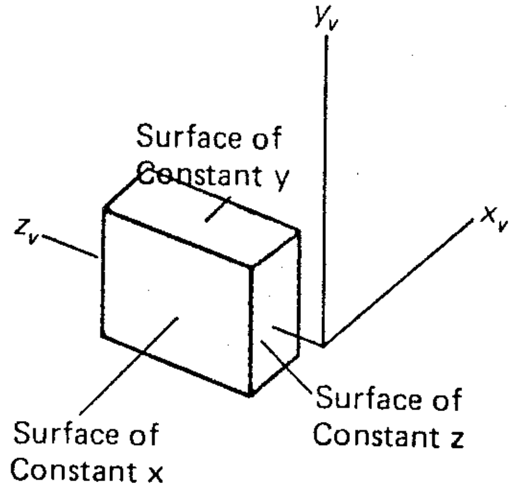
- Viewing Operations
- Hardware Implementations
- Programming Three-dimensional Views
- Extensions to the Viewing Pipeline

Viewing Operations

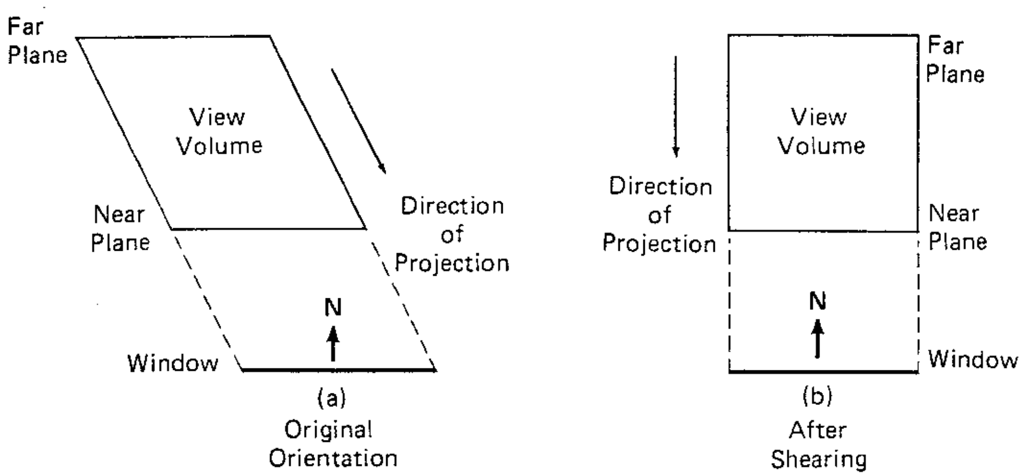


normalized view volumes

- near and far planes have constant z values, making clipping easy
- the four sides of a view volume can have arbitrary orientations, making clipping difficult
  - clipping against a regular parallelepiped (produced by an orthographic parallel projection) is easy

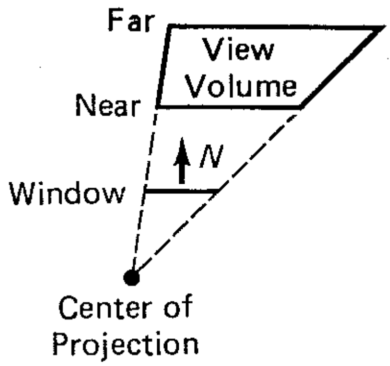


  - the view volume of an oblique parallel projections is sheared to simplify clipping

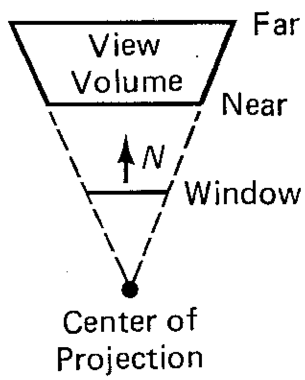


normalized view volumes, continued

- the view volume of a perspective projection is sheared and scaled to produce a rectangular parallelepiped
  - shear in x and y to bring the center of projection onto a line normal to the center of the window



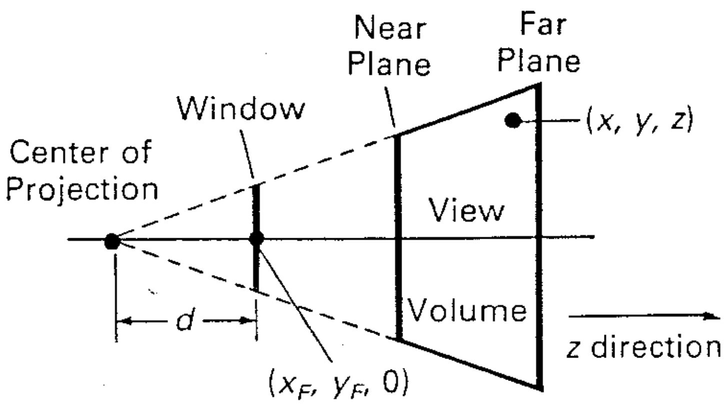
Original
Orientation
(a)



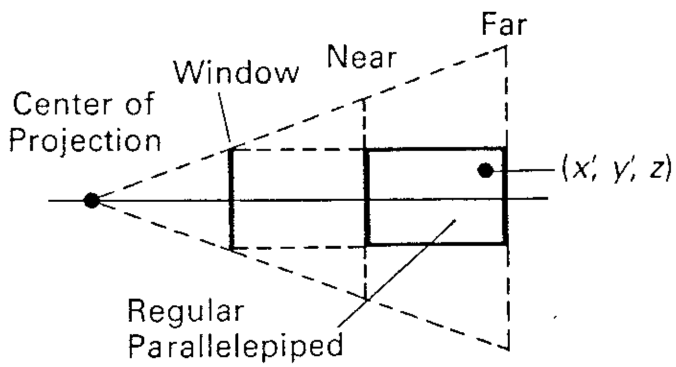
After
Transformation
(b)

normalized view volumes, continued

- scale the sides of the frustrum to the rectangular sides of a regular parallelepiped



(a)



(b)

normalized view volumes, continued

- scaling is inversely proportional to the distance from the window
  S = d/(z + d)

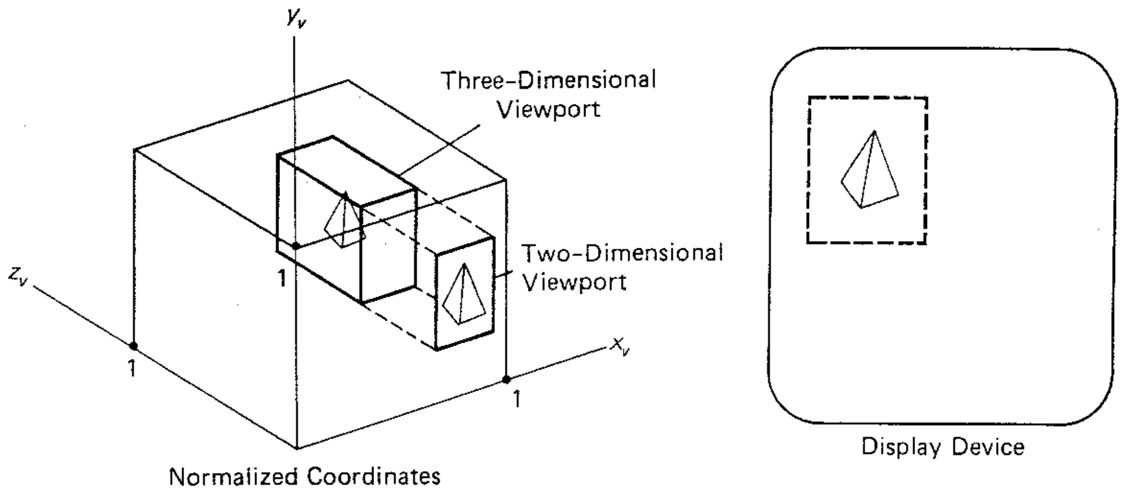
$$\begin{bmatrix} S & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & 1 & 0 \\ (1 - S)x_{\mathrm{F}} & (1 - S)y_{\mathrm{F}} & 0 & 1 \end{bmatrix}$$

- essentially, this is the perspective transformation
  - **x and y clipping and projection now consist in**
    - rejecting points beyond the far plane
    - rejecting points in front of the near plane
    - dropping the z coordinate

- **z, and therefore S, may be different for each point**

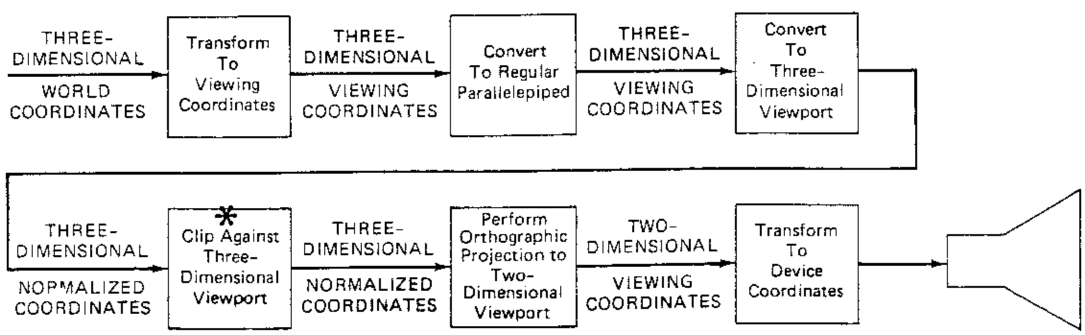
achieving more efficiency

- concatenate matrices
  - mapping from world to viewing coordinates
  - converting the view volume of an oblique parallel projection to a regular parallelepiped
  - create a normalized three-dimensional viewport
  - clip to the viewport
  - convert to device coordinates for display



or

- perform the window-to-viewport mapping before clipping



* cannot be represented by a matrix

three-dimensional window-to-viewport mapping

- similar to two-dimensional window-to-viewport mapping

$$\begin{bmatrix} D_x & 0 & 0 & 0 \\ 0 & D_y & 0 & 0 \\ 0 & 0 & D_z & 0 \\ K_x & K_y & K_z & 1 \end{bmatrix}$$

where

$$D_x = \frac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}}$$

$$D_y = \frac{yv_{\max} - yv_{\min}}{yw_{\max} - yw_{\min}}$$

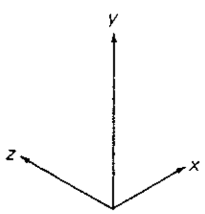
$$D_z = \frac{zv_{\max} - zv_{\min}}{d_f - d_n}$$

and

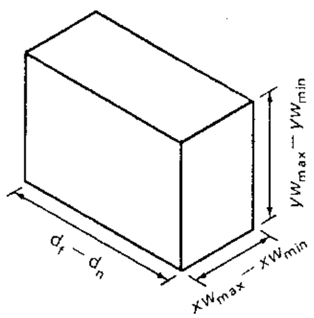
$$K_x = xv_{\min} - xw_{\min} \cdot D_x$$

$$K_y = yv_{\min} - yw_{\min} \cdot D_y$$

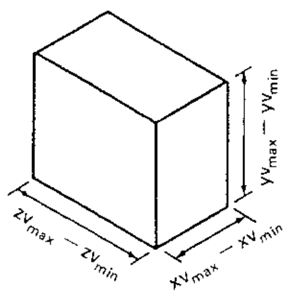
$$K_z = zv_{\min} - d_n \cdot D_z$$



Coordinate Axes Orientation



View Volume Defined by Window Coordinates and Near and Far Planes



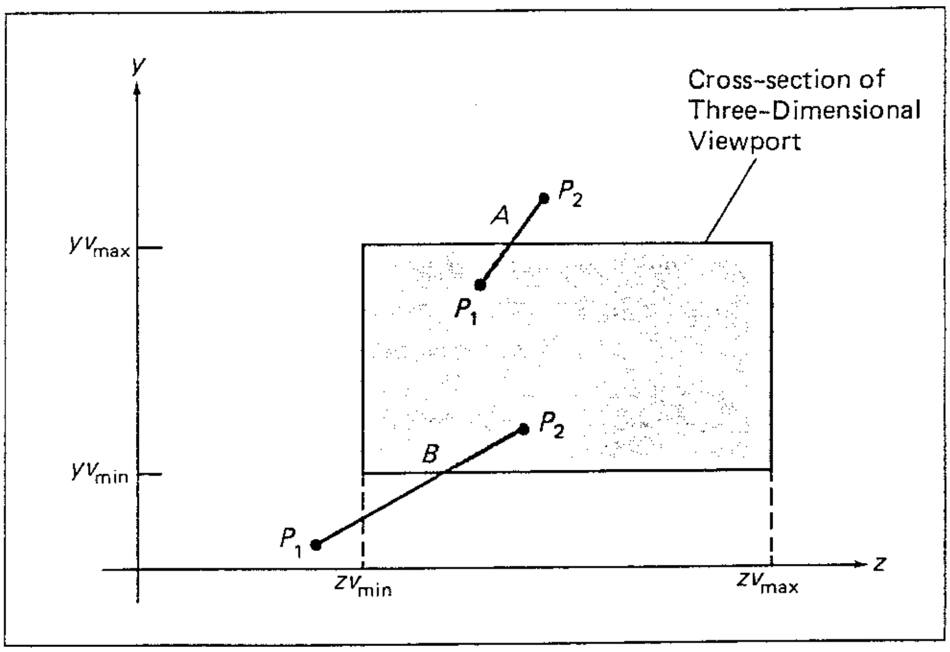
Three-Dimensional Viewport

clipping against a normalized view volume

- **extend region codes**

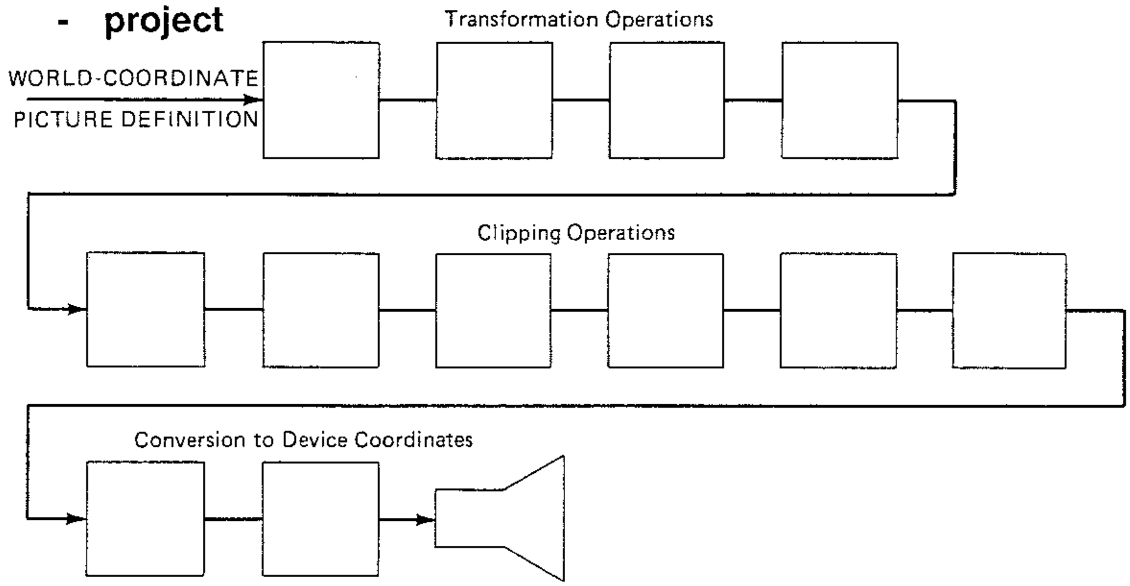
bit 1 = 1	if $x < xv_{\min}$ (left)
bit 2 = 1	if $x > xv_{\max}$ (right)
bit 3 = 1	if $y < yv_{\min}$ (below)
bit 4 = 1	if $y > yv_{\max}$ (above)
bit 5 = 1	if $z < zv_{\min}$ (front)
bit 6 = 1	if $z > zv_{\max}$ (back)

- **trivial acceptance**
- **trivial rejection**
- **subdivision**



Hardware Implementations

- chip sets using VLSI circuitry can perform viewing operations
  - transform
  - clip
  - project



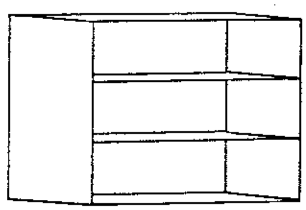
  - pipelined transformations
    - scaling
    - translation
    - rotation
    - projection
  - pipelined clipping
    - one chip for each viewport boundary
  - pipelined coordinate conversion

Programming Three-dimensional Views

- world-to-viewing system coordinates
  create_view_matrix (xo, yo, zo, xn, yn, zn, xv, yv, zv, view_matrix)
  - (xo, yo, zo) is the origin of viewing system coordinates
  - the viewing direction is from the origin of world coordinates to (xn, yn, zn)
  - (xv, yv, zv) specifies the view up vector
- projection parameters
  set_view_representation (view_index, view_matrix, projection_type, xp, yp, zp, xw_min, xw_max, yw_min, yw_max, near, far, xv_min, xv_max, yv_min, yv_max, zv_min, zv_max)
  - view_index identifies the viewing transformation
  - (xp, yp, zp) identified either the direction of projection of the center of projection, depending on projection_type
- viewing transformation selection
  set_view_index (vi)

example

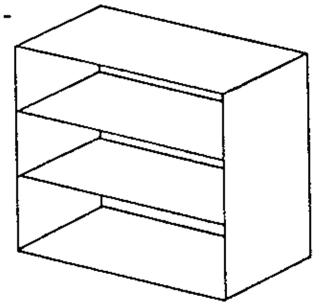
```
type
    matrix = array [1..4,1..4] of real;
    projtype = (parallel, perspective);

procedure bookcase;
    begin
        { Defines bookcase with calls to          }
        { fill_area for the back, sides, top,     }
        { bottom, and 2 shelves.  Bookcase is     }
        { defined in feet, as 3' wide, 4' high    }
        { and 1' deep, with the back, bottom,     }
        { left corner at (0, 0, 0).               }
    end; { bookcase }

procedure establish_views;
    var viewtr1, viewtr2 : matrix;
    begin
        { first view -                            }
```



```
        { view reference point is (-8, 3, 6)      }
        { view plane normal is (-1, 0, 1)         }
        { view up vector is (0, 0, 1)             }
        { Store world-to-viewing transformation   }
        { matrix in viewtr1.                      }

        create_view_matrix ( -8,3,6, -1,0,1, 0,0,1, viewtr1);

        { Use this world-to-viewing transformation }
        { and additional projection parameters to  }
        { fully specify view 2.                    }
        { center of projection is (-12, 3, 12)     }
        { window goes from (2,2) to (8,8)          }
        { put near plane at 10 and far at 12       }
        { viewport is (.5,.5,0) to (1,1,1)         }

        set_view_representation (2, viewtr1, perspective, -12,3,12,
          2, 8, 2, 8, 10, 12, 0.5, 0.5, 0, 1, 1, 1);
```

example (continued)

```
{ second view -                    }
```



```
{ view reference point is now (8, 10, 6)      }
{ view plane normal is now (1, 1, 1)          }
{ Store matrix in viewtr2.                    }

create_view_matrix (8,10,6, 1,1,1, 0,0,1, viewtr2);

{ Use viewtr2 and projection para-            }
{meters to fully specify view 3.              }
{center of projection is now (20,20.20)       }

set_view_representation (3, viewtr2, perspective, 20,20,20,
    2, 8, 2, 8, 10, 12, 0.5, 0.5, 0, 1, 1, 1)
end; {establish_views}

procedure drawcase;
    begin
        establish_views;
        set_view_index (2); { generate view using transform 2 }
        bookcase;
          .
          .
          .
        set_view_ (3); { generate view using transform 3 }
        bookcase
    end; { drawcase }
```

Extensions to the Viewing Pipeline

- operations that may precede the viewing transformation
  - segment transformations
- operations that may follow the viewing transformation
  - image transformations, applied
    to the final, two-dimensional projection

IMPLEMENTATION OF VIEWING

- Viewing Operations
- Hardware Implementations
- Programming Three-dimensional Views
- Extensions to the Viewing Pipeline