

## 5 Geometries

### 5.1 Vector Spaces

#### Vector Spaces

**Definition:**

- Set of vectors  $\mathcal{V}$ .
- Two operations. For  $\vec{v}, \vec{u} \in \mathcal{V}$ :

**Addition:**  $\vec{u} + \vec{v} \in \mathcal{V}$ .

**Scalar multiplication:**  $\alpha\vec{u} \in \mathcal{V}$ , where  $\alpha$  is a member of some field  $\mathbb{F}$ , (i.e.  $\mathbb{R}$ ).

**Axioms:**

**Addition Commutes:**  $\vec{u} + \vec{v} = \vec{v} + \vec{u}$ .

**Addition Associates:**  $(\vec{u} + \vec{v}) + \vec{w} = \vec{u} + (\vec{v} + \vec{w})$ .

**Scalar Multiplication Distributes:**  $\alpha(\vec{u} + \vec{v}) = \alpha\vec{u} + \alpha\vec{v}$ .

**Unique Zero Element:**  $\vec{0} + \vec{u} = \vec{u}$ .

**Field Unit Element:**  $1\vec{u} = \vec{u}$ .

**Span:**

- Suppose  $\mathcal{B} = \{\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n\}$ .
- $\mathcal{B}$  **spans**  $\mathcal{V}$  iff any  $\vec{v} \in \mathcal{V}$  can be written as  $\vec{v} = \sum_{i=1}^n \alpha_i \vec{v}_i$ .
- $\sum_{i=1}^n \alpha_i \vec{v}_i$  is a **linear combination** of the vectors in  $\mathcal{B}$ .

**Basis:**

- Any minimal spanning set is a basis.
- All bases are the same size.

**Dimension:**

- The number of vectors in any basis.
- We will work in 2 and 3 dimensional spaces.

### 5.2 Affine Spaces

#### Affine Space

**Definition:** Set of Vectors  $\mathcal{V}$  and a Set of Points  $\mathcal{P}$

- Vectors  $\mathcal{V}$  form a **vector space**.
- Points can be combined with vectors to make new points:  
 $P + \vec{v} \Rightarrow Q$  with  $P, Q \in \mathcal{P}$  and  $\vec{v} \in \mathcal{V}$ .

**Frame:** An affine extension of a basis:

Requires a vector basis plus a point  $\mathcal{O}$  (the **origin**):

$$F = (\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n, \mathcal{O})$$

**Dimension:** The dimension of an affine space is the same as that of  $\mathcal{V}$ .

### 5.3 Euclidean Spaces

#### Euclidean Spaces

**Metric Space:** Any space with a **distance metric**  $d(P, Q)$  defined on its elements.

**Distance Metric:**

- Metric  $d(P, Q)$  must satisfy the following axioms:
  1.  $d(P, Q) \geq 0$
  2.  $d(P, Q) = 0$  iff  $P = Q$ .
  3.  $d(P, Q) = d(Q, P)$ .
  4.  $d(P, Q) \leq d(P, R) + d(R, Q)$ .
- Distance is intrinsic to the space, and *not* a property of the frame.

**Euclidean Space:** Metric is based on a dot (inner) product:

$$d^2(P, Q) = (P - Q) \cdot (P - Q)$$

**Dot product:**

$$\begin{aligned} (\vec{u} + \vec{v}) \cdot \vec{w} &= \vec{u} \cdot \vec{w} + \vec{v} \cdot \vec{w}, \\ \alpha(\vec{u} \cdot \vec{v}) &= (\alpha\vec{u}) \cdot \vec{v} \\ &= \vec{u} \cdot (\alpha\vec{v}) \\ \vec{u} \cdot \vec{v} &= \vec{v} \cdot \vec{u}. \end{aligned}$$

**Norm:**

$$|\vec{u}| = \sqrt{\vec{u} \cdot \vec{u}}.$$

**Angles:**

$$\cos(\angle \vec{u}\vec{v}) = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}||\vec{v}|}$$

**Perpendicularity:**

$$\vec{u} \cdot \vec{v} = 0 \Rightarrow \vec{u} \perp \vec{v}$$

Perpendicularity is *not* an affine concept! There is no notion of angles in affine space.

### 5.4 Cartesian Space

#### Cartesian Space

**Cartesian Space:** A Euclidean space with an standard **orthonormal frame**  $(\vec{i}, \vec{j}, \vec{k}, \mathcal{O})$ .

**Orthogonal:**  $\vec{i} \cdot \vec{j} = \vec{j} \cdot \vec{k} = \vec{k} \cdot \vec{i} = 0$ .

**Normal:**  $|\vec{i}| = |\vec{j}| = |\vec{k}| = 1$ .

**Notation:** Specify the **Standard Frame** as  $F_S = (\vec{i}, \vec{j}, \vec{k}, \mathcal{O})$ .

As defined previously, points and vectors are

- Different objects.
- Have different operations.
- Behave differently under transformation.

**Coordinates:** Use an “an extra coordinate”:

- 0 for vectors:  $\vec{v} = (v_x, v_y, v_z, 0)$  means  $\vec{v} = v_x\vec{i} + v_y\vec{j} + v_z\vec{k}$ .
- 1 for points:  $P = (p_x, p_y, p_z, 1)$  means  $P = p_x\vec{i} + p_y\vec{j} + p_z\vec{k} + \mathcal{O}$ .
- Later we’ll see other ways to view the fourth coordinate.
- **Coordinates have no meaning without an associated frame!**
  - Sometimes we’ll omit the extra coordinate ... point or vector by context.
  - Sometimes we may not state the frame ... the Standard Frame is assumed.

## 5.5 Why Vector Spaces Inadequate

- Why not use vector spaces instead of affine spaces?

If we trace the tips of the vectors, then we get curves, etc.

- First problem: no metric

Solution: Add a metric to vector space

- Bigger problem:

- Want to represent objects with small number of “representatives”

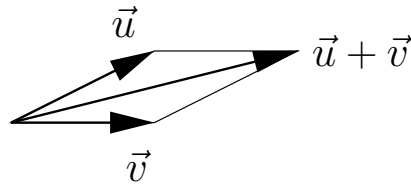
Example: three points to represent a triangle

- Want to translate, rotate, etc., our objects by translating, rotating, etc., the representatives

(vectors don’t translate, so we would have to define translation)

Let  $\vec{u}, \vec{v}$  be representatives

Let  $\vec{u} + \vec{v}$  be on our object



Let  $T(\vec{u})$  be translation by  $\vec{t}$  defined as  $T(\vec{u}) = \vec{u} + \vec{t}$ .

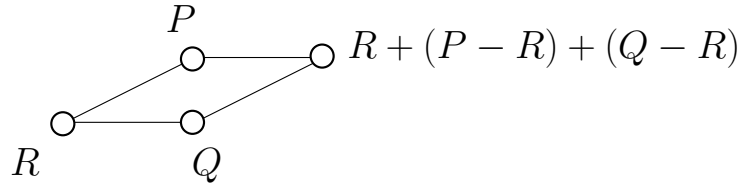
Then

$$\begin{aligned}
 T(\vec{u} + \vec{v}) &= \vec{u} + \vec{v} + \vec{t} \\
 &\neq T(\vec{u}) + T(\vec{v}) \\
 &= \vec{u} + \vec{v} + 2\vec{t}
 \end{aligned}$$

**Note: this definition of translation only used on this slide! Normally, translation is identity on vectors!**

Let  $P$  and  $Q$  be on our object

Let  $R$  be a third point, and  $R + (P - R) + (Q - R)$  be on our object



Let  $T(\vec{u})$  be translation by  $\vec{t}$  (i.e.,  $T(P) = P + \vec{t}$ ,  $T(\vec{v}) = \vec{v}$ ).

Then

$$\begin{aligned} T(R + (P - R) + (Q - R)) &= R + (P - R) + (Q - R) + \vec{t} \\ &= T(R) + T(P - R) + T(Q - R) \end{aligned}$$

## 5.6 Summary of Geometric Spaces

Space	Objects	Operators
Vector Space	Vector	$\vec{u} + \vec{v}$ , $\alpha\vec{v}$
Affine Space	Vector, Point	$\vec{u} + \vec{v}$ , $\alpha\vec{v}$ , $P + \vec{v}$
Euclidean Space	Vector Point	$\vec{u} + \vec{v}$ , $\alpha\vec{v}$ , $P + \vec{v}$ , $\vec{u} \cdot \vec{v}$ , $d(P, Q)$ Question: where does the cross
Cartesian Space	Vector, Point, O.N. Frame	$\vec{u} + \vec{v}$ , $\alpha\vec{v}$ , $P + \vec{v}$ , $\vec{u} \cdot \vec{v}$ , $d(P, Q)$ , $\vec{i}, \vec{j}, \vec{k}$ , $\mathcal{O}$ product enter?

(Readings: Watt: 1.1, 1.2. White book: Appendix A. Hearn and Baker: A-2, A-3. )

## 6 Affine Geometry and Transformations

### 6.1 Linear Combinations

#### Linear Combinations

**Vectors:**

$$\mathcal{V} = \{\vec{u}\}, \quad \vec{u} + \vec{v}, \quad \alpha\vec{u}$$

**By Extension:**

$$\sum_i \alpha_i \vec{u}_i \quad (\text{no restriction on } \alpha_i)$$

**Linear Transformations:**

$$T(\vec{u} + \vec{v}) = T(\vec{u}) + T(\vec{v}), \quad T(\alpha\vec{u}) = \alpha T(\vec{u})$$

**By Extension:**

$$T\left(\sum_i \alpha_i \vec{u}_i\right) = \sum_i \alpha_i T(\vec{u}_i)$$

**Points:**

$$\mathcal{P} = \{P\}, \quad P + \vec{u} = Q$$

**By Extension...**

### 6.2 Affine Combinations

#### Affine Combinations

**Define Point Subtraction:**

$$Q - P \text{ means } \vec{v} \in \mathcal{V} \text{ such that } Q = P + \vec{v} \text{ for } P, Q \in \mathcal{P}.$$

**By Extension:**

$$\sum \alpha_i P_i \text{ is a \textbf{vector} iff } \sum \alpha_i = 0$$

**Define Point Blending:**

$$Q = (1 - a)Q_1 + aQ_2 \text{ means } Q = Q_1 + a(Q_2 - Q_1) \text{ with } Q \in \mathcal{P}$$

**Alternatively:**

$$\text{we may write } Q = a_1Q_1 + a_2Q_2 \text{ where } a_1 + a_2 = 1.$$

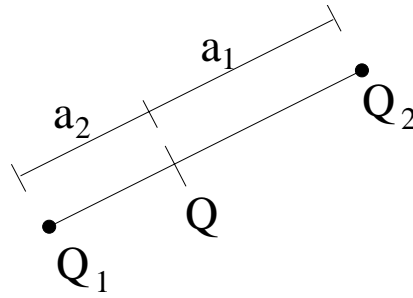
**By Extension:**

$$\sum a_i P_i \text{ is a \textbf{point} iff } \sum a_i = 1$$

**Geometrically:**

- The following ratio holds for  $Q = a_1Q_1 + a_2Q_2$

$$\frac{|Q - Q_1|}{|Q - Q_2|} = \frac{a_2}{a_1} \quad (a_1 + a_2 = 1)$$



- If  $Q$  breaks the line segment  $\overline{Q_1Q_2}$  into the ratio  $b_2 : b_1$  then

$$Q = \frac{b_1Q_1 + b_2Q_2}{b_1 + b_2} \quad (b_1 + b_2 \neq 0)$$

### Legal vector combinations:

Vectors can be formed into any combinations  $\sum_i \alpha_i \vec{u}_i$  (a “linear combination”).

### Legal point combinations:

Points can be formed into combinations  $\sum_i a_i P_i$  iff

- The coefficients sum to 1: The result is a point (an “affine combination”).
- The coefficients sum to 0: The result is a vector (a “vector combination”).

### Parametric Line Equation:

has geometric meaning (in an affine sense):

$$\begin{aligned} L(t) &= A + t(B - A) \\ &= (1 - t)A + tB \end{aligned}$$

The weights  $t$  and  $(1 - t)$  create an affine combination.

The result is a point (on the line).

### Parametric Ray Equation:

Same as above, but  $t \geq 0$ . Will write as

$$R(t) = A + t\vec{d}$$

Where  $A$  is the point of origin and  $\vec{d}$  is the direction of the ray. Used in ray-tracing.

(Readings: White book, Appendix A)

## 6.3 Affine Transformations

### Affine Transformations

Let  $T : \mathcal{A}_1 \mapsto \mathcal{A}_2$ , where  $\mathcal{A}_1$  and  $\mathcal{A}_2$  are affine spaces.

Then  $T$  is said to be an affine transformation if:

- $T$  maps vectors to vectors and points to points

- $T$  is a linear transformation on the vectors
- $T(P + \vec{u}) = T(P) + T(\vec{u})$

**By Extension:**

$T$  preserves affine combinations on the points:

$$T(a_1Q_1 + \cdots + a_nQ_n) = a_1T(Q_1) + \cdots + a_nT(Q_n),$$

If  $\sum a_i = 1$ , result is a point.

If  $\sum a_i = 0$ , result is a vector.

**Observations:**

- Affine transformations map lines to lines:

$$T((1-t)P_0 + tP_1) = (1-t)T(P_0) + tT(P_1)$$

- Affine transformations map rays to rays:

$$T(A + t\vec{d}) = T(A) + tT(\vec{d})$$

- Affine transformations preserve ratios of distance along a line  
(converse is also true: preserves ratios of such distances  $\Rightarrow$  affine).
- Absolute distances or angles may not be preserved.  
Absolute distances and angles are not affine concepts...

**Examples:**

- translations
- rotations
- scales
- shears
- reflections

Translations and rotations called *rigid body motions*.

**Affine vs Linear**

Which is a larger class of transformations: Affine or Linear?

- All affine transformations are linear transformations on vectors.
- Consider identity transformation on vectors.  
There is one such linear transformation.
- Consider Translations:
  - Identity transformation on vectors
  - Infinite number different ones (based on effect on points)

Thus, there are an infinite number of affine transformations that are identity transformation on vectors.

- What makes affine transformations a bigger class than linear transformation is their translational behaviour on points.

## Extending Affine Transformations to Vectors

Suppose we only have  $T$  defined on points.

**Define**  $T(\vec{v})$  as follows:

- There exist points  $Q$  and  $R$  such that  $\vec{v} = Q - R$ .
- Define  $T(\vec{v})$  to be  $T(Q) - T(R)$ .

Note that  $Q$  and  $R$  are not unique.

The definition works for  $P + \vec{v}$ :

$$\begin{aligned} T(P + \vec{v}) &= T(P + Q - R) \\ &= T(P) + T(Q) - T(R) \\ &= T(P) + T(\vec{v}) \end{aligned}$$

Can now show that the definition is well defined.

**Theorem:** Affine transformations map parallel lines to parallel lines.

## Mapping Through an Affine Transformation

Need to pick a numerical representation; use *coordinates*:

**Let**  $\mathcal{A}$  and  $\mathcal{B}$  be affine spaces.

- Let  $T : \mathcal{A} \mapsto \mathcal{B}$  be an affine transformation.
- Let  $F_{\mathcal{A}} = (\vec{v}_1, \vec{v}_2, \mathcal{O}_{\mathcal{V}})$  be a frame for  $\mathcal{A}$ .
- Let  $F_{\mathcal{B}} = (\vec{w}_1, \vec{w}_2, \mathcal{O}_{\mathcal{W}})$  be a frame for  $\mathcal{B}$ .
- Let  $P$  be a point in  $\mathcal{A}$  whose *coordinates* relative  $F_{\mathcal{A}}$  are  $(p_1, p_2, 1)$ .

$$(P = p_1\vec{v}_1 + p_2\vec{v}_2 + 1\mathcal{O}_{\mathcal{V}})$$

$\mathcal{O}_{\mathcal{V}}$  and  $\mathcal{O}_{\mathcal{W}}$  are called the *origins* of their respective frames.

**Question:** What are the coordinates  $(p'_1, p'_2, 1)$  of  $T(P)$  relative to the frame  $F_{\mathcal{B}}$ ?

**Fact:** An affine transformation is completely characterized by the image of a frame in the domain:

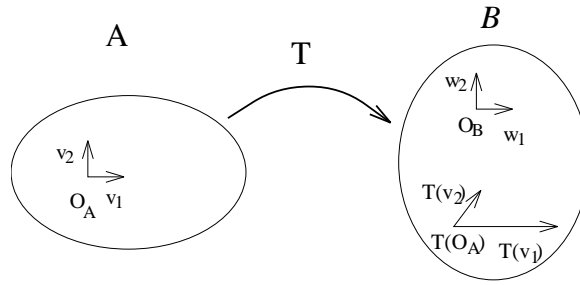
$$\begin{aligned} T(P) &= T(p_1\vec{v}_1 + p_2\vec{v}_2 + \mathcal{O}_{\mathcal{V}}) \\ &= p_1T(\vec{v}_1) + p_2T(\vec{v}_2) + T(\mathcal{O}_{\mathcal{V}}). \end{aligned}$$

If

$$\begin{aligned} T(\vec{v}_1) &= t_{1,1}\vec{w}_1 + t_{2,1}\vec{w}_2 \\ T(\vec{v}_2) &= t_{1,2}\vec{w}_1 + t_{2,2}\vec{w}_2 \\ T(\mathcal{O}_{\mathcal{V}}) &= t_{1,3}\vec{w}_1 + t_{2,3}\vec{w}_2 + \mathcal{O}_{\mathcal{W}} \end{aligned}$$

then we can find  $(p'_1, p'_2, 1)$  by substitution and gathering like terms.





(Readings: White book, Appendix A)

## 6.4 Matrix Representation of Transformations

### Matrix Representation of Transformations

**Represent** points and vectors as  $n \times 1$  matrices. In 2D,

$$P \equiv \mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix}$$

$$\vec{v} \equiv \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ 0 \end{bmatrix}$$

**This is a Shorthand:** Coordinates are specified relative to a frame  $F = (\vec{v}_1, \vec{v}_2, \mathcal{O}_V)$ :

$$P \equiv [\vec{v}_1, \vec{v}_2, \mathcal{O}_V] \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix}$$

$$= F\mathbf{p}.$$

**Technically,**

- Should write the  $\mathbf{p}$  as  $\mathbf{p}_F$ .
- The frame  $F$  should note what space it's in.

**Usually,** we're lazy and let ' $\mathbf{p}$ ' denote both

- The point
- Its matrix representation relative to an understood frame.

**Transformations:**

- $F_A = (\vec{v}_1, \vec{v}_2, \mathcal{O}_V)$  is the frame of the domain,
- $F_B = (\vec{w}_1, \vec{w}_2, \mathcal{O}_W)$  is the frame of the range.

Then ...

$P = \mathbf{p}$  transforms to  $\mathbf{p}' = M_T \mathbf{p}$ .

Can also read this as  $F_A = F_B M_T$ .

$M_T$  is said to be the **matrix representation** of  $T$  relative to  $F_A$  and  $F_B$ .

- First column of  $M_T$  is representation of  $T(\vec{v}_1)$  in  $F_B$ .
- Second column of  $M_T$  is representation of  $T(\vec{v}_2)$  in  $F_B$ .
- Third column of  $M_T$  is representation of  $T(\mathcal{O}_V)$  in  $F_B$ .

## 6.5 Geometric Transformations

### Geometric Transformations

**Construct** matrices for simple geometric transformations.

**Combine** simple transformations into more complex ones.

- Assume that the range and domain frames are the Standard Frame.
- Will begin with 2D, generalize later.

**Translation:** Specified by the vector  $[\Delta x, \Delta y, 0]^T$ :

- A point  $[x, y, 1]^T$  will map to  $[x + \Delta x, y + \Delta y, 1]^T$ .
- A vector will remain unchanged under translation.
- Translation is **NOT** a linear transformation.
- Translation is linear on sums of vectors...

### Matrix representation of translation

- We can create a matrix representation of translation:

$$\overbrace{\begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix}}^{T(\Delta x, \Delta y)} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + \Delta x \\ y + \Delta y \\ 1 \end{bmatrix}$$

- $T(\Delta x, \Delta y)$  will mean the above matrix.
- Note that vectors are unchanged by this matrix.
- Although more expensive to compute than the other version of translation, we prefer this one:
  - Uniform treatment of points and vectors
  - Other transformations will also be in matrix form.

We can compose transformations by matrix multiply. Thus, the composite operation less expensive if translation composed, too.

### Scale about the origin:

Specified by factors  $s_x, s_y \in \mathbb{R}$ .

- Applies to points or vectors, is linear.

- A point  $[x, y, 1]^T$  will map to  $[s_x x, s_y y, 1]^T$ .
- A vector  $[x, y, 0]^T$  will map to  $[s_x x, s_y y, 0]^T$ .
- Matrix representation:

$$\overbrace{\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{S(s_x, s_y)} \begin{bmatrix} x \\ y \\ 0 \text{ or } 1 \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ 0 \text{ or } 1 \end{bmatrix}$$

**Rotation:** Counterclockwise about the origin, by angle  $\theta$ .

- Applies to points or vectors, is linear.
- Matrix representation:

$$\overbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{R(\theta)} \begin{bmatrix} x \\ y \\ 0 \text{ or } 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 0 \text{ or } 1 \end{bmatrix}$$

**Shear:** Intermixes coordinates according to  $\alpha, \beta \in \mathbb{R}$ :

- Applies to points or vectors, is linear.
- Matrix representation:

$$\overbrace{\begin{bmatrix} 1 & \beta & 0 \\ \alpha & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{Sh(\alpha, \beta)} \begin{bmatrix} x \\ y \\ 0 \text{ or } 1 \end{bmatrix} = \begin{bmatrix} x + \beta y \\ \alpha x + y \\ 0 \text{ or } 1 \end{bmatrix}$$

- Easiest to see if we set one of  $\alpha$  or  $\beta$  to zero.

**Reflection:** Through a line.

- Applies to points or vectors, is linear.
- Example: through  $x$ -axis, matrix representation is

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

- See book for other examples

**Note:** Vectors map through all these transformations as we want them to.

(Readings: Hearn and Baker, 5-1, 5-2, and 5-4; Red book, 5.2, 5.3; White book, 5.1, 5.2)

## 6.6 Compositions of Transformations

### Compositions of Transformations

Suppose we want to rotate around an arbitrary point  $P \equiv [x, y, 1]^T$ .

- Could derive a more general transformation matrix ...
- Alternative idea: Compose simple transformations
  1. Translate  $P$  to origin
  2. Rotate around origin
  3. Translate origin back to  $P$
- Suppose  $P = [x_o, y_o, 1]^T$
- The the desired transformation is

$$\begin{aligned}
 & T(x_o, y_o) \circ R(\theta) \circ T(-x_o, -y_o) \\
 &= \begin{bmatrix} 1 & 0 & x_o \\ 0 & 1 & y_o \\ 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} 1 & 0 & -x_o \\ 0 & 1 & -y_o \\ 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} \cos(\theta) & -\sin(\theta) & x_o(1 - \cos(\theta)) + y_o \sin(\theta) \\ \sin(\theta) & \cos(\theta) & y_o(1 - \cos(\theta)) - x_o \sin(\theta) \\ 0 & 0 & 1 \end{bmatrix}
 \end{aligned}$$

- Note where  $P$  maps: to  $P$ .
- Won't compute these matrices analytically;  
Just use the basic transformations,  
and run the matrix multiply numerically.

**Order is important!**

$$\begin{aligned}
 & T(-\Delta x, -\Delta y) \circ T(\Delta x, \Delta y) \circ R(\theta) = R(\theta) \\
 & \neq T(-\Delta x, -\Delta y) \circ R(\theta) \circ T(\Delta x, \Delta y).
 \end{aligned}$$

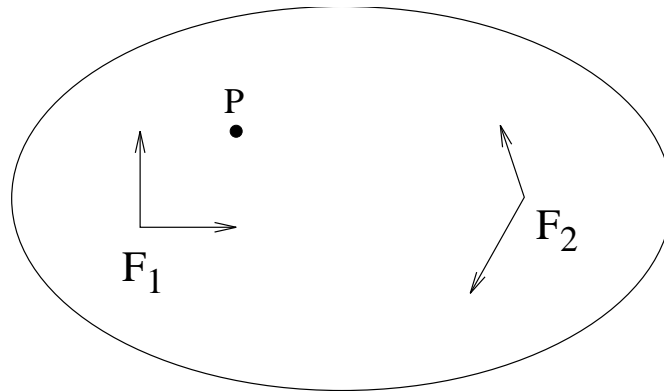
(Readings: Hearn and Baker, Section 5-3; Red book, 5.4; White book, 5.3 )

## 6.7 Change of Basis

### Change of Basis

**Suppose:**

- We have two coordinate frames for a space,  $F_1$  and  $F_2$ ,
- Want to change from coordinates relative to  $F_1$  to coordinates relative to  $F_2$ .



**Know**  $P \equiv \mathbf{p} = [x, y, 1]^T$  relative to  $F_1 = (\vec{w}_1, \vec{w}_2, \mathcal{O}_W)$ .

**Want** the coordinates of  $P$  relative to  $F_2 = (\vec{v}_1, \vec{v}_2, \mathcal{O}_V)$ .

**How** do we get  $f_{i,j}$ ?

- If  $F_2$  is orthonormal:

$$f_{i,j} = \vec{w}_j \cdot \vec{v}_i,$$

$$f_{i,3} = (\mathcal{O}_W - \mathcal{O}_V) \cdot \vec{v}_i.$$

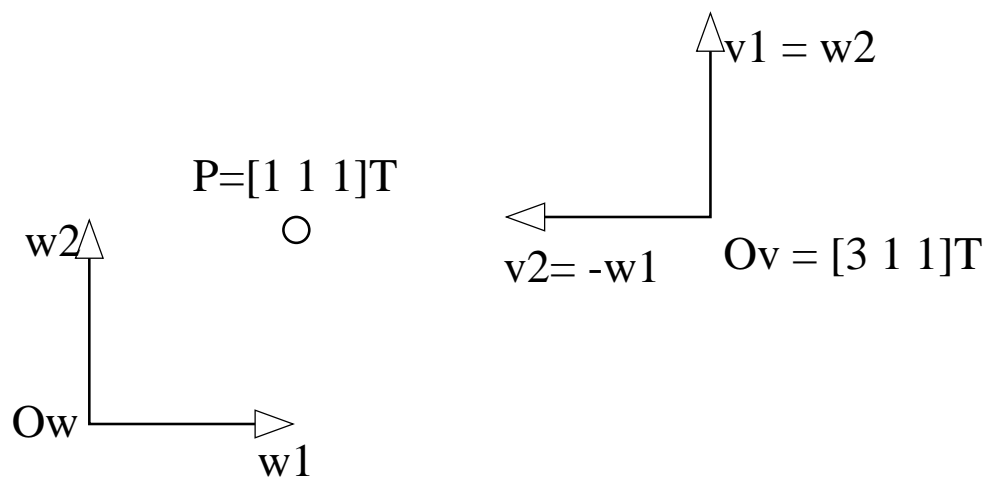
- If  $F_2$  is orthogonal:

$$f_{i,j} = \frac{\vec{w}_j \cdot \vec{v}_i}{\vec{v}_i \cdot \vec{v}_i},$$

$$f_{i,3} = \frac{(\mathcal{O}_W - \mathcal{O}_V) \cdot \vec{v}_i}{\vec{v}_i \cdot \vec{v}_i}.$$

- Otherwise, we have to solve a small system of linear equations, using  $F_1 = F_2 M_{1,2}$ .
- Change of basis from  $F_1$  to Standard Cartesian Frame is trivial (since frame elements normally expressed with respect to Standard Cartesian Frame).

Example:



where  $F_W$  is the standard coordinate frame.

**Generalization** to 3D is straightforward ...

**Example:**

- Define two frames:

$$\begin{aligned} F_W &= (\vec{w}_1, \vec{w}_2, \vec{w}_3, \mathcal{O}_W) \\ &= \left( \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right) \\ F_V &= (\vec{v}_1, \vec{v}_2, \vec{v}_3, \mathcal{O}_V) \\ &= \left( \begin{bmatrix} \sqrt{2}/2 \\ \sqrt{2}/2 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} \sqrt{2}/2 \\ -\sqrt{2}/2 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 3 \\ 1 \end{bmatrix} \right) \end{aligned}$$

- All coordinates are specified relative to the standard frame in a Cartesian 3 space.
- In this example,  $F_W$  is the standard frame.
- Note that both  $F_W$  and  $F_V$  are orthonormal.
- The matrix mapping  $F_W$  to  $F_V$  is given by

$$M = \begin{bmatrix} \sqrt{2}/2 & \sqrt{2}/2 & 0 & -\sqrt{2}/2 \\ 0 & 0 & 1 & -3 \\ \sqrt{2}/2 & -\sqrt{2}/2 & 0 & -\sqrt{2}/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Question: What is the matrix mapping from  $F_V$  to  $F_W$ ?

**Notes**

- On the computer, frame elements usually specified in Standard Frame for space.  
Eg, a frame  $F = [\vec{v}_1, \vec{v}_2, \mathcal{O}_V]$  is given by

$$[ [v_{1x}, v_{1y}, 0]^T, [v_{2x}, v_{2y}, 0]^T, [v_{3x}, v_{3y}, 1]^T ]$$

relative to Standard Frame.

Question: What are coordinates of these basis elements relative to  $F$ ?

- Frames are usually orthonormal.
- A point “mapped” by a change of basis does *not change*;  
We have merely expressed its coordinates relative to a different frame.

(Readings: Watt: 1.1.1. Hearn and Baker: Section 5-5 (not as general as here, though). Red book: 5.9. White book: 5.8.)

## 6.8 Ambiguity

### Ambiguity

#### Three Types of Transformations:

1.  $T : \mathcal{A} \mapsto \mathcal{B}$  (between two spaces)
2.  $T : \mathcal{A} \mapsto \mathcal{A}$  (“warp” an object within its own space)
3.  $T$  : change of coordinates

#### Changes of Coordinates:

- Given 2 frames:
  - $F_1 = (\vec{v}_1, \vec{v}_2, \mathcal{O})$ , orthonormal,
  - $F_2 = (2\vec{v}_1, 2\vec{v}_2, \mathcal{O})$ , orthogonal.
- Suppose  $\vec{v} \equiv F_1[1, 1, 0]^T$ .
- Then  $\vec{v} \equiv F_2[1/2, 1/2, 0]^T$ .

**Question:** What is the length of  $\vec{v}$ ?

**Suppose** we have  $P \equiv [p_1, p_2, 1]^T$  and  $Q \equiv [q_1, q_2, 1]^T$

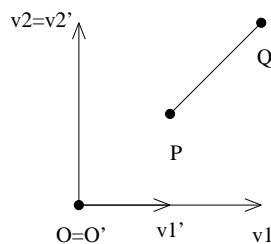
- $P, Q$  relative to  $F = (\vec{v}_1, \vec{v}_2, \mathcal{O})$ ,
- We are given a matrix representation of a transformation  $T$ :

$$M_T = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

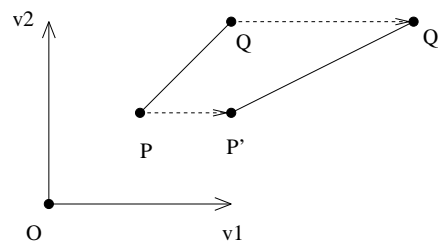
- Consider  $P' = TP$  and  $Q' = TQ$ .
- How do we interpret  $P'$  and  $Q'$ ?

How do we interpret  $P'$  and  $Q'$ ?

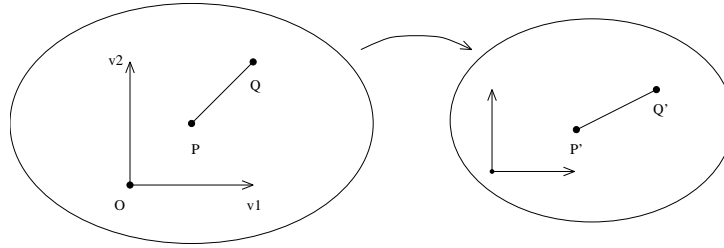
1. Change of Coordinates?



2. Scale?



## 3. Transformations between spaces?



**Do we care? YES!**

- In (1) nothing changes except the representation.
- In (1) distances are preserved while they change in (2) and the question has no meaning in (3).
- In (3), we've completely changed spaces.

**Consider** the meaning of  $|P' - P|$

1.  $|P' - P| = 0$
2.  $|P' - P| = \sqrt{(2p_1 - p_1)^2 + (p_2 - p_2)^2} = |p_1|$
3.  $|P' - P|$  has no meaning

**To fully specify a transformation, we need**

1. A matrix
2. A domain space
3. A range space
4. A coordinate frame in each space

## 6.9 3D Transformations

### 3D Transformations

Assume a right handed coordinate system

Points  $P \equiv [x, y, z, 1]^T$ , Vectors  $\vec{v} \equiv [x, y, z, 0]^T$

**Translation:**

$$T(\Delta x, \Delta y, \Delta z) = \begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**Scale:** About the origin

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



**Rotation:** About a coordinate axis

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

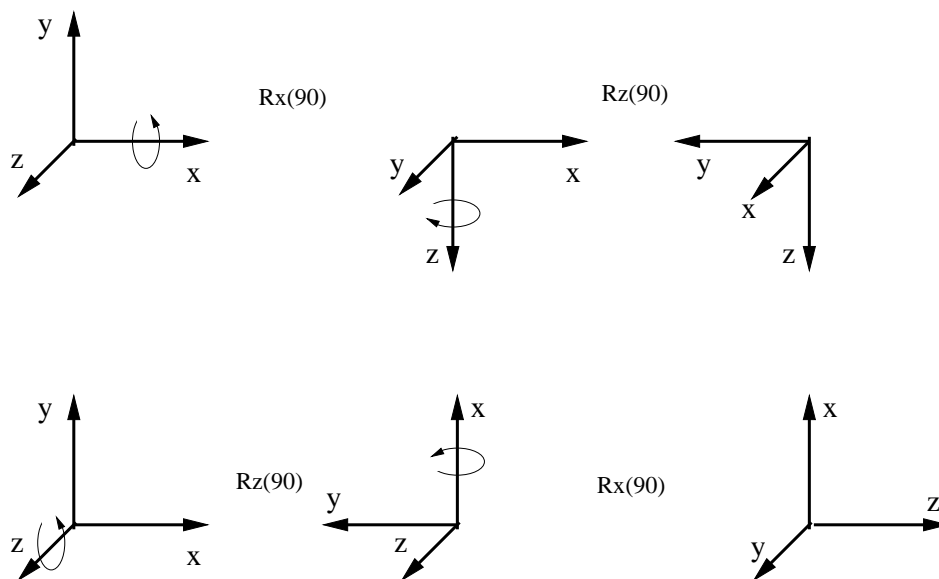
$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

**Shear:** see book

**Reflection:** see book

**Composition:** works same way (but order counts when composing rotations).



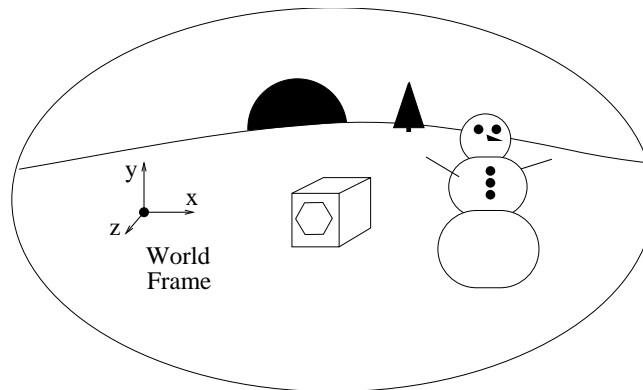
(Readings: Hearn and Baker, Chapter 11; Red book, 5.7; White book, 5.6 )

## 6.10 World and Viewing Frames

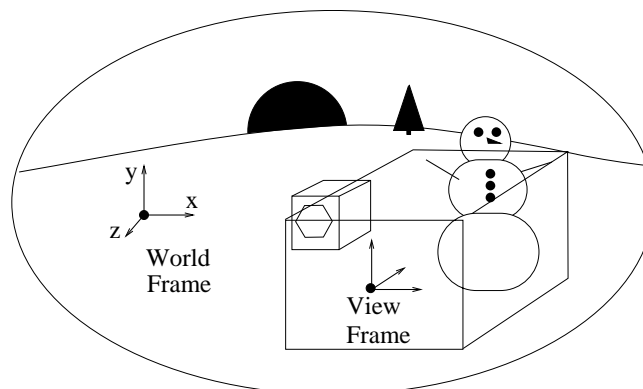
### World and Viewing Frames

- Typically, our space  $S$  is a Cartesian space.
  - Call the standard frame the **world frame**.
  - The world frame is typically **right handed**.

- Our scene description is specified in terms of the world frame.



- The viewer may be anywhere and looking in any direction.
  - Often,  $x$  to the right,  $y$  up, and  $z$  straight ahead.
    - \*  $z$  is called the *viewing direction*.
    - \* This is a **left handed** coordinate system.
  - We could instead specify  $z$  and  $y$  as vectors
    - \*  $z$  is the **view direction**.
    - \*  $y$  is the *up vector*.
    - \* Compute  $x = y \times z$
    - \* Get a **right handed** coordinate system.
  - We can do a change of basis
    - \* Specify a frame relative to the viewer.
    - \* Change coordinates to this frame.
- Once in viewing coordinates,
  - Usually place a clipping “box” around the scene.
  - Box oriented relative to the viewing frame.



- An **orthographic projection** is made by “removing the  $z$ -coordinate.”

- Squashes 3D onto 2D, where we can do the window-to-viewport map.
- The projection of the clipping box is used as the window.

- Mathematically, relative to

$$F_V = (\vec{i}, \vec{j}, \vec{k}, \mathcal{O})$$

we map  $Q \equiv [q_1, q_2, q_3, 1]^T$  onto

$$F_P = (\vec{u}, \vec{v}, \mathcal{O}')$$

as follows:

$$\text{Ortho}(q_1\vec{i} + q_2\vec{j} + q_3\vec{k} + \mathcal{O}) = q_1\vec{u} + q_2\vec{v} + \mathcal{O}'$$

or if we ignore the frames,

$$[q_1, q_2, q_3, 1]^T \mapsto [q_1, q_2, 1]^T$$

- We can write this in matrix form:

$$\begin{bmatrix} q_1 \\ q_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ 1 \end{bmatrix}$$

- Question: why would we want to write this in matrix form?

### Viewing-World-Modelling Transformations :

- Want to do modelling transformations and viewing transformation (as in Assignment 2).
- If  $V$  represents World-View transformation, and  $M$  represents modelling transformation, then

$$VM$$

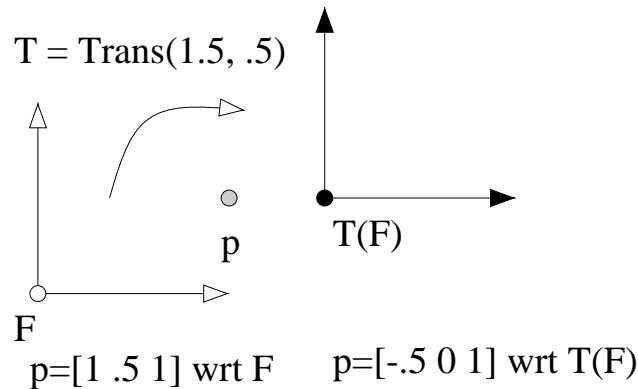
transforms from modelling coordinates to viewing coordinates.

**Note:**  $M$  is performing both modelling transformation and Model to World change of basis.

- Question: If we transform the viewing frame (relative to viewing frame) how do we adjust  $V$ ?
- Question: If we transform model (relative to modelling frame) how do we adjust  $M$ ?

### Viewing Transformations:

- Assume all frames are orthonormal
- When we transform the View Frame by  $T$ , apply  $T^{-1}$  to anything expressed in old view frame coordinates to get new View Frame coordinates



- To compute new World-to-View change of basis, need to express World Frame in new View Frame  
Get this by transforming World Frame elements represented in old View Frame by  $T^{-1}$ .
- Recall that the columns of the World-to-View change-of-basis matrix are the basis elements of the World Frame expressed relative to the View Frame.
- If  $V$  is old World-to-View change-of-basis matrix, then  $T^{-1}V$  will be new World-to-View change-of-basis matrix, since each column of  $V$  represents World Frame element, and the corresponding column of  $T^{-1}V$  contains  $T^{-1}$  of this element.

### Modelling Transformations:

- Note that the columns of  $M$  are the Model Frame elements expressed relative to the World Frame.
- Want to perform modelling transformation relative to modelling coordinates.
- If we have previously transformed Model Frame, then we next transform relative to transformed Model Frame.
- Example: If

$$M = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and we translate one unit relative to the first Model Frame basis vector, then we want to translate by  $(x_1, y_1, z_1, 0)$  relative to the World Frame.

- Could write this as

$$M' = \begin{bmatrix} 1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & y_1 \\ 0 & 0 & 1 & z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot M = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 + x_1 \\ y_1 & y_2 & y_3 & y_4 + y_1 \\ z_1 & z_2 & z_3 & z_4 + z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- But this is also equal to

$$M' = M \cdot \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 + x_1 \\ y_1 & y_2 & y_3 & y_4 + y_1 \\ z_1 & z_2 & z_3 & z_4 + z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- In general, if we want to transform by  $T$  our model relative to the current Model Frame, then

$$MT$$

yields that transformation.

- Summary:

Modelling transformations embodied in matrix  $M$

World-to-View change of basis in matrix  $V$

$VM$  transforms from modelling coordinates to viewing coordinates

If we further transform the View Frame by  $T$  relative to the View Frame, then the new change-of-basis matrix  $V'$  is given by

$$V' = T^{-1}V$$

If we further transform the model by  $T$  relative to the modelling frame, the new modelling transformation  $M'$  is given by

$$M' = MT$$

- For Assignment 2, need to do further dissection of transformations, but this is the basic idea.

(Readings: Hearn and Baker, Section 6-2, first part of Section 12-3; Red book, 6.7; White book, 6.6 )

## 6.11 Normals

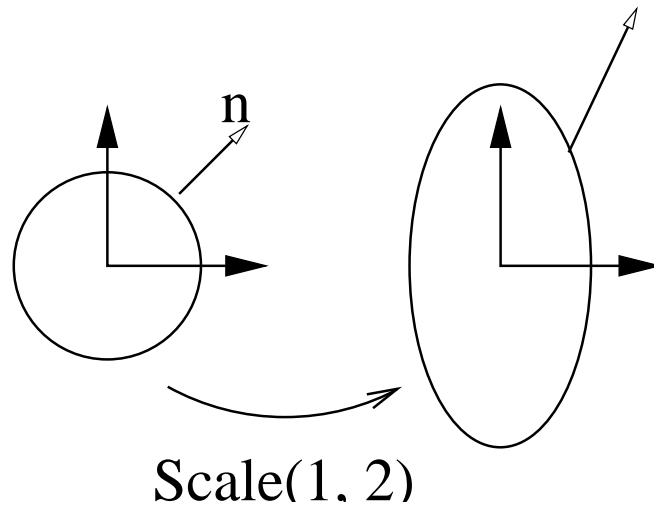
### Transforming Normals

**The Truth:** Can really only apply affine transforms to points.

Vectors can be transformed correctly iff they are defined by differences of points.

### Transforming Normal Vectors:

Consider non-uniform scale of circle, and normal to circle:



Why doesn't normal transform correctly?

- Normal vectors **ARE NOT** defined by differences of points (formally, they are *covectors*, which are *dual* to vectors).
- Tangent vectors **ARE** defined by differences of points.
- Normals are vectors perpendicular to all tangents at a point:

$$\vec{N} \cdot \vec{T} \equiv \mathbf{n}^T \mathbf{t} = 0.$$

- Note that the natural representation of  $\vec{N}$  is as a *row vector*.
- Suppose we have a transformation  $M$ , a point  $P \equiv \mathbf{p}$ , and a tangent  $\vec{T} \equiv \mathbf{t}$  at  $P$ .
- Let  $M_\ell$  be the “linear part” of  $M$ , i.e. the upper  $3 \times 3$  submatrix.

$$\begin{aligned} \mathbf{p}' &= M\mathbf{p}, \\ \mathbf{t}' &= M\mathbf{t} \\ &= M_\ell \mathbf{t}. \\ \mathbf{n}^T \mathbf{t} &= \mathbf{n}^T M_\ell^{-1} M_\ell \mathbf{t} \\ &= (M_\ell^{-1T} \mathbf{n})^T (M_\ell \mathbf{t}) \\ &= (\mathbf{n}')^T \mathbf{t}' \\ &\equiv \vec{N}' \cdot \vec{T}'. \end{aligned}$$

- Transform normals by inverse transpose of linear part of transformation:  $\mathbf{n}' = M_\ell^{-1T} \mathbf{n}$ .
- If  $M_T$  is O.N. (usual case for rigid body transforms),  $M_T^{-1T} = M_T$ .
- Only worry if you have a non-uniform scale or a shear transformation.
- Transforming lines: Transform implicit form in a similar way.
- Transforming planes: Transform implicit form in a similar way.

(Readings: Red Book, 5.8 (?); White Book, 5.6. )

## 7 Windows, Viewports, NDC

### 7.1 Window to Viewport Mapping

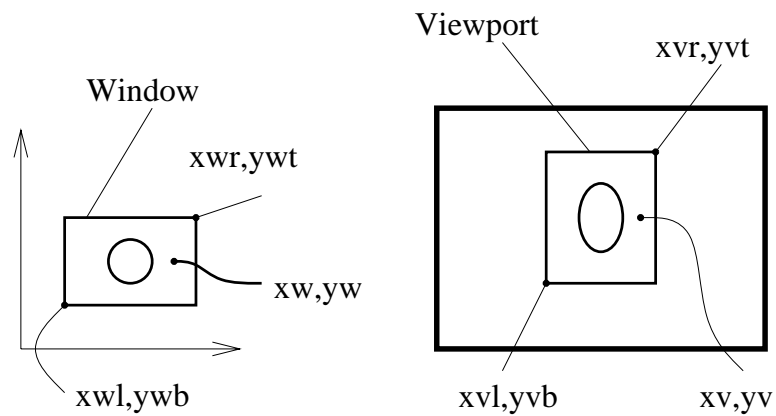
#### Window to Viewport Mapping

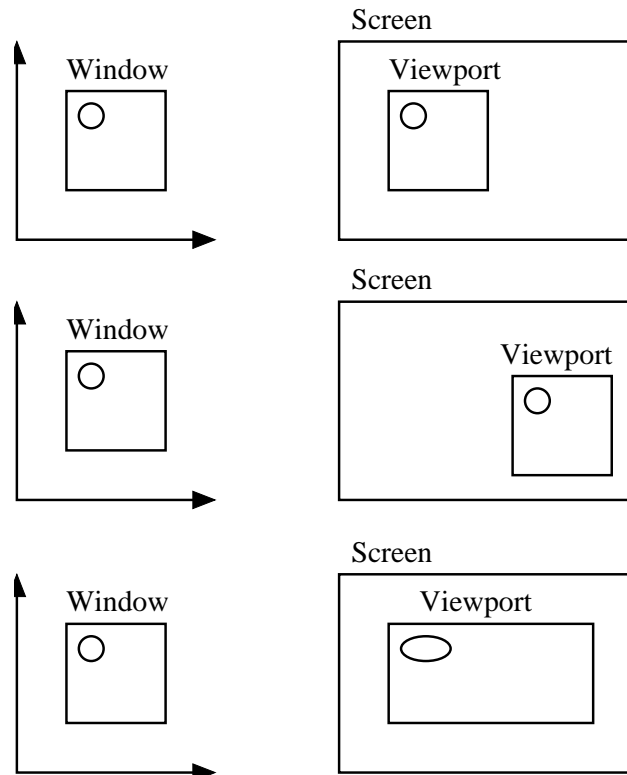
- Start with 3D scene, but eventually project to 2D scene
- 2D scene is infinite plane. Device has a finite visible rectangle. What do we do?
- Answer: map rectangular region of 2D device scene to device.

**Window:** rectangular region of interest in scene.

**Viewport:** rectangular region on device.

Usually, both rectangles are aligned with the coordinate axes.





- Window point  $(x_w, y_w)$  maps to viewport point  $(x_v, y_v)$ .

Length and height of the window are  $L_w$  and  $H_w$ ,  
 Length and height of the viewport are  $L_v$  and  $H_v$ .

- Proportionally map each of the coordinates according to:

$$\frac{\Delta x_w}{L_w} = \frac{\Delta x_v}{L_v}, \quad \frac{\Delta y_w}{H_w} = \frac{\Delta y_v}{H_v}.$$

- To map  $x_w$  to  $x_v$ :

$$\begin{aligned} \frac{x_w - x_{wl}}{L_w} &= \frac{x_v - x_{vl}}{L_v} \\ \Rightarrow x_v &= \frac{L_v}{L_w}(x_w - x_{wl}) + x_{vl}, \end{aligned}$$

and similarly for  $y_v$ .

- If  $H_w/L_w \neq H_v/L_v$  the image will be distorted.  
 These quantities are called the **aspect ratios** of the window and viewport.

(Readings: Watt: none. Hearn and Baker: Section 6-3. Red book: 5.5. White book: 5.4, Blinn: 16.

Intuitively, the window-to-viewport formula can be read as:

- Convert  $x_w$  to a distance from the window corner.
- Scale this  $w$  distance to get a  $v$  distance.
- Add to viewport corner to get  $x_v$ .

)



## 7.2 Normalized Device Coordinates

### Normalized Device Coordinates

- Where do we specify our viewport?
- Could specify it in device coordinates ...  
BUT, suppose we want to run on several platforms/devices

Two common conventions for DCS:

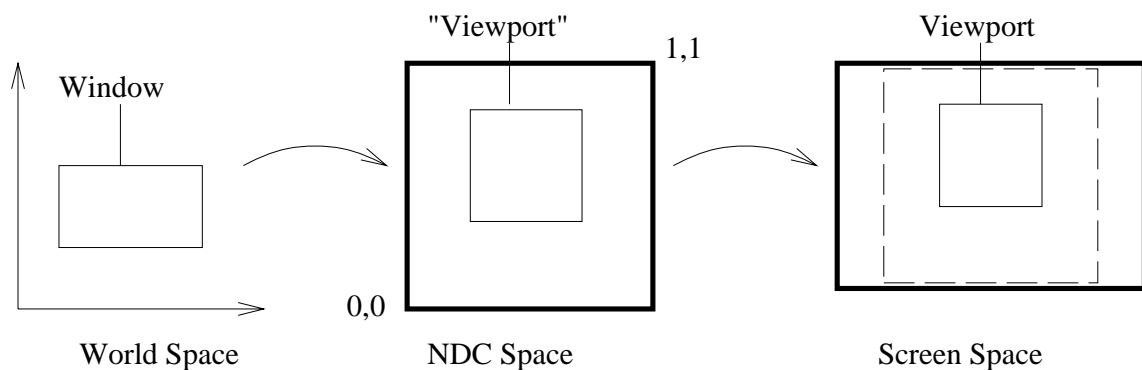
- Origin in the lower left, with  $x$  to the right and  $y$  upward.
- Origin in the top left, with  $x$  to the right and  $y$  downward.

Many different resolutions for graphics display devices:

- Workstations commonly have  $1280 \times 1024$  frame buffers.
- A PostScript page is  $612 \times 792$  points, but  $2550 \times 3300$  pixels at 300dpi.
- And so on ...

Aspect ratios may vary ...

- If we map directly from WCS to a DCS, then changing our device requires rewriting this mapping (among other changes).
- Instead, use **Normalized Device Coordinates (NDC)** as an intermediate coordinate system that gets mapped to the device layer.
- Will consider using only a square portion of the device.  
Windows in WCS will be mapped to viewports that are specified within a unit square in NDC space.
- Map viewports from NDC coordinates to the screen.



(Readings: Watt: none. Hearn and Baker: Sections 2-7, 6-3. Red book: 6.3. White book: 6.5. )



## 8 Clipping

### 8.1 Clipping

#### Clipping

**Clipping:** Remove points outside a region of interest.

- Discard (parts of) primitives outside our window...

**Point clipping:** Remove points outside window.

- A point is either entirely inside the region or not.

**Line clipping:** Remove portion of line segment outside window.

- Line segments can straddle the region boundary.
- Liang-Barsky algorithm efficiently clips line segments to a halfspace.
- Halfspaces can be combined to bound a convex region.
- Can use some of the ideas in Liang-Barsky to clip points.

**Parametric representation of line:**

$$L(t) = (1 - t)A + tB$$

or equivalently

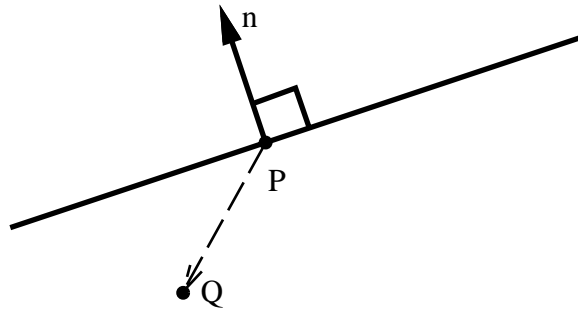
$$L(t) = A + t(B - A)$$

- $A$  and  $B$  are non-coincident points.
- For  $t \in \mathbb{R}$ ,  $L(t)$  defines an infinite line.
- For  $t \in [0, 1]$ ,  $L(t)$  defines a line segment from  $A$  to  $B$ .
- Good for generating points on a line.
- Not so good for testing if a given point is on a line.

**Implicit representation of line:**

$$\ell(Q) = (Q - P) \cdot \vec{n}$$

- $P$  is a point on the line.
- $\vec{n}$  is a vector perpendicular to the line.
- $\ell(Q)$  gives us the signed distance from any point  $Q$  to the line.
- The sign of  $\ell(Q)$  tells us if  $Q$  is on the left or right of the line, relative to the direction of  $\vec{n}$ .
- If  $\ell(Q)$  is zero, then  $Q$  is on the line.
- Use same form for the implicit representation of a halfspace.



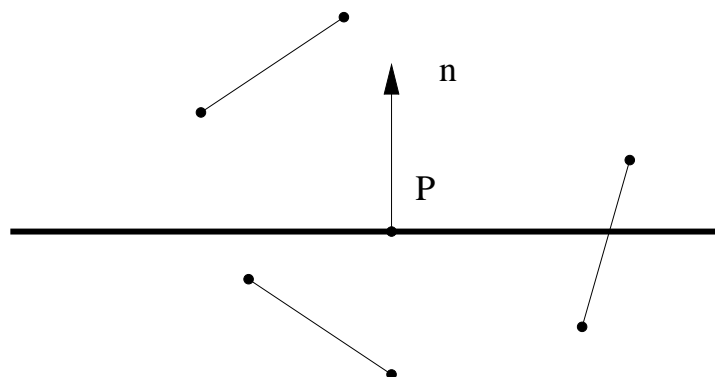
### Clipping a point to a halfspace:

- Represent window edge as implicit line/halfspace.
- Use the implicit form of edge to classify a point  $Q$ .
- Must choose a convention for the normal: points to the *inside*.
- Check the sign of  $\ell(Q)$ :
  - If  $\ell(Q) > 0$ , then  $Q$  is inside.
  - Otherwise clip (discard)  $Q$ :  
It is on the edge or outside.  
May want to keep things on the boundary.

### Clipping a line segment to a halfspace:

There are three cases:

1. The line segment is entirely inside:  
*Keep it.*
2. The line segment is entirely outside:  
*Discard it.*
3. The line segment is partially inside and partially outside:  
*Generate new line to represent part inside.*



### Input Specification:

- Window edge: implicit,  $\ell(Q) = (Q - P) \cdot \vec{n}$
- Line segment: parametric,  $L(t) = A + t(B - A)$ .

**Do the easy stuff first:**

We can devise easy (and fast!) tests for the first two cases:

- $\ell(A) < 0$  **AND**  $\ell(B) < 0 \implies$  Outside
- $\ell(A) > 0$  **AND**  $\ell(B) > 0 \implies$  Inside

Need to decide: are **boundary points inside or outside?**

**Trivial tests** are important in computer graphics:

- Particularly if the trivial case is the most common one.
- Particularly if we can reuse the computation for the non-trivial case.

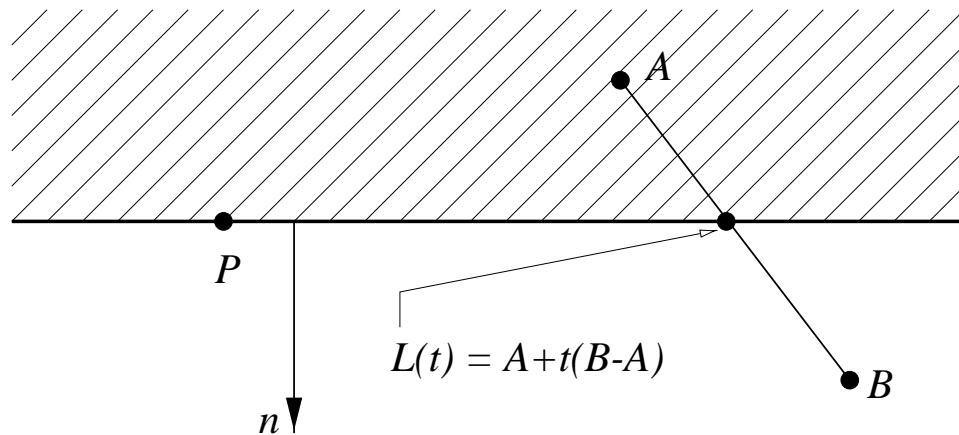
**Do the hard stuff only if we have to:**

If line segment partially in/partially out, need to clip it:

- Line segment from  $A$  to  $B$  in **parametric** form:

$$L(t) = (1 - t)A + tB = A + t(B - A)$$

- When  $t = 0$ ,  $L(t) = A$ . When  $t = 1$ ,  $L(t) = B$ .
- We now have the following:



Recall:  $\ell(Q) = (Q - P) \cdot \vec{n}$

- We want  $t$  such that  $\ell(L(t)) = 0$ :

$$\begin{aligned} (L(t) - P) \cdot \vec{n} &= (A + t(B - A) - P) \cdot \vec{n} \\ &= (A - P) \cdot \vec{n} + t(B - A) \cdot \vec{n} \\ &= 0 \end{aligned}$$

- Solving for  $t$  gives us

$$t = \frac{(A - P) \cdot \vec{n}}{(A - B) \cdot \vec{n}}$$

- **NOTE:**

The values we use for our simple test can be reused to compute  $t$ :

$$t = \frac{(A - P) \cdot \vec{n}}{(A - P) \cdot \vec{n} - (B - P) \cdot \vec{n}}$$

**Clipping a line segment to a window:**

Just clip to each of four halfspaces in turn.

**Pseudo-code (here  $wec = \text{window-edge coordinates}$ ):**

```

Given line segment (A,B), clip in-place:
for each edge (P,n)
    wecA = (A-P) . n
    wecB = (B-P) . n
    if ( wecA < 0 AND wecB < 0 ) then reject
    if ( wecA >= 0 AND wecB >= 0 ) then next
    t = wecA / (wecA - wecB)
    if ( wecA < 0 ) then
        A = A + t*(B-A)
    else
        B = A + t*(B-A)
    endif
endfor

```

**Note:**

- Liang-Barsky Algorithm can clip lines to any **convex** window.
- Optimizations can be made for the special case of horizontal and vertical window edges.

**Question:**

Should we clip before or after window-to-viewport mapping?

**Line-clip Algorithm generalizes to 3D:**

- Half-space now lies on one side of a *plane*.
- Plane also given by normal and point.
- Implicit formula for plane in 3D is same as that for line in 2D.
- Parametric formula for line to be clipped is unchanged.

(Readings: Watt: none. Hearn and Baker: Sections 6-5 through 6-7 (12-5 for 3D clipping). Red book: 3.9. White Book: 3.11. Blinn: 13. )

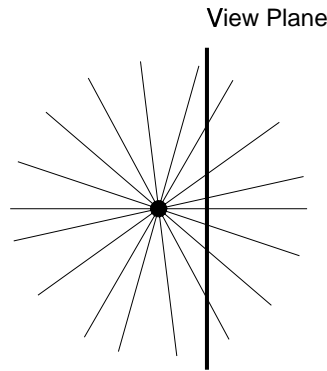
## 9 Projections and Projective Transformations

### 9.1 Projections

#### Projections

##### Perspective Projection

- Identify all points with a line through the eyepoint.
- Slice lines with viewing plane, take intersection point as projection.



- This is **not** an affine transformation, but a **projective transformation**.

##### Projective Transformations:

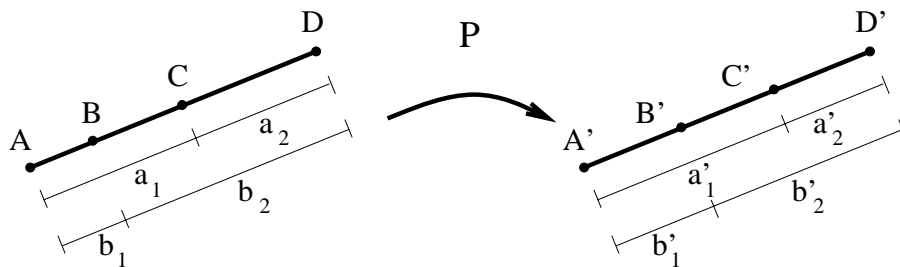
- Angles are not preserved (not preserved under Affine Transformation).
- Distances are not preserved (not preserved under Affine Transformation).
- Ratios of distances are not preserved.
- Affine combinations are not preserved.
- Straight lines are mapped to straight lines.
- *Cross ratios* are preserved.

##### Cross Ratios

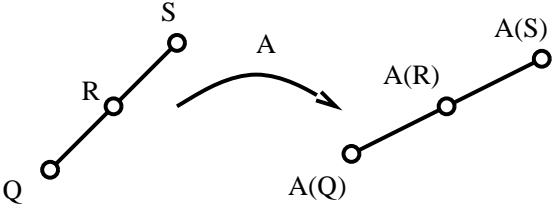
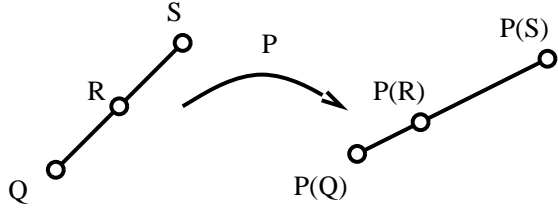
- Cross ratio:  $|AC| = a_1$ ,  $|CD| = a_2$ ,  $|AB| = b_1$ ,  $|BD| = b_2$ , then

$$\frac{a_1/a_2}{b_1/b_2} \left( = \frac{a'_1/a'_2}{b'_1/b'_2} \right)$$

This can also be used to define a projective transformation (ie, that lines map to lines and cross ratios are preserved).

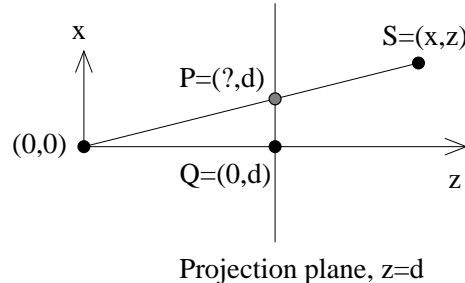


**Comparison:**

Affine Transformations	Projective Transformations
Image of 2 points on a line determine image of line	Image of 3 points on a line determine image of line
Image of 3 points on a plane determine image of plane	Image of 4 points on a plane determine image of plane
In dimension $n$ space, image of $n + 1$ points/vectors defines affine map.	In dimension $n$ space, image of $n + 2$ points/vectors defines projective map.
Vectors map to vectors $\vec{v} = Q - R = R - S \Rightarrow$ $A(Q) - A(R) = A(R) - A(S)$	Mapping of vector is ill-defined $\vec{v} = Q - R = R - S$ but $P(Q) - P(R) \neq P(R) - P(S)$
	
Can represent with matrix multiply (sort of)	Can represent with matrix multiply and normalization

**Perspective Map:**

- Given a point  $S$ , we want to find its projection  $P$ .



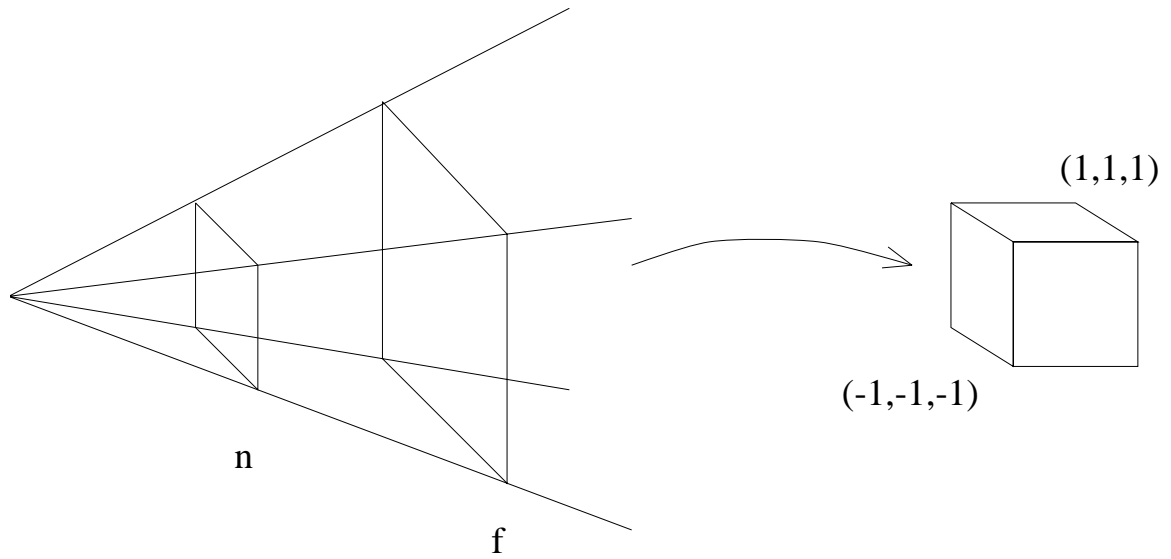
- Similar triangles:  $P = (xd/z, d)$
- In 3D,  $(x, y, z) \mapsto (xd/z, yd/z, d)$
- Have identified all points on a line through the origin with a point in the projection plane.  
 $(x, y, z) \equiv (kx, ky, kz), k \neq 0.$
- These are known as homogeneous coordinates.
- If we have solids or coloured lines,  
then we need to know “which one is in front”.
- This map loses all  $z$  information, so it is inadequate.

**Pseudo-OpenGL version** of the perspective map:

- Maps a near clipping plane  $z = n$  to  $z' = -1$



- Maps a far clipping plane  $z = f$  to  $z' = 1$



- The “box” in world space known as “truncated viewing pyramid” or “frustum”
  - Project  $x, y$  as before
  - To simplify things, we will project into the  $z = 1$  plane.

#### Derivation:

- Want to map  $x$  to  $x/z$  (and similarly for  $y$ ).
- Use matrix multiply followed by **homogenization**:

$$\begin{aligned}
 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & c \\ 0 & 0 & b & d \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} &= \begin{bmatrix} x \\ y \\ az + c \\ bz + d \end{bmatrix} \\
 &\equiv \begin{bmatrix} \frac{x}{bz+d} \\ \frac{y}{bz+d} \\ \frac{az+c}{bz+d} \\ 1 \end{bmatrix}
 \end{aligned}$$

- Solve for  $a, b, c$ , and  $d$  such that  $z \in [n, f]$  maps to  $z' \in [-1, 1]$ .
- Satisfying our constraints gives us

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ \frac{z(f+n)-2fn}{f-n} \\ z \end{bmatrix}$$

- After homogenizing we get

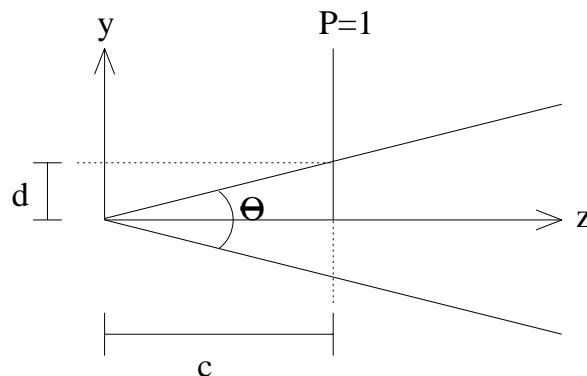
$$\left[ \frac{x}{z}, \frac{y}{z}, \frac{z(f+n)-2fn}{z(f-n)}, 1 \right]^T$$

- Could use this formula instead of performing the matrix multiply followed by the division ...
- If we multiply this matrix in with the geometric transforms, the only additional work is the divide.

The OpenGL perspective matrix uses

- $a = -\frac{f+n}{f-n}$  and  $b = -1$ .
  - OpenGL looks down  $z = -1$  rather than  $z = 1$ .
  - Note that when you specify  $n$  and  $f$ , they are given as *positive* distances down  $z = -1$ .
- The upper left entries are very different.
  - OpenGL uses this one matrix to both project and map to NDC.
  - How do we set  $x$  or  $y$  to map to  $[-1, 1]$ ?
  - We don't want to do both because we may not have square windows.

OpenGL maps  $y$  to  $[-1, 1]$ :



- Want to map distance  $d$  to 1.
- $y \mapsto y/z$  is the current projection ...

Our final matrix is

$$\begin{bmatrix} \frac{\cot(\theta/2)}{\text{aspect}} & 0 & 0 & 0 \\ 0 & \cot(\theta/2) & 0 & 0 \\ 0 & 0 & \pm \frac{f+n}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & \pm 1 & 0 \end{bmatrix}$$

where the  $\pm$  is 1 if we look down the  $z$  axis and -1 if we look down the  $-z$  axis.

OpenGL uses a slightly more general form of this matrix that allows skewed viewing pyramids.

## 9.2 Why Map Z?

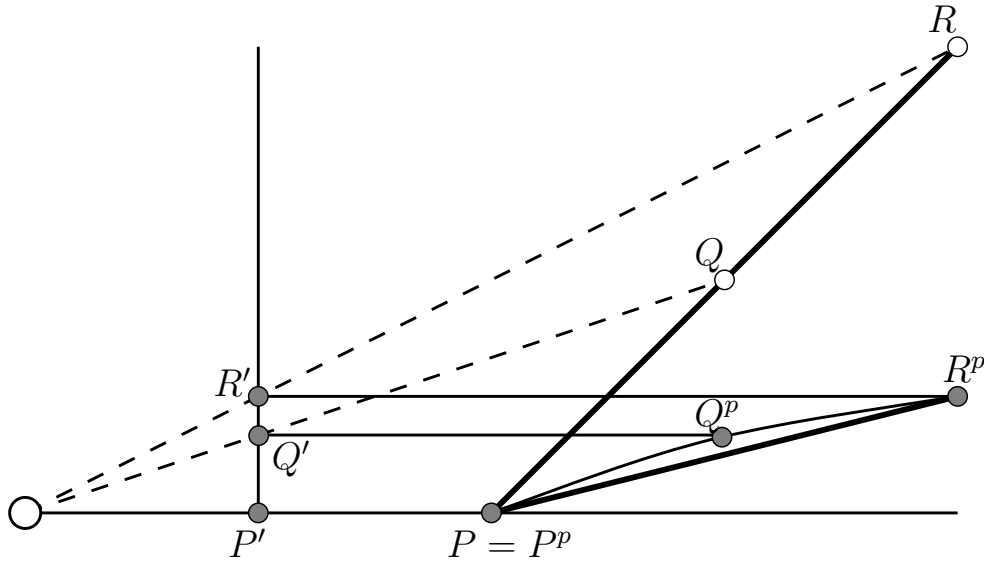
### Why Map Z?

- 3D  $\mapsto$  2D projections map all  $z$  to same value.
- Need  $z$  to determine occlusion, so a 3D to 2D projective transformation doesn't work.
- Further, we want 3D lines to map to 3D lines (this is useful in hidden surface removal)
- The mapping  $(x, y, z, 1) \mapsto (xn/z, yn/z, n, 1)$  maps lines to lines, but loses all depth information.
- We could use

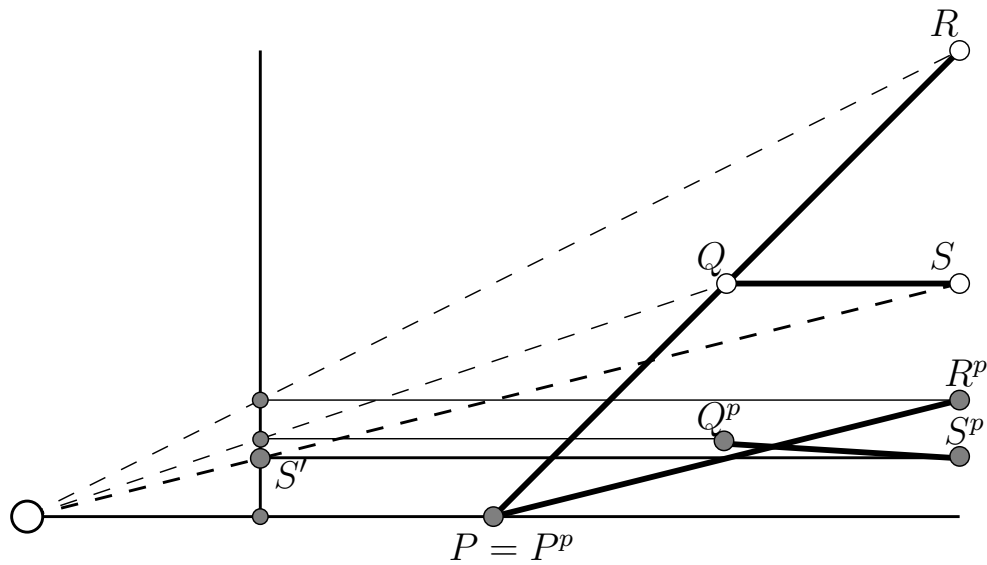
$$(x, y, z, 1) \mapsto \left( \frac{xn}{z}, \frac{yn}{z}, z, 1 \right)$$

Thus, if we map the endpoints of a line segment, these end points will have the same relative depths after this mapping.

BUT: It fails to map lines to lines



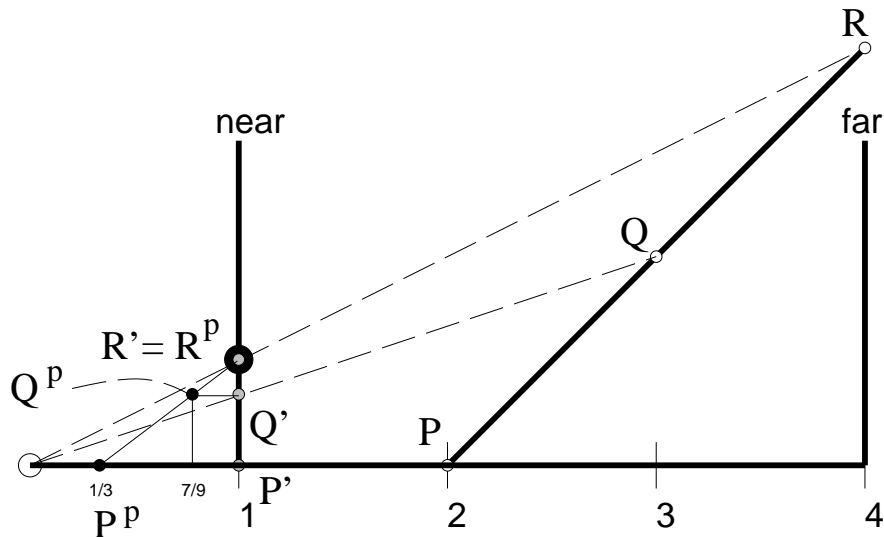
- In this figure,  $P, Q, R$  map to  $P', Q', R'$  under a pure projective transformation.
- With the mapping  $(x, y, z, 1) \mapsto (\frac{xn}{z}, \frac{yn}{z}, z, 1)$   $P, Q, R$  actually map to  $P^p, Q^p, R^p$ , which fail to lie on a straight line.



- Now map  $S$  to  $S'$  to  $S^p$ .
- Line segments  $P^p R^p$  and  $Q^p S^p$  cross!
- The map

$$(x, y, z, 1) \mapsto \left( \frac{xn}{z}, \frac{yn}{z}, \frac{zf + zn - 2fn}{z(f - n)}, 1 \right)$$

**does** map lines to lines, **and** it preserves depth information.



### 9.3 Mapping Z

#### Mapping Z

- It's clear how  $x$  and  $y$  map. How about  $z$ ?

- The  $z$  map affects: clipping, numerics

$$z \mapsto \frac{zf + zn - 2fn}{z(f - n)} = P(z)$$

- We know  $P(f) = 1$  and  $P(n) = -1$ . What maps to 0?

$$\begin{aligned} P(z) &= 0 \\ \Rightarrow \frac{zf + zn - 2fn}{z(f - n)} &= 0 \\ \Rightarrow z &= \frac{2fn}{f + n} \end{aligned}$$

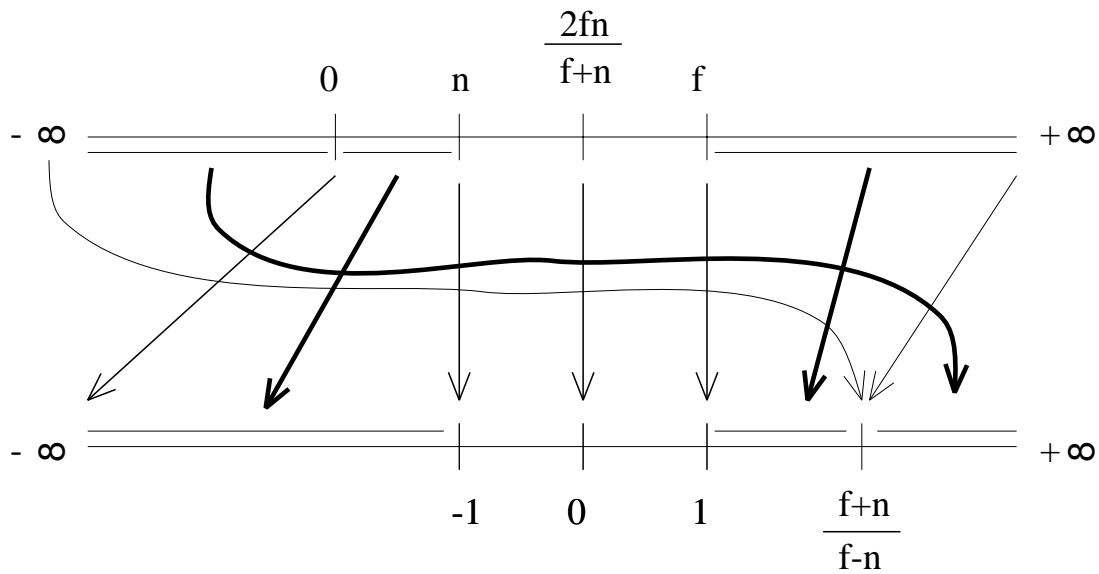
Note that  $f^2 + fn > 2fn > fn + n^2$  so

$$f > \frac{2fn}{f + n} > n$$

- What happens as  $z$  goes to 0 or to infinity?

$$\begin{aligned} \lim_{z \rightarrow 0^+} P(z) &= \frac{-2fn}{z(f - n)} \\ &= -\infty \\ \lim_{z \rightarrow 0^-} P(z) &= \frac{-2fn}{z(f - n)} \\ &= +\infty \\ \lim_{z \rightarrow +\infty} P(z) &= \frac{z(f + n)}{z(f - n)} \\ &= \frac{f + n}{f - n} \\ \lim_{z \rightarrow -\infty} P(z) &= \frac{z(f + n)}{z(f - n)} \\ &= \frac{f + n}{f - n} \end{aligned}$$

Pictorially, we have



- What happens if we vary  $f$  and  $n$ ?

—

$$\begin{aligned}\lim_{f \rightarrow n} P(z) &= \frac{z(f+n) - 2fn}{z(f-n)} \\ &= \frac{(2zn - 2n^2)}{z \cdot 0}\end{aligned}$$

—

$$\begin{aligned}\lim_{f \rightarrow \infty} P(z) &= \frac{zf - 2fn}{zf} \\ &= \frac{z - 2n}{z}\end{aligned}$$

—

$$\begin{aligned}\lim_{n \rightarrow 0} P(z) &= \frac{zf}{zf} \\ &= 1\end{aligned}$$

- What happens as  $f$  and  $n$  move away from each other.  
Look at size of the regions  $[n, 2fn/(f+n)]$  and  $[2fn/(f+n), f]$ .
- When  $f$  is large compared to  $n$ , we have

$$\frac{2fn}{f+n} \doteq 2n$$

So

$$\frac{2fn}{f+n} - n \doteq n$$

and

$$f - \frac{2fn}{f+n} \doteq f - 2n.$$

But both intervals are mapped to a region of size 1.

## 9.4 3D Clipping

### 3D Clipping

- When do we clip in 3D?

We should clip to the near plane *before* we project. Otherwise, we might attempt to project a point with  $z = 0$  and then  $x/z$  and  $y/z$  are undefined.

- We could clip to all 6 sides of the truncated viewing pyramid.

But the plane equations are simpler if we clip after projection, because all sides of volume are parallel to coordinate plane.

- Clipping to a plane in 3D is identical to clipping to a line in 2D.
- We can also clip in homogeneous coordinates.

(Readings: Red Book, 6.6.4; White book, 6.5.4. )

## 9.5 Homogeneous Clipping

### Homogeneous Clipping

#### Projection: transform and homogenize

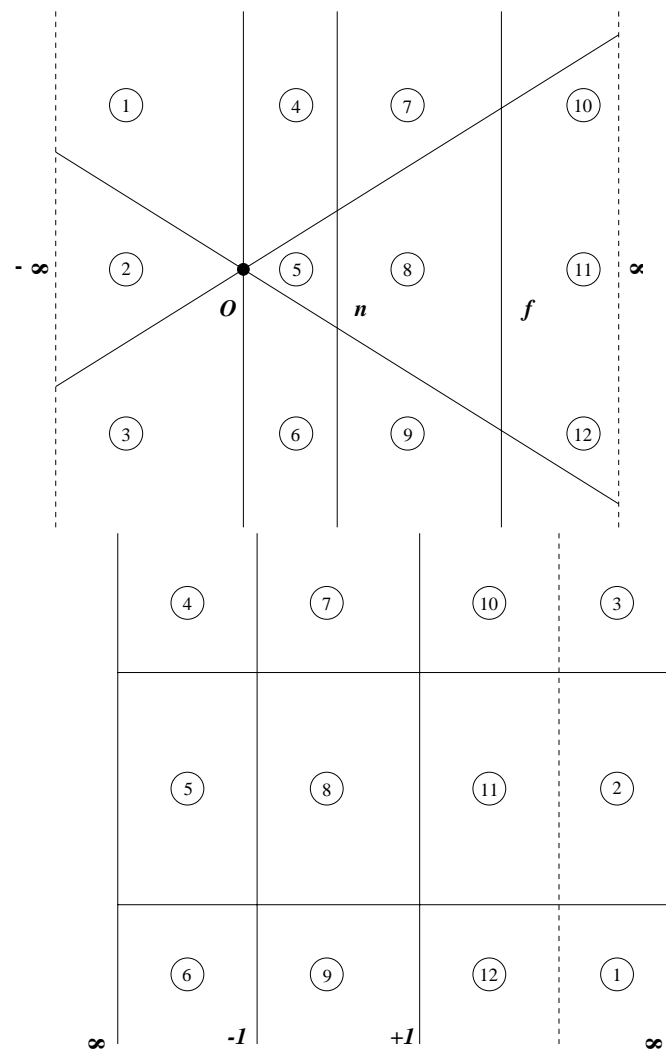
- Linear transformation

$$\begin{bmatrix} nr & 0 & 0 & 0 \\ 0 & ns & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{z} \\ \bar{w} \end{bmatrix}$$

- Homogenization

$$\begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{z} \\ \bar{w} \end{bmatrix} = \begin{bmatrix} \bar{x}/\bar{w} \\ \bar{y}/\bar{w} \\ \bar{z}/\bar{w} \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

#### Region mapping:



### Clipping not good after homogenization:

- Ambiguity after homogenization

$$-1 \leq \frac{\bar{x}, \bar{y}, \bar{z}}{\bar{w}} \leq +1$$

- Numerator can be positive or negative
- Denominator can be positive or negative
- Normalization expended on points that are subsequently clipped

### Clip in homogeneous coordinates:

- Compare unnormalized coordinate against  $\bar{w}$

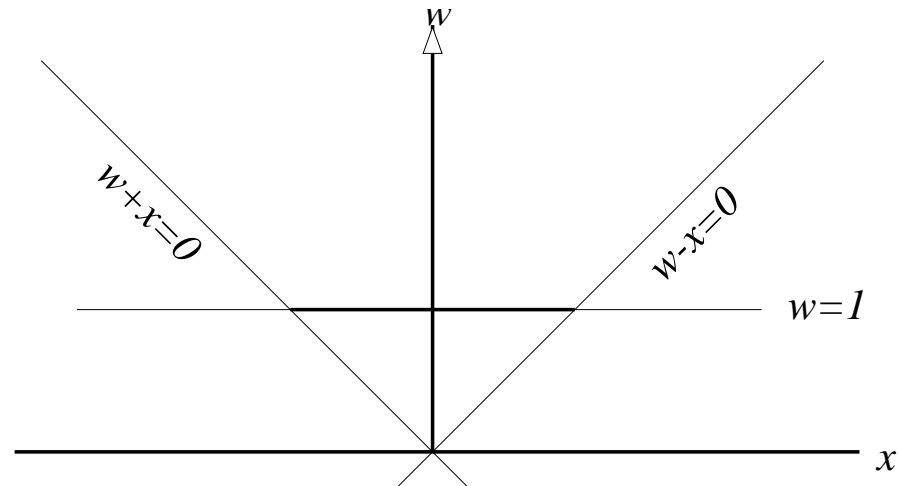
$$-|\bar{w}| \leq \bar{x}, \bar{y}, \bar{z} \leq +|\bar{w}|$$

### Clipping Homogeneous Coordinates

- Assume NDC window of  $[-1, 1] \times [-1, 1]$



- To clip to  $X = -1$  (left):
  - Projected coordinates: Clip to  $X = -1$
  - Homogeneous coordinate: Clip to  $\bar{x}/\bar{w} = -1$
  - Homogeneous plane:  $\bar{w} + \bar{x} = 0$



- Point is visible if  $\bar{w} + \bar{x} > 0$
- Repeat for remaining boundaries:
  - $X = \bar{x}/\bar{w} = 1$
  - $Y = \bar{y}/\bar{w} = -1$
  - $Y = \bar{y}/\bar{w} = 1$
  - Near and far clipping planes

## 9.6 Pinhole Camera vs. Camera vs. Perception

### Pinhole Camera vs. Camera

- “Lines map to lines – can’t be true because I’ve seen pictures that curve lines”
- Camera is not a pinhole camera
- Wide angle lens suffers from *barrel distortion*

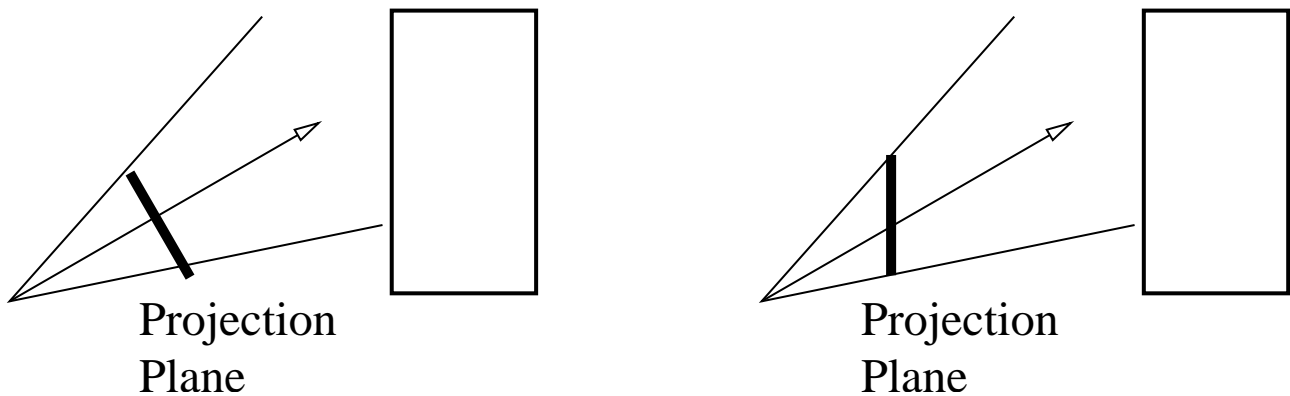
### Vertical Convergence

- Is vertical convergence a pinhole camera effect or another kind of distortion?

## Slanted apartments

- Vertical convergence is a perspective projection effect.  
Occurs when you tilt the camera/view ”up”

- You can get rid of it if view plane parallel to up axis



- Pinhole camera (ray tracer) - Mathematically correct, but looks wrong  
Spheres, cylinders: same size, on a plane parallel to view plane:

## Spheres on cylinders

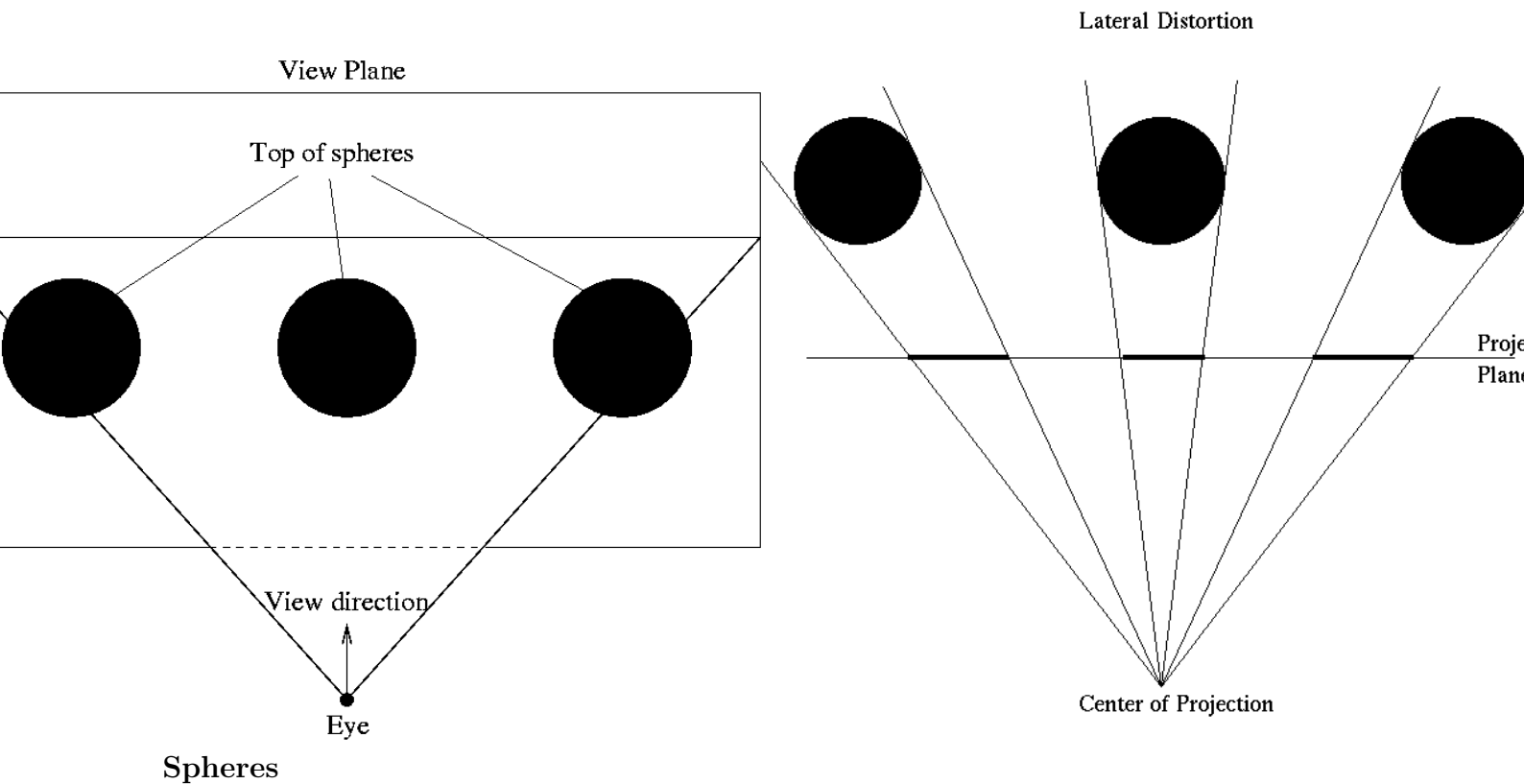
### Art

- Painting Sculpture - Mathematically incorrect, but "looks right"

## Skew-eyed David

What's going on?

## Distortion



# Spheres, and lots of them!

Occurs in Computer Graphics Images

## Ben-hur races in front of distorted columns

In Real Life

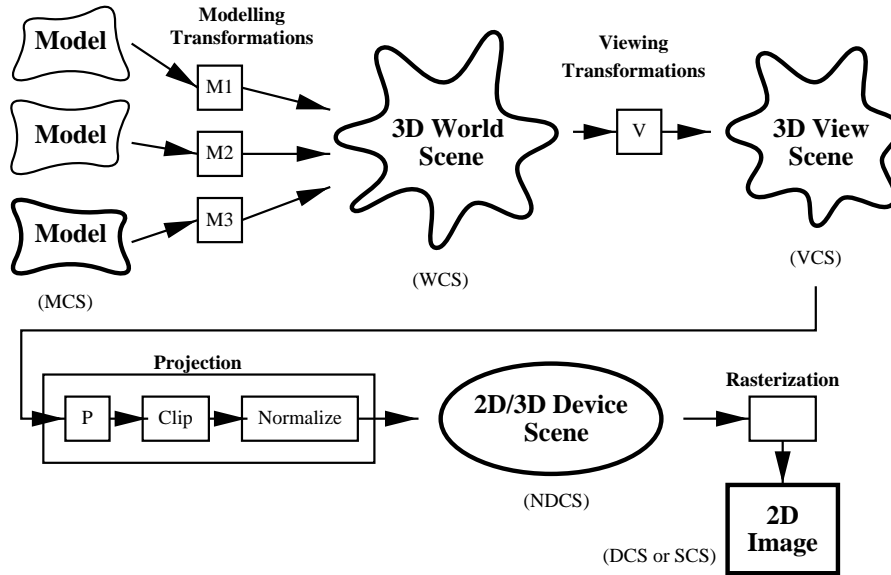
- Eye/attention only on small part of field of view  
Sphere looks circular
- Rest of field of view is “there”  
Peripheral spheres not circular, but not focus of attention and you don’t notice
- When you look at different object, you shift projection plane  
Different sphere looks circular
- In painting, all spheres drawn as circular  
When not looking at them, they are mathematically wrong but since not focus of attention they are “close enough”

- In graphics...

## 10 Transformation Applications and Extensions

### 10.1 Rendering Pipeline Revisited

#### Rendering Pipeline Revisited



Composition of Transforms:  $\mathbf{p}' \equiv PV\mathbf{M}_i\mathbf{p}$ .

### 10.2 Derivation by Composition

#### Derivation by Composition

- Can derive the matrix for angle-axis rotation by composing basic transformations.
  - Rotation given by  $\vec{a} = (x, y, z)$  and  $\theta$ .
  - Assume that  $|\vec{a}| = 1$ .
  - General idea: Map  $\vec{a}$  onto one of the canonical axes, rotate by  $\theta$ , map back.
1. Pick the closest axis to  $\vec{a}$  using  $\max_i \vec{e}_i \cdot \vec{a} = \max(x, y, z)$ . (Assume we chose the  $x$ -axis in the following).
  2. Project  $\vec{a}$  onto  $\vec{b}$  in the  $xz$  plane:

$$\vec{b} = (x, 0, z).$$

3. Compute  $\cos(\phi)$  and  $\sin(\phi)$ , where  $\phi$  is the angle of  $\vec{b}$  with the  $x$ -axis.

$$\begin{aligned} \cos(\phi) &= \frac{x}{\sqrt{x^2 + z^2}}, \\ \sin(\phi) &= \frac{z}{\sqrt{x^2 + z^2}}. \end{aligned}$$

4. Use  $\cos(\phi)$  and  $\sin(\phi)$  to create  $R_y(-\phi)$ :

$$R_y(-\phi) = \begin{bmatrix} \cos(-\phi) & 0 & -\sin(-\phi) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(-\phi) & 0 & \cos(-\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

5. Rotate  $\vec{a}$  onto the  $xy$  plane using  $R_y(-\phi)$ :

$$\begin{aligned} \vec{c} &= R_y(-\phi)\vec{a} \\ &= \left( \sqrt{x^2 + z^2}, y, 0 \right). \end{aligned}$$

6. Compute  $\cos(\psi)$  and  $\sin(\psi)$ , where  $\psi$  is the angle of  $\vec{c}$  with the  $x$ -axis.

$$\begin{aligned} \cos(\psi) &= \frac{\sqrt{x^2 + z^2}}{\sqrt{x^2 + y^2 + z^2}} \\ &= \frac{\sqrt{x^2 + z^2}}{\sqrt{x^2 + y^2 + z^2}}, \\ \sin(\psi) &= \frac{y}{\sqrt{x^2 + y^2 + z^2}} \\ &= \frac{y}{\sqrt{x^2 + y^2 + z^2}}. \end{aligned}$$

7. Use  $\cos(\psi)$  and  $\sin(\psi)$  to create  $R_z(-\psi)$ :

$$R_z(-\psi) = \begin{bmatrix} \cos(-\psi) & -\sin(-\psi) & 0 & 0 \\ \sin(-\psi) & \cos(-\psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

8. Rotate  $\vec{c}$  onto the  $x$  axis using  $R_z(-\psi)$ .

9. Rotate about the  $x$ -axis by  $\theta$ :  $R_x(-\theta)$ .

10. Reverse  $z$ -axis rotation:  $R_z(\psi)$ .

11. Reverse  $y$ -axis rotation:  $R_y(\phi)$ .

The overall transformation is

$$R(\theta, \vec{a}) = R_y(\phi) \circ R_z(\psi) \circ R_x(\theta) \circ R_z(-\psi) \circ R_y(-\phi).$$

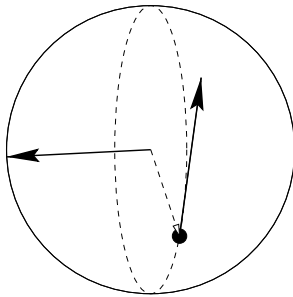
### 10.3 3D Rotation User Interfaces

#### 3D Rotation User Interfaces

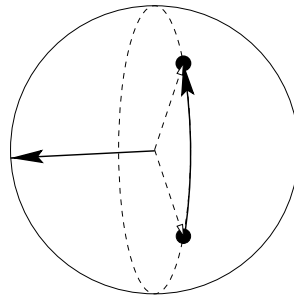
**Goal:** Want to specify angle-axis rotation “directly”.

**Problem:** May only have mouse, which only has two degrees of freedom.

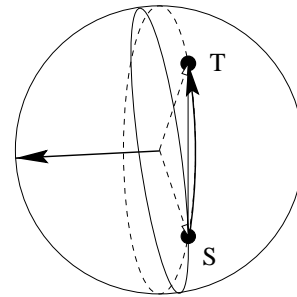
**Solutions:** Virtual Sphere, Arcball.



Virtual Sphere



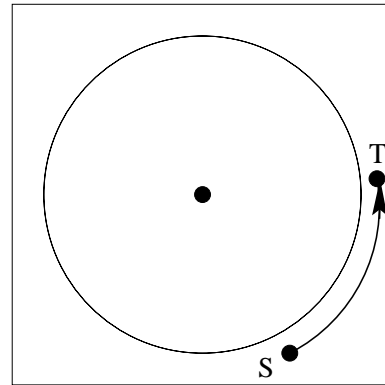
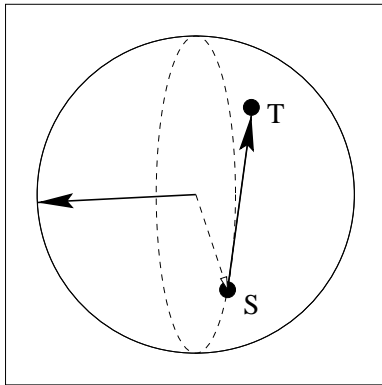
Arcball



Comparison

## 10.4 The Virtual Sphere

### The Virtual Sphere



1. Define portion of screen to be projection of virtual sphere.
2. Get two sequential samples of mouse position,  $S$  and  $T$ .
3. Map 2D point  $S$  to 3D unit vector  $\vec{p}$  on sphere.
4. Map 2D vector  $\vec{ST}$  to 3D tangential velocity  $\vec{d}$ .
5. Normalize  $\vec{d}$ .
6. Axis:  $\vec{a} = \vec{p} \times \vec{d}$ .
7. Angle:  $\theta = \alpha |\vec{ST}|$ .  
(Choose  $\alpha$  so a  $180^\circ$  rotation can be obtained.)
8. Save  $T$  to use as  $S$  for next time.



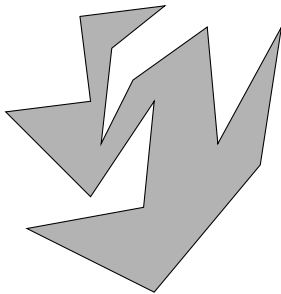


## 11 Polygons

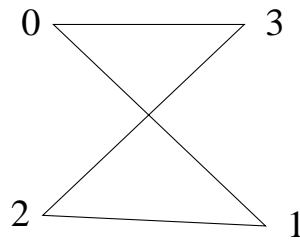
### 11.1 Polygons – Introduction

#### Polygons

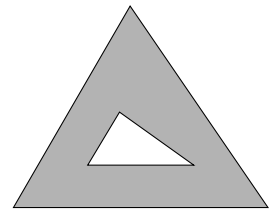
- Need an area primitive
- Simple polygon:
  - Planar set of ordered points,  $v_0, \dots, v_{n-1}$   
(sometimes we repeat  $v_0$  at end of list)
  - No holes
  - No line crossing



OK

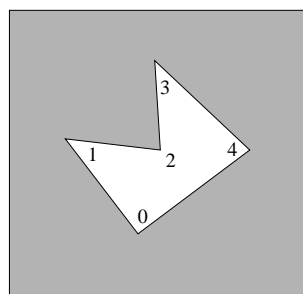
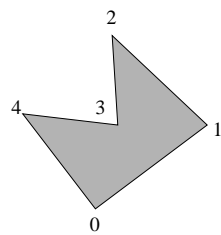


Line Crossing

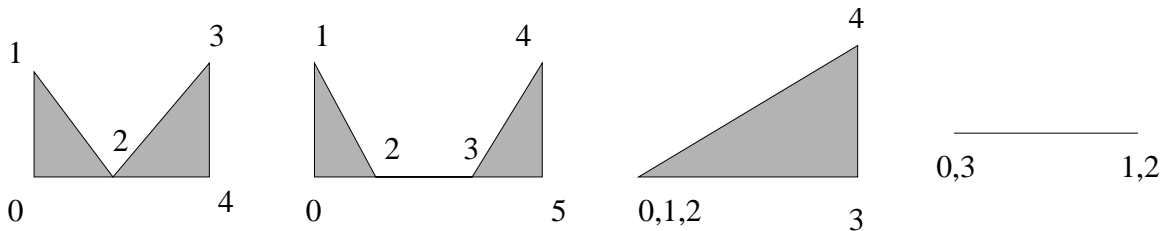


Hole

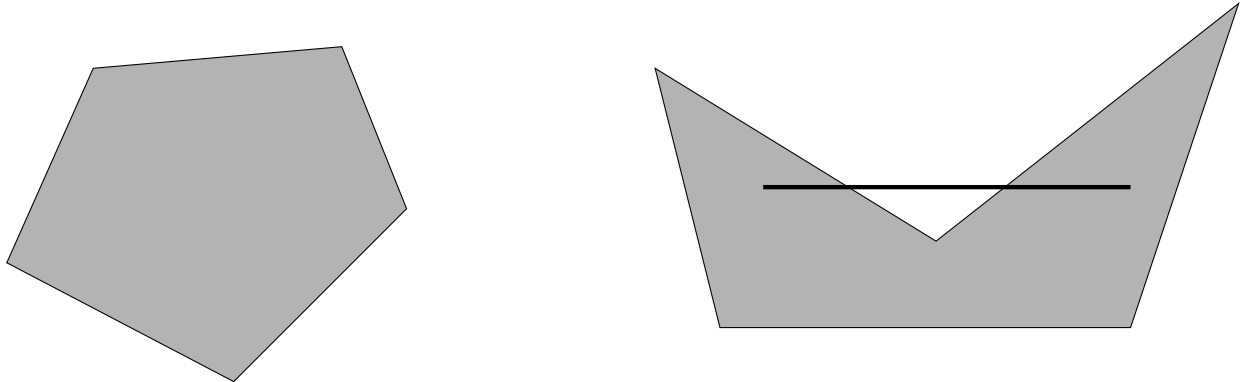
- Normally define an interior and exterior  
Points ordered in counter-clockwise order  
To the “left” as we traverse is inside



- Try to avoid degeneracies, but sometimes unavoidable



- *Convex* and *Concave* polygons



Polygon is convex if for any two points inside polygon, the line segment joining these two points is also inside.

Convex polygons behave better in many operations

- Affine transformations may introduce degeneracies

Example: Orthographic projection may project entire polygon to a line segment.

## 11.2 Polygon Clipping

### Polygon Clipping (Sutherland-Hodgman):

- Window must be a convex polygon
- Polygon to be clipped can be convex or not

#### Approach:

- Polygon to be clipped is given as  $v_1, \dots, v_n$
- Each polygon edge is a pair  $[v_i, v_{i+1}]$   $i = 1, \dots, n$ 
  - Don't forget wraparound;  $[v_n, v_1]$  is also an edge
- Process all polygon edges in succession against a window edge
  - Polygon in – polygon out
  - $v_1, \dots, v_n \rightarrow w_1, \dots, w_m$
- Repeat on resulting polygon with next sequential window edge

### Contrast with Line Clipping:

- **Line Clipping:**
  - Clip only against possibly intersecting window edges
  - Deal with window edges in any order
  - Deal with line segment endpoints in either order
- **Polygon Clipping:**
  - Each window edge must be used

- Polygon edges must be handled in sequence
- Polygon edge endpoints have a given order
- Stripped-down line-segment/window-edge clip is a subtask

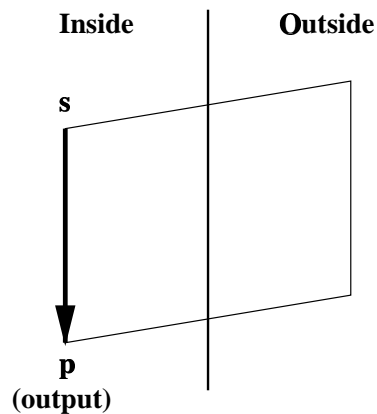
**Notation:**

- $\mathbf{s} = v_i$  is the polygon edge starting vertex
- $\mathbf{p} = v_{i+1}$  is the polygon edge ending vertex
- $\mathbf{i}$  is a polygon-edge/window-edge intersection point
- $w_j$  is the next polygon vertex to be output

There are four cases to consider

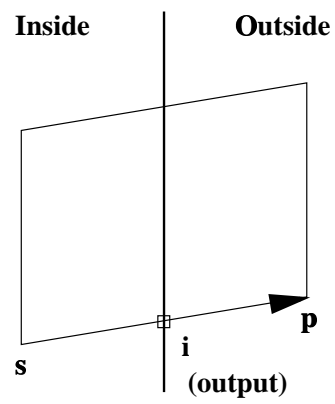
**Case 1:** Polygon edge is entirely inside the window edge

- $\mathbf{p}$  is next vertex of resulting polygon
- $\mathbf{p} \rightarrow w_j$  and  $j + 1 \rightarrow j$



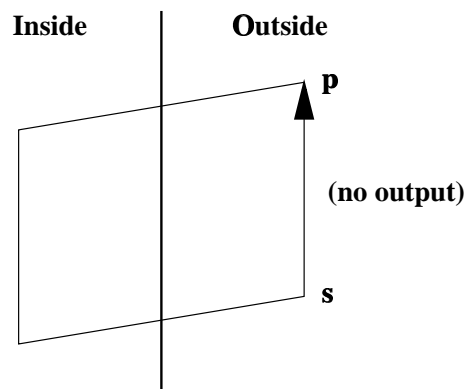
**Case 2:** Polygon edge crosses window edge going out

- Intersection point  $\mathbf{i}$  is next vertex of resulting polygon
- $\mathbf{i} \rightarrow w_j$  and  $j + 1 \rightarrow j$

**Case 2**

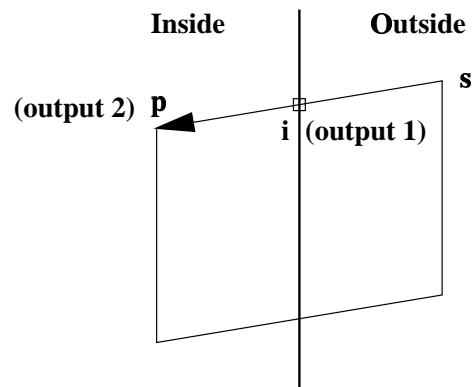
**Case 3:** Polygon edge is entirely outside the window edge

- No output

**Case 3**

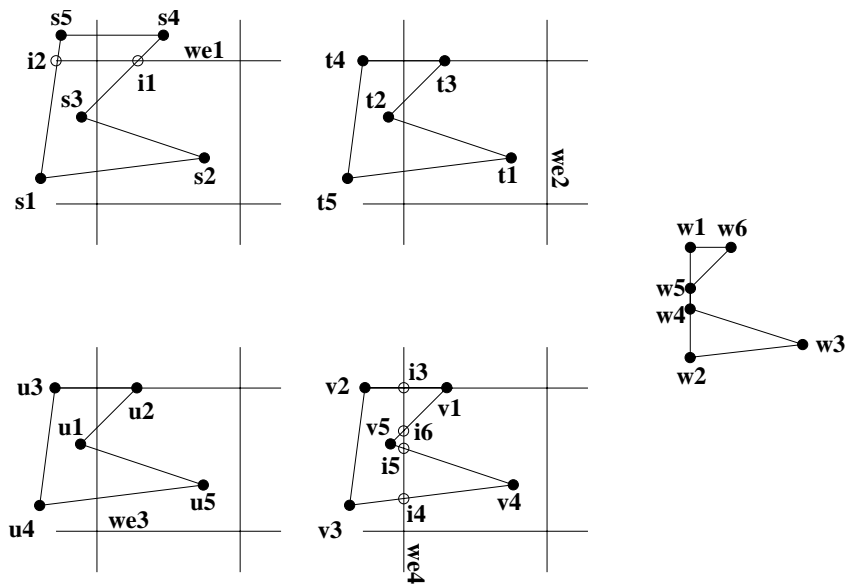
**Case 4:** Polygon edge crosses window edge going in

- Intersection point **i** and **p** are next two vertices of resulting polygon
- **i**  $\rightarrow w_j$  and **p**  $\rightarrow w_{j+1}$  and  $j + 2 \rightarrow j$



Case 4

**An Example:** With a non-convex polygon...

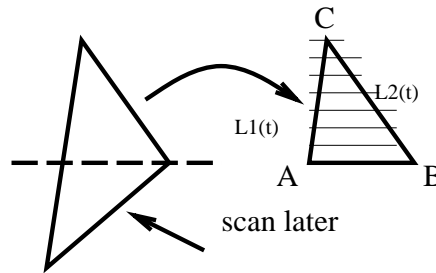


(Readings: Watt: 6.1. Hearn and Baker: Section 6-8. Red book: 3.11. White book: 3.14. )

### 11.3 Polygon Scan Conversion

#### Scan Conversion

- Once mapped to device coordinates, want to scan convert polygon.
- Scan converting a general polygon is complicated.
- Here we will look at scan conversion of a triangle.
- Look at  $y$  value of vertices. Split triangle along horizontal line at middle  $y$  value.



- Step along  $L1$  and  $L2$  together along the scan lines from  $A$  to  $C$  and from  $B$  to  $C$  respectively.
- Scan convert each horizontal line.

#### Code for triangle scan conversion

- Assume that triangle has been split and that  $A, B, C$  are in device coordinates, that  $A.x < B.x$ , and that  $A.y = B.y \neq C.y$ .

```

y = A.y;
d0 = (C.x-A.x)/(C.y-A.y);
d1 = (C.x-B.x)/(C.y-B.y);
x0 = A.x;
x1 = B.x;
while ( y <= C.y ) do
  for ( x = x0 to x1 ) do
    WritePixel(x,y);
  end
  x0 += d0;
  x1 += d1;
  y++;
end

```

- This is a floating point algorithm

(Readings: Watt: 6.4. Hearn and Baker: Chapter 3-11. Red book: 3.5 (more general than above). White book: 3.6 (more general than above). )

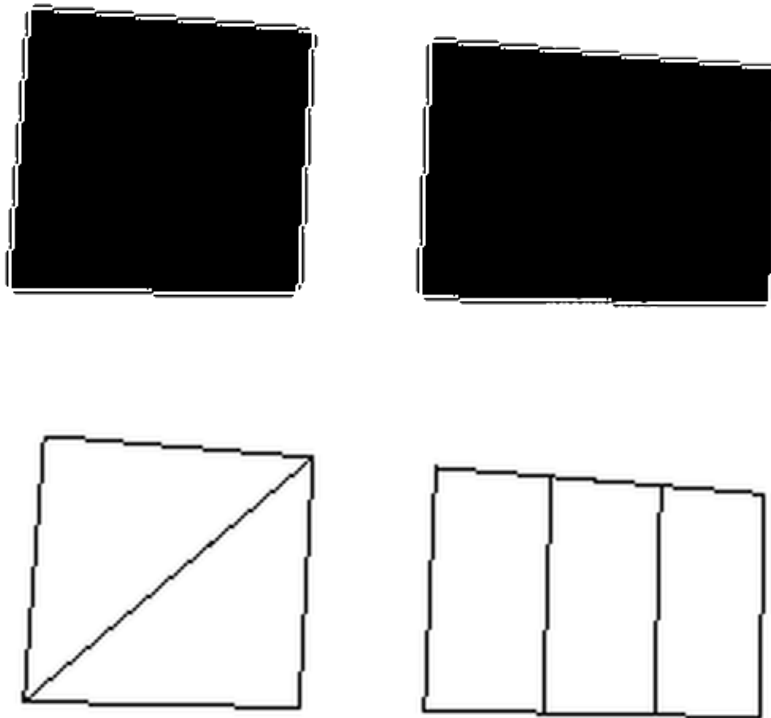
#### 11.4 Dos and Don'ts

When modelling with polygonal objects, remember the following guidelines:

- Model Solid Objects — No façades  
It's easier to write other software if we can assume polygon faces are oriented
- No T-vertices  
These cause shading anomalies and pixel dropout
- No overlapping co-planar faces  
Violates solid object constraint.  
If different colours, then psychedelic effect due to floating point roundoff

- Well shaped polygon  
Equilateral triangle is best.  
Long, skinny triangles pose problems for numerical algorithms

### **Bad Polygon Examples**







## 12 Hidden Surface Removal

### 12.1 Hidden Surface Removal

#### Hidden Surface Removal

- When drawing lots of polygons, we want to draw only those “visible” to viewer.
- There are a variety of algorithms with different strong points.
- Issues:
  - Online
  - Device independent
  - Fast
  - Memory requirements
  - Easy to implement in hardware

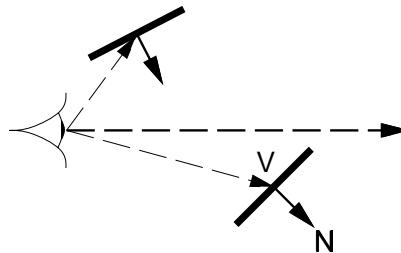
(Readings: Watt: 6.6. Red book: 13. White book: 15. Hearn and Baker: Chapter 13. )

### 12.2 Backface Culling

#### Backface Culling

#### Backface Culling

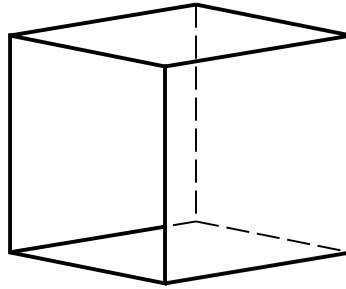
- A simple way to perform hidden surface is to remove all “backfacing” polygons.
- If polygon normal facing away from the viewer then it is “backfacing.”
- For solid objects, this means the polygon will not be seen by the viewer.



- Thus, if  $N \cdot V > 0$ , then cull polygon.
- Note that  $V$  is vector from eye to point on polygon  
You cannot use the view direction for this.

#### Backface Culling Not a complete solution

- If objects not convex, need to do more work.
- If polygons two sided (i.e., they do not enclose a volume) then we can't use it.

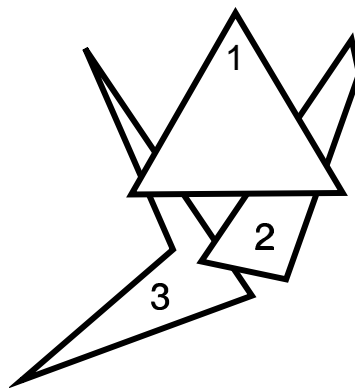


- A HUGE speed advantage if we can use it since the test is cheap and we expect at least half the polygons will be discarded.
- Usually performed in conjunction with a more complete hidden surface algorithm.
- Easy to integrate into hardware (and usually improves performance by a factor of 2).

### 12.3 Painter's Algorithm

#### Painter's Algorithm

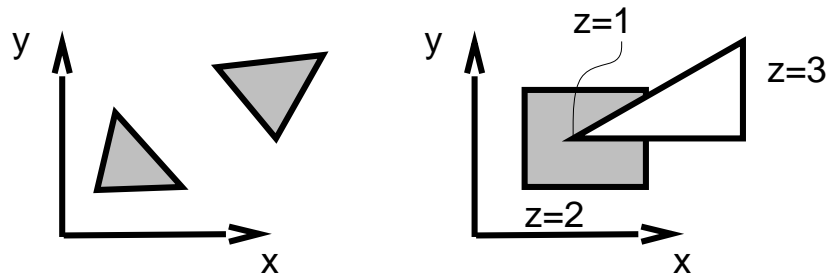
- Idea: Draw polygons as an oil painter might: The farthest one first.
  - Sort polygons on farthest  $z$
  - Resolve ambiguities where  $z$ 's overlap
  - Scan convert from largest  $z$  to smallest  $z$



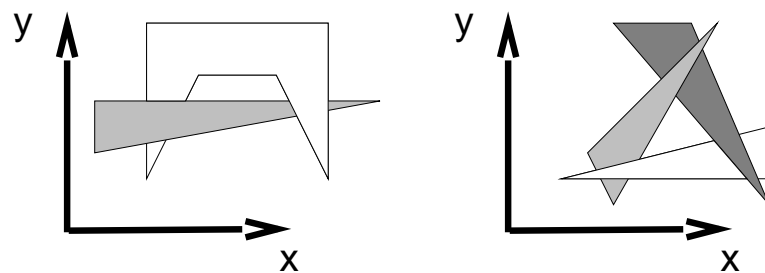
- Since closest drawn last, it will be on top (and therefore it will be seen).
- Need all polygons at once in order to sort.

#### Painter's Algorithm — Z overlap

- Some cases are easy:



- But other cases are nasty!



(have to split polygons)

- $\Omega(n^2)$  algorithm, lots of subtle detail

## 12.4 Warnock's Algorithm

### Warnock's Algorithm

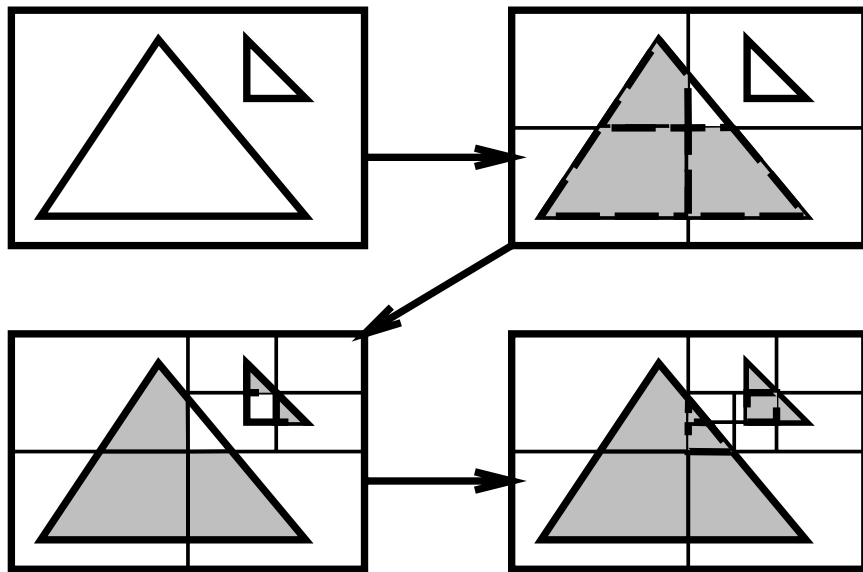
- A divide and conquer algorithm

```

Warnock(PolyList PL, ViewPort VP)
If ( PL simple in VP) then
    Draw PL in VP
else
    Split VP vertically and horizontally into VP1,VP2,VP3,VP4
    Warnock(PL in VP1, VP1)
    Warnock(PL in VP2, VP2)
    Warnock(PL in VP3, VP3)
    Warnock(PL in VP4, VP4)
end
  
```

- What does “simple” mean?
  - No more than one polygon in viewport  
Scan convert polygon clipped to viewport
  - Viewport only 1 pixel in size  
Shade pixel based on closest polygon in the pixel

## Warnock's Algorithm



- Runtime:  $O(p \times n)$ 
  - $p$ : number of pixels
  - $n$ : number of polygons

## 12.5 Z-Buffer Algorithm

## Z-Buffer Algorithm

- Perspective transformation maps viewing pyramid to viewing box in a manner that maps lines to lines
- This transformation also maps polygons to polygons
- Idea: When we scan convert, step in  $z$  as well as  $x$  and  $y$ .
- In addition to framebuffer, we'll have a depth buffer (or  $z$  buffer) where we write  $z$  values
- Initially,  $z$  buffer values set to  $\infty$   
Depth of far clipping plane (usually 1) will also suffice
- Step in  $z$ , both in the while loop and in the for loop.
- Scan convert using the following WritePixel:

```
WritePixel(int x, int y, float z, colour)
if ( z < zbuf[x][y] ) then
    zbuf[x][y] = z;
    framebuffer[x][y] = colour;
end
```

- Runtime:  $O(p_c + n)$ 
  - $p_c$ : number of scan converted pixels
  - $n$ : number of polygons

### Z-buffer in Hardware

Z-buffer is the algorithm of choice for hardware implementation

- + Easy to implement
- + Simple hardware implementation
- + Online algorithm (i.e., we don't need to load all polygons at once in order to run algorithm)
- Doubles memory requirements (at least)
  - But memory is cheap!
- Scale/device dependent

## 12.6 Comparison of Algorithms

### Comparison of Algorithms

- Backface culling fast, but insufficient by itself
- Painter's algorithm device independent, but details tough, and algorithm is slow
- Warnock's algorithm easy to implement, but not very fast, and is semi-device dependent.
- Z-buffer online, fast, and easy to implement, but device dependent and memory intensive.  
Algorithm of choice for hardware

Comparison of no hidden surface removal, backface culling, and hidden surface removal:

