COSC363 Computer Graphics

7

Mathematical Preliminaries

Internals of a graphics engine

R. Mukundan (mukundan@canterbury.ac.nz)
Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

Homogeneous Coordinates

- A point with Cartesian coordinates (*x*, *y*, *z*) can be expressed in homogeneous coordinates as (*hx*, *hy, hz*, *h*) where *h* is a **non-zero** real number.

```
glVertex3f (10, 2, –3);
glVertex4f (10, 2, –3, 1);
glVertex4f (60, 12, –18, 6)
glVertex4f (–20, –4, 6, –2)
```

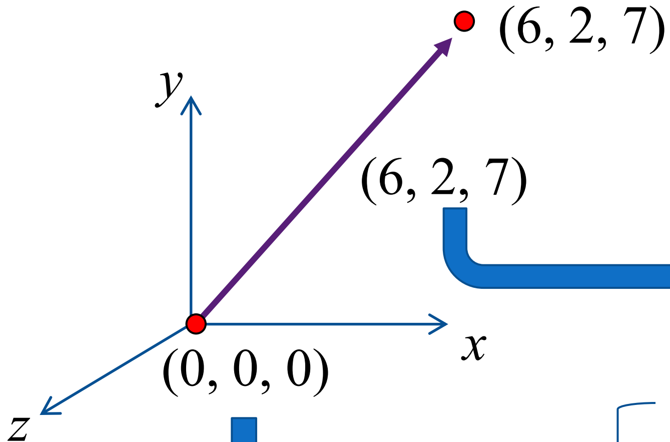
Different representations of the same point

- To convert from homogeneous coordinates to Cartesian coordinates, divide the first three components by the fourth element: $(a, b, c, d) \equiv (a/d, \ \ b/d, \ \ c/d)$

  Example: The *xyz* coordinates of the point (12, –16, 1, 4) are (3, –4, 0.25)

- A vector with components (*x*, *y*, *z*) is represented in homogeneous coordinates as (*x*, *y, z,* 0).

Points and Vectors in Homogeneous Coordinates



A Point :

$= (6, 2, 7, 1)$
$= (18, 6, 21, 3)$
$= (-6, -2, -7, -1)$

$= (6h, 2h, 7h, h), \quad h \neq 0$

A Vector :

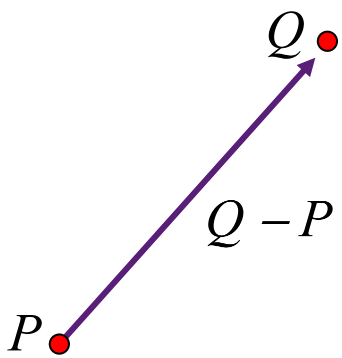
$= (6, 2, 7, 0)$

A Point :

$= (0, 0, 0, 1)$
$= (0, 0, 0, 500)$

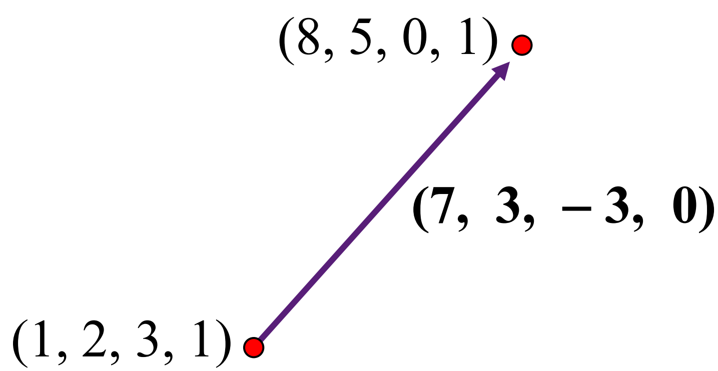
Point Operations

The difference between two points is a vector.



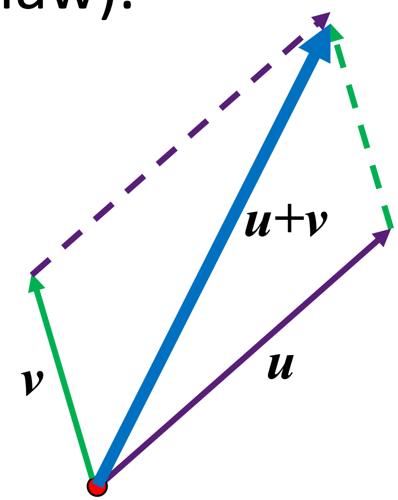
Example:

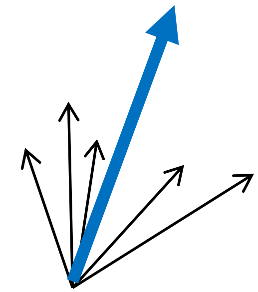
<u>Note</u>: If using homogeneous coordinates, the fourth element for both points must be 1.



Vector Operations

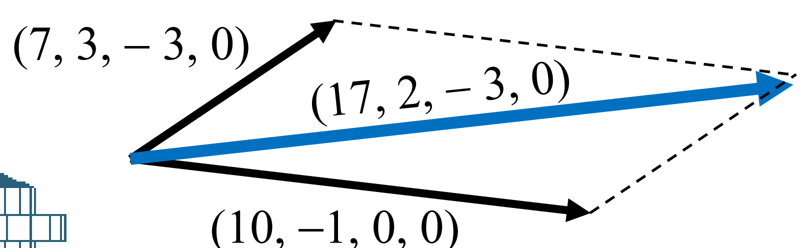
The sum of two vectors is a vector (obtained using parallelogram law).

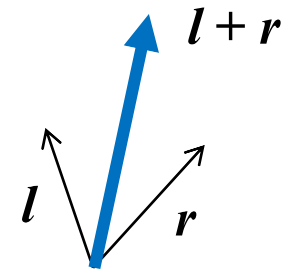




Adding several vectors at a point gives a vector in the average direction.

Example:

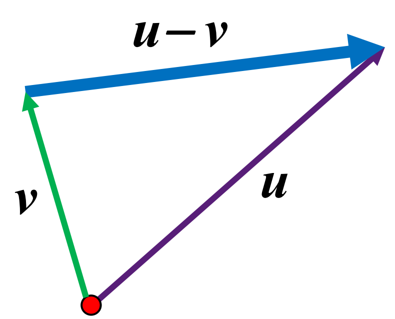




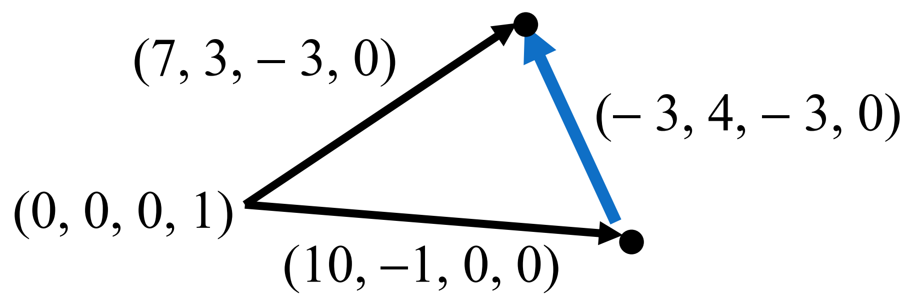
If the vectors have equal magnitude, their sum is a vector that makes equal angles with both vectors.

Vector Operations

The difference between two vectors is also a vector.



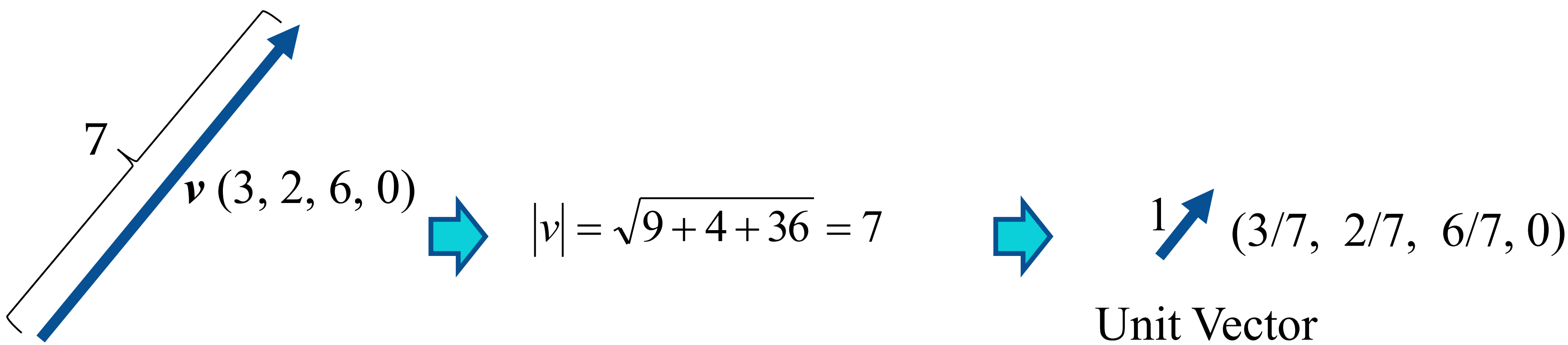
Example:



Unit Vector

- If ***v*** = (*x*, *y*, *z*) denotes a vector, its magnitude is given by |***v***| = $\sqrt{x^2 + y^2 + z^2}$

- A **unit vector** is a vector with magnitude 1.
- **Normalization** is the process of converting a vector to a unit vector by dividing each of its components by its magnitude.



Unit Vector

Given a *unit* vector ***v*** = (*x*, *y*, *z*) along a particular direction, a vector with magnitude *k* in that direction is obtained as

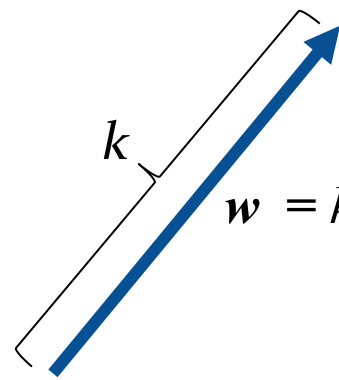
$\boldsymbol{w} = k\,\boldsymbol{v}$

(x, y, z)

$(x, y, z, 0)$

Unit Vector

$x^2 + y^2 + z^2 = 1.$



$\boldsymbol{w} = k\,\boldsymbol{v} = (kx, ky, kz)$

$= (kx, ky, kz, 0)$

Vector Dot Product

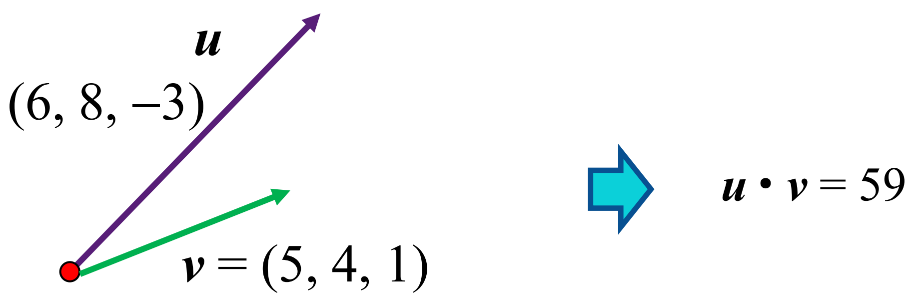
- The **dot product** of two vectors

  $\boldsymbol{v}_1 = (x_1, y_1, z_1)$ and

  $\boldsymbol{v}_2 = (x_2, y_2, z_2)$ is given by

  $$\boldsymbol{v}_1 \bullet \boldsymbol{v}_2 = x_1 x_2 + y_1 y_2 + z_1 z_2$$

- The dot product is a *scalar value,* not a vector.



Angle between two vectors

If $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ denote two vectors, then the angle ϕ between them is given by the following equation:

$$\cos\phi = \left(\frac{v_1}{|v_1|}\right) \bullet \left(\frac{v_2}{|v_2|}\right)$$

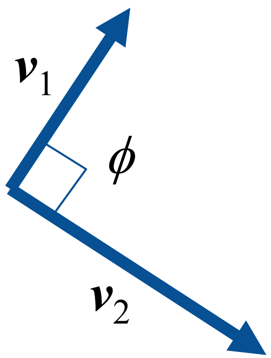
= The dot product of the corresponding **unit** vectors

<u>Example</u>: Compute the angle between the vectors (2, 3, 3) and (1, 1, 0):

- Normalize both vectors: (0.426, 0.64, 0.64), (0.707, 0.707, 0)
- Compute the dot product: 0.754 (= $\cos\phi$)
- $\phi = \cos^{-1}(0.754)$ = 41.06 Degs.

Orthogonality of Vectors

- Two vectors $\boldsymbol{v}_1$, $\boldsymbol{v}_2$ are perpendicular (orthogonal) to each other if and only if $\boldsymbol{v}_1 \bullet \boldsymbol{v}_2 = 0$.

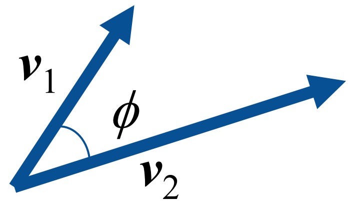


$$\boldsymbol{v}_1 \bullet \boldsymbol{v}_2 = 0 \quad \Leftrightarrow \quad \cos\phi = 0 \quad \Leftrightarrow \quad \phi = \pm 90 \text{ degs}$$

- <u>Example</u>: Show that the vectors (5, 2, –8) and (2, 7, 3) are perpendicular.
  - Compute the dot product: $10 + 14 - 24 = 0$
  - Since the dot product is 0, the vectors are orthogonal to each other. (There is no need to normalize the vectors)

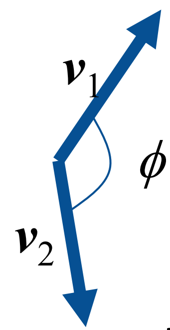
Relative Orientation

- A vector $\boldsymbol{v}_1$ is said to be oriented towards another vector $\boldsymbol{v}_2$ if $\boldsymbol{v}_1 \bullet \boldsymbol{v}_2 > 0$.



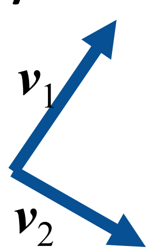
$\cos\phi > 0$
$\phi < \pi/2$

- A vector $\boldsymbol{v}_1$ is said to be oriented in the opposite direction as another vector $\boldsymbol{v}_2$ if $\boldsymbol{v}_1 \bullet \boldsymbol{v}_2 < 0$.



$\cos\phi < 0$
$\phi > \pi/2$

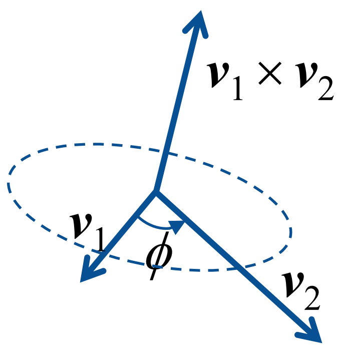
- The third possibility is that the vectors are orthogonal.

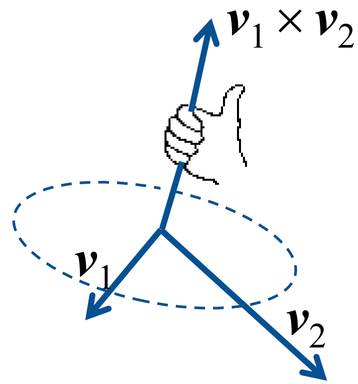


$\cos\phi = 0$
$\phi = \pi/2$

Vector Cross Product

- The cross product of two vectors $\boldsymbol{v}_1 = (x_1, y_1, z_1)$ and $\boldsymbol{v}_2 = (x_2, y_2, z_2)$ is a *vector* given by $\boldsymbol{v}_1 \times \boldsymbol{v}_2 = (y_1 z_2 - y_2 z_1,\ z_1 x_2 - z_2 x_1,\ x_1 y_2 - x_2 y_1)$.
- The above vector is perpendicular to both $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. The direction of $\boldsymbol{v}_1 \times \boldsymbol{v}_2$ is given by the right-hand rule





Vector Cross Product

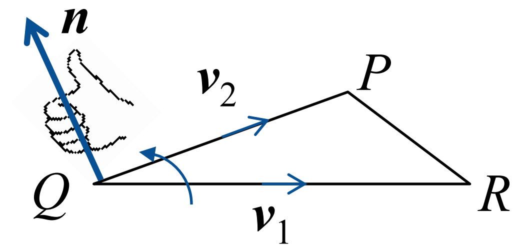
If ϕ is the angle between the vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$,

$$\sin\phi = \left|\left(\frac{v_1}{|v_1|}\right)\times\left(\frac{v_2}{|v_2|}\right)\right|$$

If $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are parallel, then $\boldsymbol{v}_1 \times \boldsymbol{v}_2$ is a zero vector.

Surface Normal Vector: Triangle

- Consider a triangle with vertices $P = (x_1, y_1, z_1)$, $Q = (x_2, y_2, z_2)$ and $R = (x_3, y_3, z_3)$.



  - We form two vectors at Q: $\boldsymbol{v}_1 = R - Q$, and $\boldsymbol{v}_2 = P - Q$.

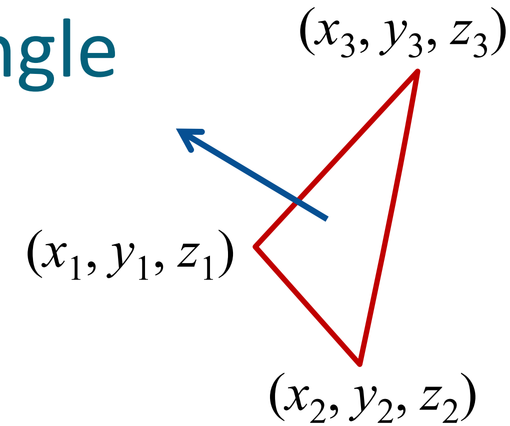
    $\boldsymbol{v}_1 = (x_3 - x_2,\ y_3 - y_2,\ z_3 - z_2)$, $\boldsymbol{v}_2 = (x_1 - x_2,\ y_1 - y_2,\ z_1 - z_2)$

  - The cross product $\boldsymbol{v}_1 \times \boldsymbol{v}_2$ gives the normal vector for the plane of the triangle. The normal vector is denoted by $\boldsymbol{n}$.

$$\boldsymbol{n} = \big(\ (y_3 - y_2)(z_1 - z_2) - (y_1 - y_2)(z_3 - z_2),\quad (z_3 - z_2)(x_1 - x_2) - (z_1 - z_2)(x_3 - x_2),\quad (x_3 - x_2)(y_1 - y_2) - (x_1 - x_2)(y_3 - y_2)\ \big)$$

$$= \big(\ y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2),\quad z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(x_1 - x_2),\quad x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)\ \big)$$

Surface Normal Vector: Triangle



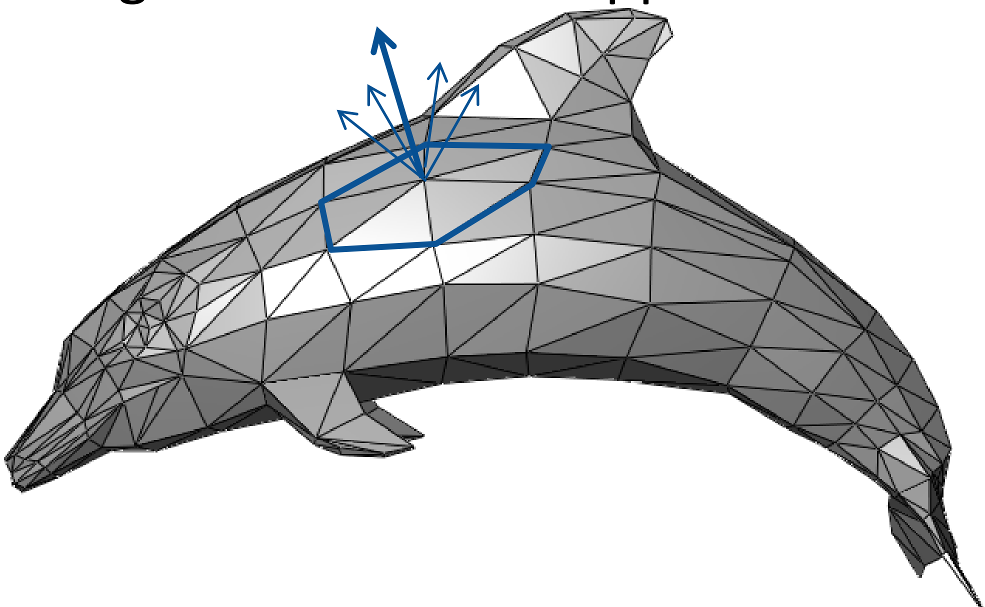
Input: 3 vertices of a triangle.

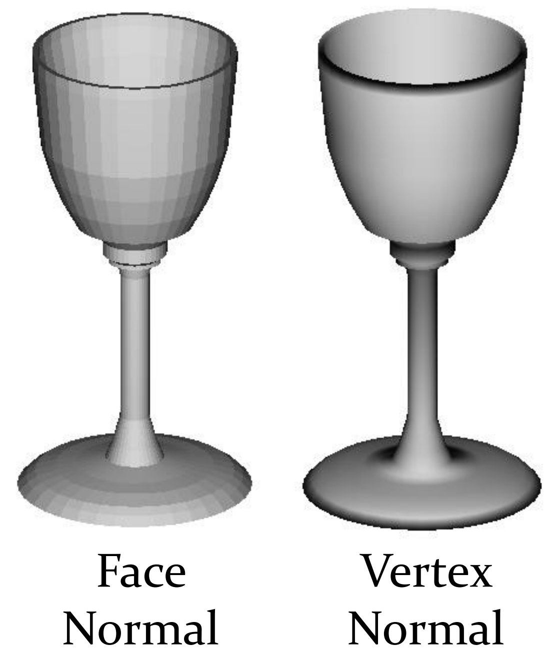
```
void normal(float x1, float y1, float z1,
            float x2, float y2, float z2,
            float x3, float y3, float z3 )
{
    float nx, ny, nz;
    nx = y1*(z2-z3)+ y2*(z3-z1)+ y3*(z1-z2);
    ny = z1*(x2-x3)+ z2*(x3-x1)+ z3*(x1-x2);
    nz = x1*(y2-y3)+ x2*(y3-y1)+ x3*(y1-y2);

    glNormal3f(nx, ny, nz);
}
```

Face Normal vs. Vertex Normal

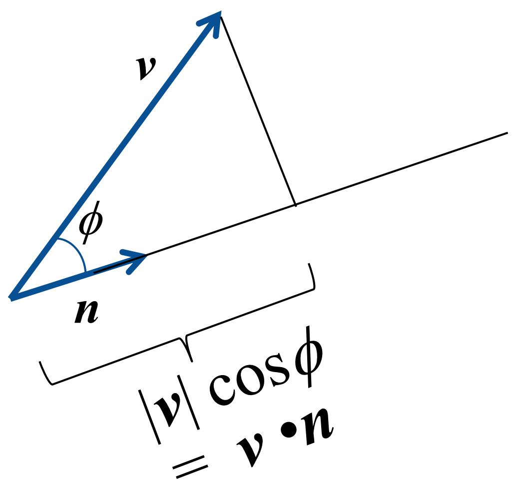
- Face normal: Assigning a single normal vector to a triangle gives a nearly uniform shade of colour to the whole triangle. The polygonal structure of the object becomes clearly visible.
- Vertex normal: The surface normal vectors at a vertex are all added together to get the average normal vector at that vertex. This gives a smoother appearance of the surface.

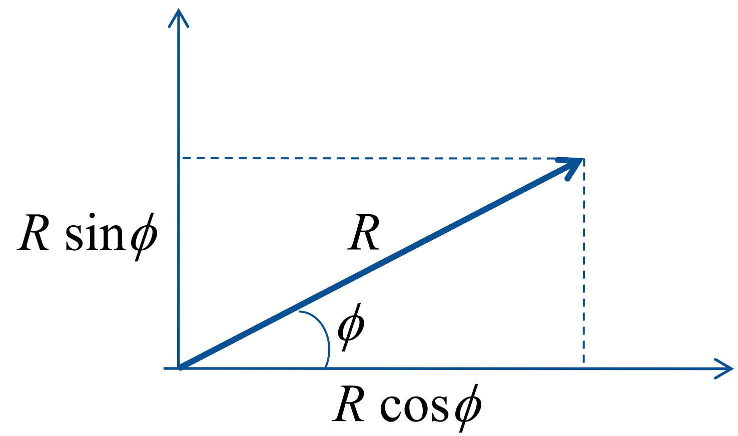




Projection of a Vector

Often, it is required to compute the component (projection) of a vector ***v*** along the direction of another <u>unit</u> vector ***n***.





Projections along orthogonal directions

- The length of the projection of ***v*** along ***n*** is ***v•n***
- The projected vector is (***v•n***) ***n*** (slide 7)

Matrices

- OpenGL uses 4x4 matrices for representing transformations.
- A 4x4 matrix may be stored in a two-dimensional array *a*[i][j]: i = row index (0..3), j = column index (0..3).
- Alternatively, the matrix can be stored in a single array m[k], k = 0..15, in either row-major order or column-major order. OpenGL always stores matrices in column-major order.

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad \begin{bmatrix} m_0 & m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 & m_7 \\ m_8 & m_9 & m_{10} & m_{11} \\ m_{12} & m_{13} & m_{14} & m_{15} \end{bmatrix} \qquad \begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix}$$

(Row Major Order) (Column Major Order)

OpenGL

Matrices

- Identity Matrix

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- For any matrix **A**, **AI = IA = A**
- OpenGL Example:

```
float matrix[16]={0.5, 3.0, 0.1, 0, 0, 10., 6.0, 0,
               8.0, 1.0,-4.2, 0, -2.0, 0, 9.0, 1.0};
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(matrix);
```

$$\Rightarrow \begin{bmatrix} 0.5 & 0 & 8 & -2 \\ 3 & 10 & 1 & 0 \\ 0.1 & 6 & -4.2 & 9 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrix Product

The multiplication of a 4x4 matrix and a 4x1 vector gives a 4x1 vector.

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00}x + a_{01}y + a_{02}z + a_{03} \\ a_{10}x + a_{11}y + a_{12}z + a_{13} \\ a_{20}x + a_{21}y + a_{22}z + a_{23} \\ a_{30}x + a_{31}y + a_{32}z + a_{33} \end{bmatrix}$$

Example:

$$\begin{bmatrix} 3 & 0 & 1 & 1 \\ -2 & 1 & 5 & 0 \\ 1 & -1 & 2 & 1 \\ 0 & 4 & 1 & -3 \end{bmatrix} \begin{bmatrix} 2 \\ 7 \\ -2 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ -7 \\ -8 \\ 23 \end{bmatrix}$$

Matrix Product

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} . & . & . & . \\ . & . & \boxed{.} & . \\ . & . & . & . \\ . & . & . & . \end{bmatrix}$$

$$c_{12} = a_{10}\, b_{02} + a_{11}\, b_{12} + a_{12}\, b_{22} + a_{13}\, b_{32}$$

General formula: $c_{ij} = \sum_{k=0}^{3} a_{ik} b_{kj}$

Example:

$$\begin{bmatrix} 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & -1 & -5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.7 & 0 & 0.7 & 0 \\ 0 & 1 & 0 & 0 \\ -0.7 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1.4 & 2 \\ 0 & 1 & 0 & 0 \\ 1.4 & 0 & 0 & -5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrix multiplication is **non-commutative**. In general, $AB \neq BA$

Transformation Matrix

The transformation of a point $(x, y, z, 1)$ to another point $(x', y', z', 1)$ can be expressed as a matrix-vector multiplication:

$$\begin{bmatrix} x'_p \\ y'_p \\ z'_p \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

Transformed point

A general transformation matrix

Input point

Translation Matrix

- The translation of a point (x, y, z, 1) by (a, b, c) yields another point (x+a , y+b , z+c , 1)

$$\begin{bmatrix} x+a \\ y+b \\ z+c \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{Translation Matrix}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- OpenGL function: `glTranslatef(a, b, c)`

Translation Matrix

The translation matrix has no effect on a vector (*x*, *y*, *z*, 0):

$$\begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{Translation Matrix}} \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

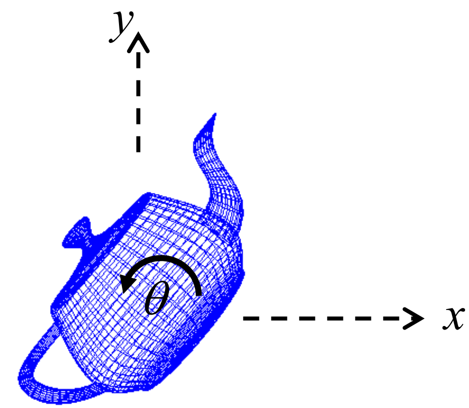
Scale Matrix

- The scaling of a point $(x, y, z, 1)$ by factors (a, b, c) yields another point $(xa, yb, zc, 1)$

$$\begin{bmatrix} xa \\ yb \\ zc \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{Scale Matrix}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- OpenGL function: `glScalef(a, b, c)`

Rotation About the Z-axis



- Equations:

  $x' = x\cos\theta - y\sin\theta$

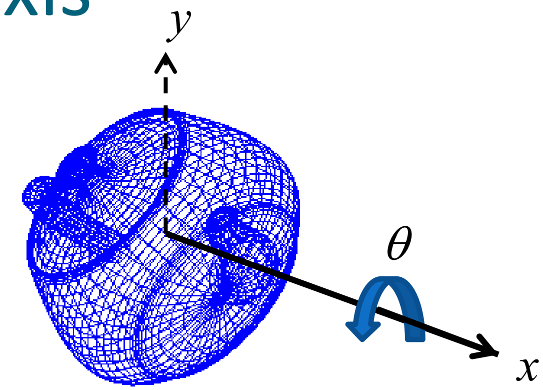
  $y' = x\sin\theta + y\cos\theta$

  $z' = z$

- Matrix Form: $$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- OpenGL function: `glRotatef(theta, 0, 0, 1)`

Rotation About the X-axis



- Equations:

  $x' = x$

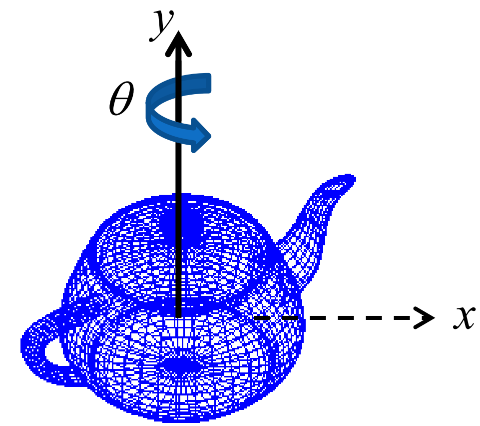
  $y' = y\cos\theta - z\sin\theta$

  $z' = y\sin\theta + z\cos\theta$

- Matrix Form: $$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- OpenGL function: `glRotatef(theta, 1, 0, 0)`

Rotation About the Y-axis



- Equations:

  $x' = x\cos\theta + z\sin\theta$

  $y' = y$

  $z' = -x\sin\theta + z\cos\theta$

- Matrix Form:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- OpenGL function: `glRotatef(theta, 0, 1, 0)`

Custom Transformations

User-defined transformations can be represented in matrix form and applied with other transforms.

```
float myMatrix[16]={0.5, 3.0, 0.1, 0,
                      0, 10., 6.0, 0,
                      8.0, 1.0,-4.2, 0,
                     -2.0, 0, 9.0, 1.0};
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(...)
glPushMatrix();
  glTranslatef(5, 2, -3);
  glMultMatrixf(myMatrix);
  glRotatef(25, 0, 1, 0);
  glutSolidTeapot(1);
glPopMatrix();
```

$$\begin{bmatrix} 0.5 & 0 & 8 & -2 \\ 3 & 10 & 1 & 0 \\ 0.1 & 6 & -4.2 & 9 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Teapot rotated→transformed using myMatrix →translated

Affine Transformation

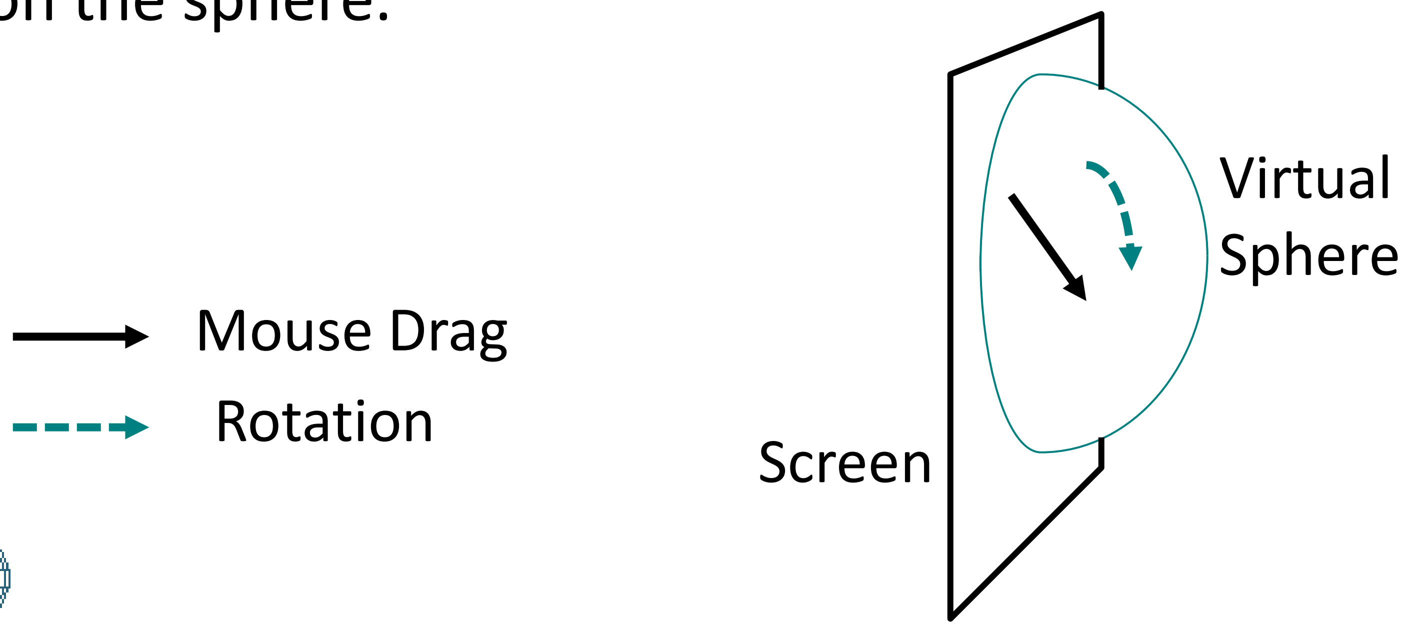
- A general linear transformation followed by a translation is called an affine transformation.
- Matrix form:

$$\begin{bmatrix} x'_p \\ y'_p \\ z'_p \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

- Translation, rotation, scaling and shear transformations are all affine transformations.
- Under an affine transformation, line segments transform into line segments, and parallel lines transform into parallel lines.

Virtual Trackball

- A user interface for drag-rotating an object.
- Assume that the objects displayed on the screen are attached to a virtual sphere.
- When the mouse is dragged from one point to another on the screen, a corresponding path of rotation is generated on the sphere.

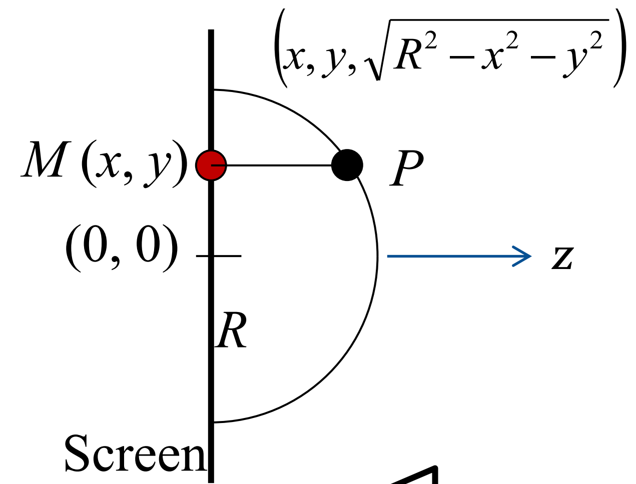


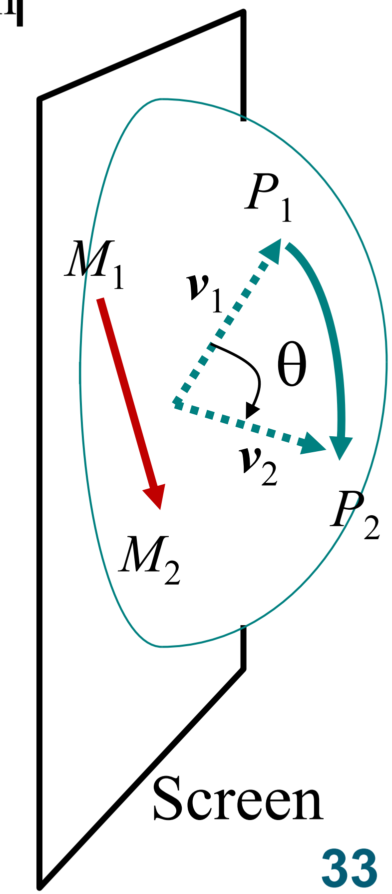
Virtual Trackball

- Let M_1M_2 be the path through which the mouse is dragged, and P_1, P_2, the corresponding points on the virtual sphere.
- The angle of rotation is the angle between unit vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$

$$\theta = \cos^{-1}(v_1 \bullet v_2)$$

- The axis of rotation is the axis perpendicular to both $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, given by $\boldsymbol{v}_1 \times \boldsymbol{v}_2 = (l, m, n)$
- Use glRotatef(θ, l, m, n) to rotate the object.

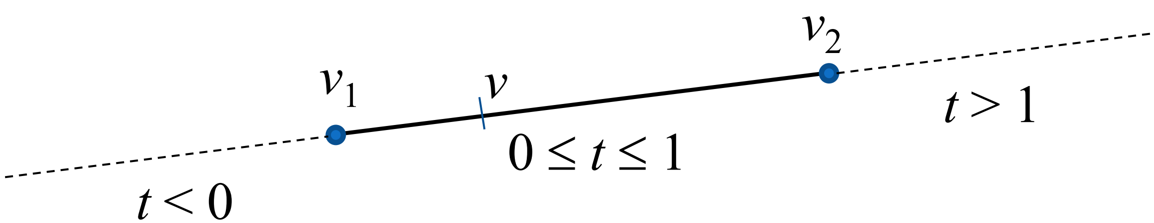




Linear Interpolation

Linear interpolation is useful in computing an in-between value, given the values v_1, v_2 of some attribute at the end points of a path.

$$v = \mathbf{(1–}\boldsymbol{t}\mathbf{)}\, v_1 + \boldsymbol{t}\, v_2\ , \qquad 0 \le t \le 1.$$



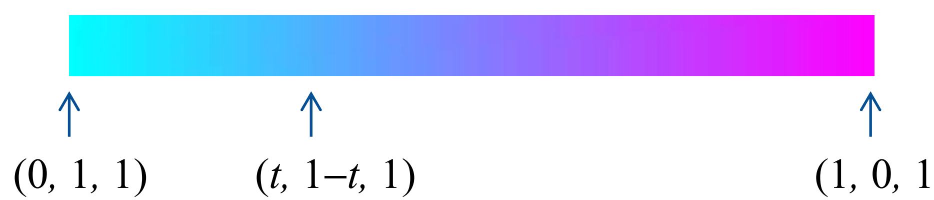
Example:

$v_1 = (0, 1, 1)$

$v_2 = (1, 0, 1)$

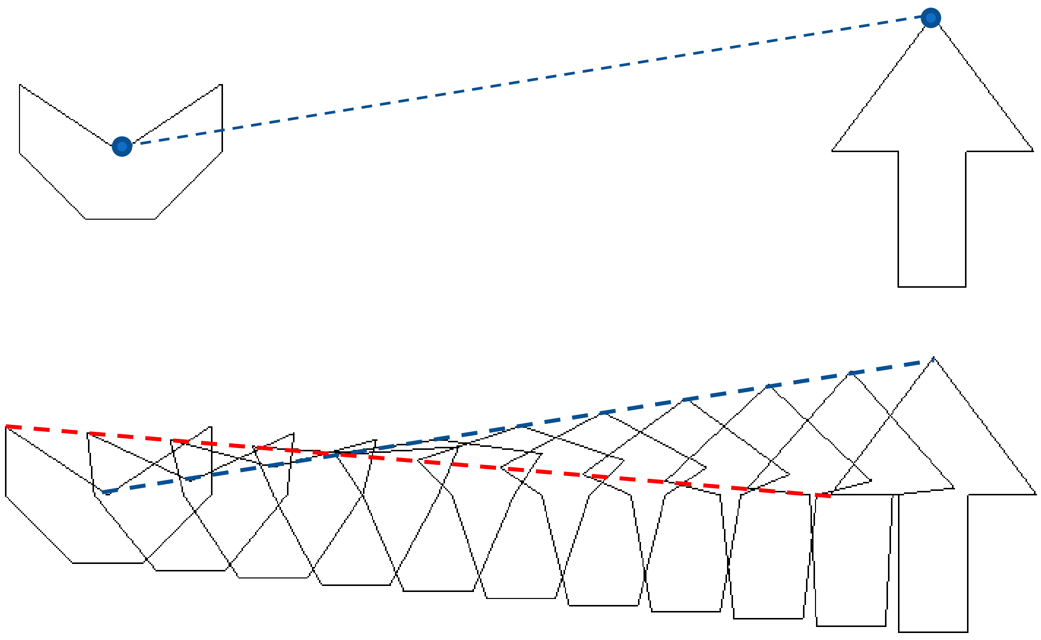
$v\ = (1 - t)(0, 1, 1) + t\,(1, 0, 1)$

$\quad = (\,t,\ 1 - t,\ 1)$



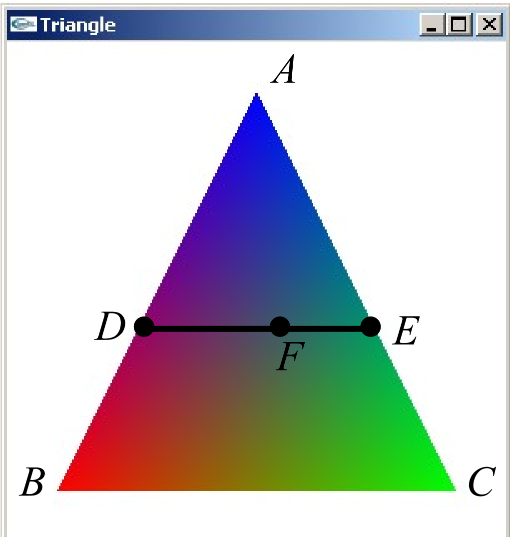
Linear Interpolation

Interpolating between corresponding points of two shapes generates a **shape-tween**.



Bi-Linear Interpolation

- Given the values of an attribute (such as colour) at the vertices of a triangle, bi-linear interpolation is used to obtain the values at the interior points.



$$\boldsymbol{D} = (1-t)\,\boldsymbol{A} + t\,\boldsymbol{B}$$
$$\boldsymbol{E} = (1-t)\,\boldsymbol{A} + t\,\boldsymbol{C}$$
$0 \le t \le 1$

$$\boldsymbol{F} = (1-s)\,\boldsymbol{D} + s\,\boldsymbol{E}$$
$0 \le s \le 1$

$$\boldsymbol{F} = (1-k_1-k_2)\,\boldsymbol{A} + k_1\boldsymbol{B} + k_2\boldsymbol{C}$$

Convex Combination

- Interpolate along the two edges *AC, BC* using a single parameter *t* , to get *D, E.*
- Interpolate along *DE* using a second parameter *s*, to get *F*