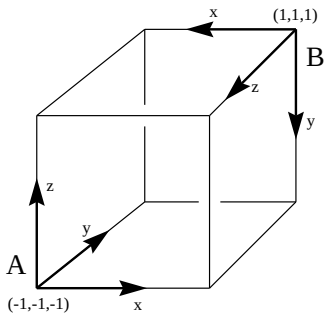
Transformation Matrices = Axes



$$ {}_{W}T_{A} = \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

$$ {}_{A}T_{B} = \begin{pmatrix} -1 & 0 & 0 & 2 \\ 0 & -1 & 0 & 2 \\ 0 & 0 & -1 & 2 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

Copyright © Pat Hanrahan

Transformations as Coordinate Systems

1. **Applying a transformation is equivalent to converting points from one coordinate system to another coordinate system.**

$$ {}_{B}T_{A} $$

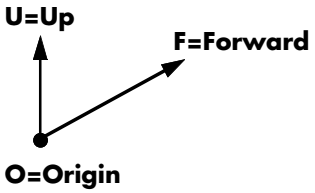
2. **The transformation in the other direction is the inverse**

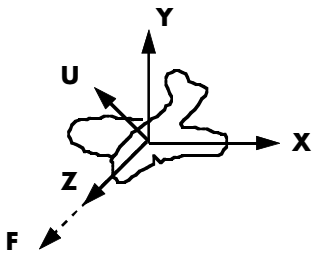
$$ {}_{A}T_{B} = {}_{B}T_{A}^{-1} $$

3. **The columns of a transformation matrix may be interpreted as the x, y, z, and origin of the axes defining a coordinate system**

Copyright © Pat Hanrahan

Orientation Matrix





W = O

Z = F

X = U x Z

Y = Z x X

T=(X Y Z W)

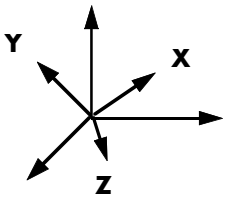
Copyright © Pat Hanrahan

Rotation Matrix = Orthogonal Matrix

$$R^T R = \begin{pmatrix} \hat{X} \\ \hat{Y} \\ \hat{Z} \end{pmatrix} \begin{pmatrix} \hat{X} & \hat{Y} & \hat{Z} \end{pmatrix} = \begin{pmatrix} \hat{X}\bullet\hat{X} & \hat{X}\bullet\hat{Y} & \hat{X}\bullet\hat{Z} \\ \hat{Y}\bullet\hat{X} & \hat{Y}\bullet\hat{Y} & \hat{Y}\bullet\hat{Z} \\ \hat{Z}\bullet\hat{X} & \hat{Z}\bullet\hat{Y} & \hat{Z}\bullet\hat{Z} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = I$$

$$R = \begin{pmatrix} \hat{X} & \hat{Y} & \hat{Z} \end{pmatrix}$$



$$\boxed{R^{-1} = R^T}$$

Copyright © Pat Hanrahan

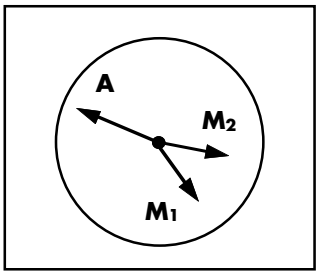
Rolling Ball

Control rotation by mouse difficult

mouse x -> pan

mouse y -> tilt

Rolling Ball metaphor



$$\vec{M} = \begin{pmatrix} x & y & \sqrt{1-x^2-y^2} \end{pmatrix}$$

$$\vec{A} = \vec{M}_1 \times \vec{M}_2$$

$$R_1 = \begin{pmatrix} \hat{M}_1 & \hat{A} \times \hat{M}_1 & \hat{A} \end{pmatrix}$$

$$R_2 = \begin{pmatrix} \hat{M}_2 & \hat{A} \times \hat{M}_2 & \hat{A} \end{pmatrix}$$

$$R = R_2 R_1^T$$

Copyright © Pat Hanrahan

Composition of Transformations

- **Sequence of transforms = single transform**

  P′ = A (B (C (D (P))))

  = (A B C D) P

  = T P

- **Inverse of a sequence**

  P = (D^{-1} C^{-1} B^{-1} A^{-1}) (A B C D) P

  Therefore

  (A B C D) $^{-1}$ = D^{-1} C^{-1} B^{-1} A^{-1}

  1. Apply inverses in the reverse order

  2. Concatenate and numerically invert matrix

Copyright © Pat Hanrahan

Current Transformation Matrix (CTM)

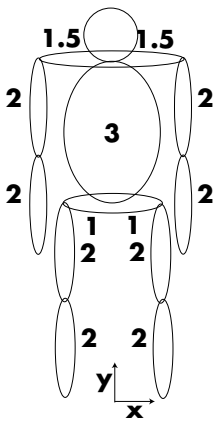
- **Graphics systems maintain a *current transformation matrix (CTM).* All geometry is transformed by the CTM.**
- **The CTM defines the current or *object* or *local* coordinate system. All geometry is defined in the current coordinate system.**
- **Transformation commands are concatenated onto the ctm. Note: The last transformation specified is the first to be performed.**

$$CTM' = CTM * T$$

- **The CTM may be pushed onto a transformation stack, and latter popped to restore a previous state.**

Copyright © Pat Hanrahan

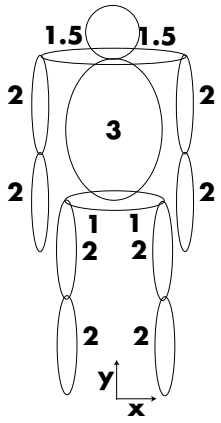
Skeleton




```
body
  torso
    head
    shoulder
      larm
        upperarm
        lowerarm
        hand
      rarm
        upperarm
        lowerarm
        hand
    hips
      lleg
        upperleg
        lowerleg
        foot
      rleg
        upperleg
        lowerleg
        foot
```


Copyright © Pat Hanrahan

Skeleton




```
translate(0,4,0)
torso();
pushmatrix();
    translate(0,3,0);
    shoulder();
    pushmatrix();
      rotatey(necky);
      rotatex(neckx);
      head();
    popmatrix();
    pushmatrix();
      translate(1.5,0,0);
      rotatex(lshoulderx);
      upperarm();
      pushmatrix();
        translate(0,-1,0);
        rotatex(lelbowx);
        lowerarm();
        ...
      popmatrix();
    popmatrix();
...
```


Copyright © Pat Hanrahan

Polyhedral Models

Cube as a sequence of polygons

```
// front face
glBegin(GL_POLYGON);
  glVertex3f(-1.,-1,  1.);
  glVertex3f( 1.,-1,  1.);
  glVertex3f( 1., 1,  1.);
  glVertex3f(-1., 1,  1.);
glEnd();

// back face
glBegin(GL_POLYGON);
  glVertex3f( 1.,-1, -1.);
  glVertex3f(-1.,-1, -1.);
  glVertex3f(-1., 1, -1.);
  glVertex3f( 1., 1, -1.);
glEnd();

// other faces ...
```

Note: CCW orientation of verts

```
frontface() {
  glBegin(GL_POLYGON);
    glVertex3f(-1.,-1,  1.);
    glVertex3f( 1.,-1,  1.);
    glVertex3f( 1., 1,  1.);
    glVertex3f(-1., 1,  1.);
  glEnd();
}

cube() {
  frontface();
  glRotatef(90.,0.,1.,0.);
  frontface();
  glRotatef(90.,0.,1.,0.);
  frontface();
  glRotatef(90.,0.,1.,0.);
  frontface();
  glRotatef(90.,0.,1.,0.);

  glRotatef(90.,1.,0.,0.);
  frontface();
  glRotatef(180.,1.,0.,0.);
  frontface();
}
```


Copyright © Pat Hanrahan

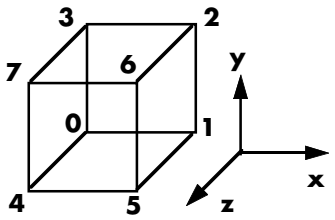
Points/Polygons

```
typedef point float[3];

point verts[8] = {
  {-1.,-1.,-1.},
  { 1.,-1.,-1.},
  { 1., 1.,-1.},
  {-1., 1.,-1.},
  {-1.,-1., 1.},
  { 1.,-1., 1.},
  { 1., 1., 1.},
  {-1., 1., 1.}
};


face(int a, int b,
     int c, int d) {

  glBegin(GL_POLYGON);
    glVertex3fv(verts[a]);
    glVertex3fv(verts[b]);
    glVertex3fv(verts[c]);
    glVertex3fv(verts[d]);
  glEnd();
}
```



Separate geometry and topology!

```
cube() {
  face(0,3,2,1);
  face(2,3,7,6);
  face(0,4,7,3);
  face(1,2,6,5);
  face(4,5,6,7);
  face(0,1,5,4);
}
```


Copyright © Pat Hanrahan

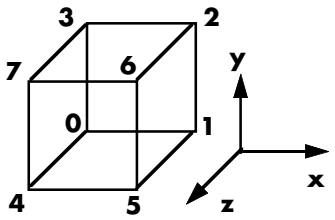
Points/Polygons

```
typedef point float[3];

point verts[8] = {
  {-1.,-1.,-1.},
  { 1.,-1.,-1.},
  { 1., 1.,-1.},
  {-1., 1.,-1.},
  {-1.,-1., 1.},
  { 1.,-1., 1.},
  { 1., 1., 1.},
  {-1., 1., 1.}
};

typedef quad int[4];

quad quads[6] = {
  {0,3,2,1},
  {2,3,7,6},
  {0,4,7,3},
  {1,2,6,5},
  {4,5,6,7},
  {0,1,5,4}
};
```




```
cube() {
  for(int i=0; i<6; i++)
    face(
      quads[i][0],
      quads[i][1],
      quads[i][2],
      quads[i][3]);
}
```


Copyright © Pat Hanrahan