**Coordinate systems**

- Also known as “frame”
- Usually orthonormal
  - orthographic = axes perpendicular to each other
  - normal = the length of axes is unit
- Right handed coordinates
  - xyz = right thumb, index, middle
  - point right thumb at z, x turns to y along the fingers
- p = [x, y, z]'
- Modeling:
  - Object coordinates
  - World coordinates
  - Camera coordinates
- Rendering pipeline
  - Normalized device coordinates
  - Window coordinates

**Vectors**

- In graphics we deal with vectors of 2, 3, or 4 dimensions
  - we'll represent a vector as an n-tuple or vertical n-matrix
  - most rules don’t care about the dimension
- Definition: **vector is a displacement**
  - it has a direction
  - and a length
  - but no location
- The **difference** between two points is a vector
  - \( \mathbf{v} = \mathbf{Q} - \mathbf{P} \)
- The **sum** of a point and a vector is a point
  - \( \mathbf{P} + \mathbf{v} = \mathbf{Q} \)

**Vector operations**

- Vector addition
  - the diagonal of a parallelogram or concatenate the vectors
  - \( \mathbf{a} = (1,2,3), \mathbf{b} = (4,5,6) \)
  - \( \mathbf{a} + \mathbf{b} = (1+4, 2+5, 3+6) \)
- Multiplication by scalar
  - change the length, maybe flip direction
  - \( \mathbf{a} = (1,2,3) \quad s \mathbf{a} = (s, 2s, 3s) \)
  - \( 5 \mathbf{a} = (5, 10, 15) \)
- Subtraction
  - addition and negative scaling
  - \( \mathbf{a} - \mathbf{b} = \mathbf{a} + -\mathbf{b} \)

**Linear combinations**

- A linear combination of two vectors \( \mathbf{v} \) and \( \mathbf{w} \)
  - \( a\mathbf{v} + b\mathbf{w} \)
- More generally
  - \( \mathbf{w} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_n\mathbf{v}_n \)
- Affine combination
  - linear combination with \( a_1 + a_2 + \cdots + a_n = 1 \)
- Convex combination
  - affine combination with \( a_i \geq 0 \)
  - aka partitioning of a unity
Examples

- Convex combination of two vectors
  - interpolates the end points
  - a line between the end points, parameterized by $0 \leq a \leq 1$
  - what does an affine combination do?
- of three vectors
  - defines the triangle
  - affine combination?
- You can use the affine coefficients as coordinates
  - called barycentric coordinates

\[
\begin{align*}
\mathbf{v} &= (1-a)\mathbf{v}_1 + a\mathbf{v}_2 \\
&= \mathbf{v}_1 + a(\mathbf{v}_2 - \mathbf{v}_1)
\end{align*}
\]

\[
\begin{align*}
\mathbf{v} &= (1-a-b)\mathbf{v}_1 + a\mathbf{v}_2 + b\mathbf{v}_3
\end{align*}
\]

Vector magnitude

- Representation of a vector
  - $\mathbf{w} = (w_1, w_2, \ldots, w_n)$
- Its magnitude
  - use the Pythagorean theorem
  - $|\mathbf{w}| = \sqrt{w_1^2 + w_2^2 + \cdots + w_n^2}$
- Unit vector
  - normalize a vector by dividing it by its magnitude
  - the length will be 1
  - represents a direction
  - $\hat{\mathbf{w}} = \mathbf{w}/|\mathbf{w}|$

Dot product

- Definition
  - $\mathbf{v} = (v_1, v_2, \ldots, v_n)$, $\mathbf{w} = (w_1, w_2, \ldots, w_n)$
  - $d = \mathbf{v} \cdot \mathbf{w} = \mathbf{v}'\mathbf{w}' = \sum_{i=1}^{n} v_i w_i$
- Properties
  - $\mathbf{a} \cdot \mathbf{b} = \mathbf{b} \cdot \mathbf{a}$
  - $(\mathbf{a} + \mathbf{c}) \cdot \mathbf{b} = \mathbf{a} \cdot \mathbf{b} + \mathbf{c} \cdot \mathbf{b}$
  - $(s\mathbf{a}) \cdot \mathbf{b} = s(\mathbf{a} \cdot \mathbf{b})$
  - $|\mathbf{b}|^2 = \mathbf{b} \cdot \mathbf{b}$
- Example:
  - $|\mathbf{a} - \mathbf{b}|^2 = (\mathbf{a} - \mathbf{b}) \cdot (\mathbf{a} - \mathbf{b}) = \mathbf{a} \cdot \mathbf{a} - 2\mathbf{a} \cdot \mathbf{b} + \mathbf{b} \cdot \mathbf{b}$
  - $= \mathbf{a} \cdot \mathbf{a} - \mathbf{b} \cdot \mathbf{a} - \mathbf{a} \cdot \mathbf{b} + \mathbf{b} \cdot \mathbf{b}$
  - $= \mathbf{a} \cdot \mathbf{a} - 2\mathbf{a} \cdot \mathbf{b} + \mathbf{b} \cdot \mathbf{b}$

Dot product

- The angle between two vectors
  - $\mathbf{p} = (|\mathbf{p}| \cos \alpha, |\mathbf{p}| \sin \alpha)$
  - $\mathbf{q} = (|\mathbf{q}| \cos(\alpha + \beta), |\mathbf{q}| \sin(\alpha + \beta))$
  - $\mathbf{p} \cdot \mathbf{q} = |\mathbf{p}| |\mathbf{q}| \cos(\alpha + \beta) \cos \alpha + |\mathbf{p}| |\mathbf{q}| \sin(\alpha + \beta) \sin \alpha$
  - $= |\mathbf{p}| |\mathbf{q}| \cos(\alpha + \beta - \alpha) = |\mathbf{p}| |\mathbf{q}| \cos \beta$
  - $\Rightarrow \cos \beta = \hat{\mathbf{p}} \cdot \hat{\mathbf{q}}$
- The sign of the dot product
  - follows the sign of cosine
  - What is $\mathbf{a}^\perp$, the perpendicular vector to $\mathbf{a} = (a_x, a_y)$?
  - $\mathbf{p} \cdot \mathbf{q} > 0$  $\mathbf{p} \cdot \mathbf{q} = 0$  $\mathbf{p} \cdot \mathbf{q} < 0$
  - $\mathbf{a}^\perp = (-a_y, a_x)$
**Dot as a projection**

- Project point C onto line AB
  - find vectors $\mathbf{a} = B - A / |B - A|$ and $\mathbf{c} = C - A$
  - project $\mathbf{c}$ onto $\mathbf{a}$: $|\mathbf{c}| \cos \angle (\mathbf{a}, \mathbf{c}) \hat{\mathbf{a}} = |\mathbf{a}| (\hat{\mathbf{a}} \cdot \hat{\mathbf{c}}) \mathbf{a} = \left( \frac{\mathbf{a} \cdot \mathbf{c}}{|\mathbf{a}|^2} \right) \mathbf{a} = \frac{\mathbf{a} \cdot \mathbf{c}}{\mathbf{a} \cdot \mathbf{a}}$
  - add the projection (vector) to A (point): $A + \frac{\mathbf{a} \cdot \mathbf{c}}{\mathbf{a} \cdot \mathbf{a}}$

- What is the (perpendicular) distance of C from the line AB?
  - project $\mathbf{c}$ to $\hat{\mathbf{a}}^\perp$: $|\hat{\mathbf{a}}^\perp \cdot \mathbf{c}|$
  - or $|\mathbf{c} - (\hat{\mathbf{a}} \cdot \hat{\mathbf{c}}) \hat{\mathbf{a}}|$

**Application: reflection vector**

- Reflect the ray of light $\mathbf{a}$ from a plane with normal vector $\mathbf{n}$
  - angle of incidence = angle of exit
- Break $\mathbf{a}$ into two components
  - vertical $\mathbf{m} = \frac{\mathbf{a} \cdot \mathbf{n}}{\mathbf{n} \cdot \mathbf{n}} \mathbf{n}$
  - horizontal $\mathbf{a} - \mathbf{m}$
- Take the horizontal component and add the reflected vertical component
  - $\mathbf{r} = \mathbf{a} - \mathbf{m} - \mathbf{m} = \mathbf{a} - 2\mathbf{m} = \mathbf{a} - 2 \frac{\mathbf{a} \cdot \mathbf{n}}{\mathbf{n} \cdot \mathbf{n}} \mathbf{n}$

**Cross product**

- Definition in 3D: $\mathbf{a} \times \mathbf{b} = (a_x b_z - a_z b_y, a_z b_x - a_x b_z, a_x b_y - a_y b_x)$
  - $i, j, k$ are unit x, y, z vectors
  - $i \times j = k$
  - $j \times k = i$
  - $k \times i = j$

- Like dot, cross is linear $\mathbf{a} \times (\mathbf{b} + \mathbf{c}) = \mathbf{a} \times \mathbf{b} + \mathbf{a} \times \mathbf{c}$
  - $\mathbf{a} \times \mathbf{a} = \mathbf{0}$

- but it’s antisymmetrical $\mathbf{a} \times \mathbf{b} = -\mathbf{b} \times \mathbf{a}$
- what is $\mathbf{a} \times \mathbf{a}$?
  - must be zero: $\mathbf{a} \times \mathbf{a} = -\mathbf{a} \times \mathbf{a} = 0 = \mathbf{0}$
  - also the cross of any two parallel vectors must be zero

- Geometric interpretation
  - $\mathbf{a} \times \mathbf{b}$ is perpendicular to both $\mathbf{a}$ and $\mathbf{b}$
  - $|\mathbf{a} \times \mathbf{b}| = \text{the area of the parallelogram determined by } \mathbf{a} \text{ and } \mathbf{b}$
  - $|\mathbf{a}||\mathbf{b}||\sin \theta$, where $\theta$ is the angle between the vectors
  - the direction of $\mathbf{a} \times \mathbf{b}$ follows from the right hand rule

- Example:
  - Find a normal vector to the triangle ABC
  - $(B - A) \times (C - A)$
**Transformations: Translation**

- Translation of a geometric primitive is done by adding a vector to every point on it.
- \( Q = P + t \)
- \( t = [tx, ty, tz]' \)
- Inverse: translate back 
  \(-t = [-tx, -ty, -tz]'\)
- Another interpretation 
  - move the local coordinate system of an object 
  - all the points therefore move as well, though the remain the same in the local coordinates

**2D rotation**

- Rotation in a plane (ccw) by \( \alpha \)
  - express \( p \) with length and angle: 
    \[ p = [x, y]' = [\rho \cos \alpha, \rho \sin \alpha]' \]
  - \( q \) is \( p \) rotated further by angle \( \beta \)
    \[ q = [\rho \cos \alpha \cos \beta - \rho \sin \alpha \sin \beta, \rho \cos \alpha \sin \beta + \rho \sin \alpha \cos \beta]' \]
    \[ = [x \cos \beta - y \sin \beta, x \sin \beta + y \cos \beta]' \]
- Inverse 
  - rotate \( q \) back by \(-\beta\) to get \( p \) again 
  - or transpose the rotation matrix 
    \[ \begin{bmatrix} \cos \beta & -\sin \beta \\ \sin \beta & \cos \beta \end{bmatrix} \]

**Aside: Matrix-vector multiplication**

- Two ways of thinking and calculating the outcome 
  \[ \begin{bmatrix} \cos \beta & -\sin \beta \\ \sin \beta & \cos \beta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \]

- Component at a time 
  - dot each row at a time with the vector 
    "the x component will be \((\cos \beta, -\sin \beta) \cdot (x, y) = x \cos \beta - y \sin \beta\) and the y component ..."

- Linear combination of vectors (columns) 
  - the new "x vector" is \((\cos \beta, \sin \beta)\) 
  - the new "y" is \((-\sin \beta, \cos \beta)\) 
  - and the outcome is 
    \[ \begin{bmatrix} \cos \beta \\ \sin \beta \end{bmatrix} x + \begin{bmatrix} -\sin \beta \\ \cos \beta \end{bmatrix} y = \begin{bmatrix} x \cos \beta - y \sin \beta \\ x \sin \beta + y \cos \beta \end{bmatrix} \]

**Rotation around a coordinate axis**

- \( q = R \ p \)
- What's the 3D rotation matrix around z? 
  \[ R_z = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

- Around x? y? 
  \[ R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix} \quad R_y = \begin{bmatrix} \cos \alpha & 0 & \sin \alpha \\ 0 & 1 & 0 \\ -\sin \alpha & 0 & \cos \alpha \end{bmatrix} \]
Rotation around a point

- Rotation takes normally place around the origin
  - origin is the only stationary point
  - so object away from it also moves

- If you want to rotate around a certain point, e.g., center of object
  - move the center to origin
  - rotate
  - move the center back

\[ q = T_c R T^{-1} p \]

Quaternions: Rotation about an arbitrary axis

- Axis \( \mathbf{n} = [x, y, z]' \), by an angle \( \alpha \)
- Use unit quaternions \( q = [q_0, q_1, q_2, q_3]' \)
  - unit: \( q' q = 1 \)
  - a generalized imaginary number
    - \( q_0 \) the real part
    - \( q_1 \) the \( i \) component, \( q_2 \), \( q_3 \) \( j \), \( k \)
    - \( q_0 = \cos(\alpha/2) \), \( [q_1, q_2, q_3]' = \sin(\alpha/2) \ \mathbf{n} / |\mathbf{n}| \)
- from quaternion to rotation matrix

Inverse of rotation

- You can read the new coordinate axes after the rotation from the columns
  - expressed in coordinates of the old frame
- Assume old coordinate axes are unit vectors, perpendicular to each other
  - they remain so after rotation
- What happens when we multiply \( R' \) with \( R \)?

\[
R' = \begin{bmatrix}
  x_x & x_y & x_z \\
  y_x & y_y & y_z \\
  z_x & z_y & z_z \\
\end{bmatrix}
\]

\[
R'R = \begin{bmatrix}
  x_x & x_y & x_z \\
  y_x & y_y & y_z \\
  z_x & z_y & z_z \\
\end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
\]

- so even in general the transpose of the rotation matrix is the inverse! \( R' = R^{-1} \)

Coordinate systems again

- A 3D coordinate system consists of
  - origin \( \mathbf{O} \)
  - vectors \( \mathbf{x}, \mathbf{y}, \mathbf{z} \)
- Form a matrix \( \begin{bmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} & \mathbf{O} \end{bmatrix} \)
- Form a point
  - start from the origin, "move" it by adding some multiples of the vectors
  \( \begin{bmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} \end{bmatrix} \begin{bmatrix} a & b & c & 1 \end{bmatrix} \)
- Form a vector
  - just add multiples of vectors, ignore the origin
  \( \begin{bmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} \end{bmatrix} \begin{bmatrix} a & b & c \end{bmatrix} \)
Homogeneous coordinates

- Move input, output, and the transformation matrix into 4D:
  - \([x, y, z]' \Rightarrow [x, y, z, w]'\)
  - \(w = 1\) for a point
  - \(w = 0\) for a vector
- Generalize even more: allow any \(w\)
  - also, any scalar multiple is the same point
  - \([x, y, z, w]' = [ax, ay, az, aw]'\)
- Back to 3D by a central projection, through the origin, to the plane \(w = 1\)
  - \([x, y, z, w]' = [x/w, y/w, z/w, w/w]' \Rightarrow [x/w, y/w, z/w]'\)
  - a vector becomes a point at infinity
- These are called homogeneous coordinates

\[
\begin{bmatrix}
x_x & y_x & z_x & O_x \\
x_y & y_y & z_y & O_y \\
x_z & y_z & z_z & O_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

An example exam question

- Question:
  - Give the 2D homogeneous matrix that performs a 45 degree counterclockwise rotation around point \((1,2)\).
- Answer 1:
  - Hmm... rotation, I recall that... it's that matrix thing
  - The angle? I'll just use calculator...
  - \(\sin(45) = 0.707\)
  - Hey, \(\cos(45)\) is also 0.707!
  - Hmm... the point was \([1 2]'\)
  - I guess I'll just multiply that out
  - and so on...

\[
\begin{bmatrix}
0.707 & 0.707 & 1 \\
-0.707 & 0.707 & 2
\end{bmatrix}
\]

Better answer

- Rotation
  - but where's the negative sign?
  - 90 deg ccw rotation moves \((1,0)\) to \((0,1)\)
  - \(\cos(90) = 0, \sin(90) = 1\)
  - check: minus goes to upper right
  - angle?
    - \(\sin\) and \(\cos\) were \(x\) and \(y\) on the unit circle
    - at 45 deg \(x = y\) so \(\cos(45) = \sin(45)\)
    - \(\cos^2(45) + \sin^2(45) = 1\)
    - \(\Rightarrow 2 \cos^2(45) = 1 \Rightarrow \cos(45) = \sqrt{1/2}\)
  - homogeneous?
    - right, add that extra row and column
    - so translations and rotations can be multiplied together

\[
T = \begin{bmatrix}
1 & 0 & 0 & t_x \\
0 & 1 & 0 & t_y \\
0 & 0 & 1 & t_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
R = \begin{bmatrix}
x_x & y_x & z_x & 0 \\
x_y & y_y & z_y & 0 \\
x_z & y_z & z_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Translation

- first center to origin \(T(-1,-2)\)
- then rotate \(R(45)\)
- move center back \(T(1,2)\)

Solution

- a matrix (not point)
**Compounded transformation**

- Now as both transformations are multiplications, we can combine them:
  \[ q = (R \cdot (T \cdot (R \cdot p))) = (R \cdot T \cdot R) \cdot p = M \cdot p \]

- Which takes more operations? multiplying matrices first: 3*64+16 vs. 4*16
  - matrix * matrix = 64 muls
  - matrix * vector = 16 muls
  - typically you have tens, even thousands of points with the same transformations
  - better to precompute them then just 16 muls / vertex

**Inverse of rigid transformation**

Method 1: Inverse is also a rigid transformation, and applied after the transformation yields identity. Analyze what it must be.

\[ MM^{-1} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} A & b \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} RA & Rb + t \\ 0 & 1 \end{bmatrix} = I_{4 \times 4} \]

\[ RA = I \Rightarrow A = R' \]

\[ M^{-1} = \begin{bmatrix} R' & -R't \\ 0 & 1 \end{bmatrix} \]

Method 2: Transformation is rotation followed by translation, their inverses trivial, use rules of matrix algebra.

\[ M = TR \]

\[ M^{-1} = (TR)^{-1} = R^{-1}T^{-1} \]

\[ = \begin{bmatrix} R' & 0_{3 \times 1} \\ 0_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} I_{3 \times 3} & -t \\ 0_{1 \times 3} & 1 \end{bmatrix} = \begin{bmatrix} R' & -R't \\ 0_{1 \times 3} & 1 \end{bmatrix} \]

**Affine transformations**

- Any matrix with the last row [0 0 0 1] produces an affine transformation
- Two named transformations in addition to \( R \) and \( T \):
  - Scaling
    \[ S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]
  - Shearing
    \[ \begin{bmatrix} 1 & \bullet & \bullet & 0 \\ \bullet & 1 & \bullet & 0 \\ \bullet & \bullet & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & .5 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]
Application: Placing a camera

- Input: \( c, o, vup \)
  - \( vup = \) View UP vector

- Convention: camera looks into negative z

- Form a matrix for the camera
  - Translation part is \( c \)
  - Z-axis is the negative view direction.
  - In this case the vector from \( o \) to \( c \)
  - Y-axis is in the plane of \( z \) and \( vup \)
  - So \( x \)-axis is perpendicular: \( vup \times z \)
  - \( y \)-axis is perp to \( z \) and \( x \): \( z \times x \)
  - Note the order of cross products, follow the right hand rule
  - Don’t forget to normalize all the axes!

- Does this matrix map world to camera or the other way around?
  - Maps camera coordinates to world.
  - Try multiplying with \([0,0,0,1]\)’ get camera origin in world coordinates!

- How do we get world-camera xform?
  - Invert camera-world.

Modeling transformations in OpenGL

- \( \text{glMatrixMode}(\text{GL}_\text{MODELVIEW}) \)
- \( \text{glLoadIdentity()} \)
- \( \text{glTranslate}(tx, ty, tz) \)
- \( \text{glScale}(sx, sy, sz) \)
- \( \text{glRotate}(\text{ang}_\text{deg}, nx, ny, nz) \)

- Get / set:
  - \( \text{glLoadMatrixf}() \)
  - \( \text{mat} = \text{glGetDoublev}() \)
  - \( \text{mat} \) is a float[16], in column-major order (like Fortran)!
  - \( \text{glMultMatrixf}() \)
  - \( \text{glPushMatrix}(), \text{glPopMatrix}() \)
  - \( \text{gluLookAt}(cx, cy, cz, ox, oy, oz, vx, vy, vz) \)

Instances and transformations

- An instance of a house is transformed by an instance transformation
  - \( \text{glMatrixMode}() \)
  - \( \text{glLoadIdentity()} \)
  - \( \text{glTranslatef}(\ldots) \)
  - \( \text{glRotatef}(\ldots) \)
  - \( \text{glScalef}(\ldots) \)
  - \( \text{house}() \)

Global, fixed coordinate system

- OpenGL’s transforms, logical as they may be, still seem backwards
  - \( \text{glLoadIdentity}() \)
  - \( \text{glTranslatef}(\ldots) \)
  - \( \text{glRotatef}(\ldots) \)
  - \( \text{glScalef}(\ldots) \)
  - \( \text{house}() \)

- They are, if you think of them as transforming the object in a fixed coordinate system
Local, changing coordinate system

- Another way to view transformations is as affecting a local coordinate system that the primitive is drawn in.
- Now the transforms appear in the “right” order.

### 3D Example: A robot arm

- Consider this robot arm with 3 degrees of freedom:
  - Base (size: $h_{\text{base}}$) rotates about its vertical axis by $\theta$
  - Lower arm (size: $h_1$) rotates in its $xy$-plane by $\phi$
  - Upper arm (size: $h_2$) rotates in its $xy$-plane by $\psi$

#### Robot arm implementation

- The robot arm can be displayed by altering the model-view matrix incrementally:

```python
def robot_arm(theta, phi, psi):
    glRotatef( theta, 0.0, 1.0, 0.0 )  # rotate around y
    base()  # draw base
    glTranslatef( 0.0, h_base, 0.0 )  # translate along y
    glRotatef( phi, 0.0, 0.0, 1.0 )   # rotate around z
    lower_arm()  # draw lower arm
    glTranslatef( 0.0, h1, 0.0 )  # translate along y
    glRotatef( psi, 0.0, 0.0, 1.0 )   # rotate around z
    upper_arm()  # draw upper arm
```

#### Hierarchical modeling

- Hierarchical models can be composed of instances using trees or DAGs:

```
chassis
  └── right front wheel
  └── left front wheel
  └── right rear wheel
  └── left rear wheel

wheel
```
- edges contain geometric transformations
- nodes contain geometry
A complex example: human figure

- Q: What's the most sensible way to traverse this tree?

A: Depth-first. Then the transformations are inherited, when you go down push (store) the current matrix, when up pop (restore) it. That’s why breadth-first doesn’t make sense.

Matrix stacks

- \texttt{glMatrixMode(mode)}
  - \texttt{GL\_MODELVIEW}
  - \texttt{GL\_PROJECTION}
  - \texttt{GL\_TEXTURE}

- Matrix commands are post-multiplied to the current matrix
  - what’s issued last, is applied to the geometry first

- Save/restore current matrix
  \texttt{glPushMatrix()}
  \texttt{glPopMatrix()}

Human figure implementation

- The traversal can be implemented by saving the model-view matrix on a stack:

\begin{verbatim}
def figure:
  torso()
  glPushMatrix()
  glTranslate( ... )
  glRotate( ... )
  head()
  glPopMatrix()
  glPushMatrix()
  glTranslate( ... )
  glRotate( ... )
  left_upper_leg()
  glTranslate( ... )
  glRotate( ... )
  left_lower_leg()
  glPopMatrix()
  ...
\end{verbatim}
Scene graphs

• The idea of hierarchical modeling can be extended to an entire scene, encompassing:
  • many different objects
  • lights
  • camera position

• This is called a scene tree or scene graph

• More about scene graphs with VRML / X3D / M3G