Hierarchical Modeling
Plan

Hierarchical Modeling
OpenGL matrix stack
Scene graph
Forward kinematics
Inverse Kinematics
Interpolating rotations
Hierarchical Modeling

Triangles, parametric curves and surfaces are the building blocks from which more complex real-world objects are modeled.

Hierarchical modeling creates complex real-world objects by combining simple primitive shapes into more complex aggregate objects.
Hierarchical models
Hierarchical models
Hierarchical models
Hierarchical models
Hierarchical models
Hierarchical models
Examples

Curves and surfaces

- base
- lid
- ...
- spout

Animated Characters

- hips
- left-leg
- ...
- r-thigh
- r-calf
- r-foot
Hierarchical Grouping of Objects

Logical organization of scene
Simple Example with Groups

```
Group {
  numObjects 3
  Group {
    numObjects 3
    Box { <BOX PARAMS> } 
    Box { <BOX PARAMS> } 
    Box { <BOX PARAMS> } 
  }
  Group {
    numObjects 2
    Group {
      Box { <BOX PARAMS> } 
      Box { <BOX PARAMS> } 
      Box { <BOX PARAMS> } 
    }
    Group {
      Box { <BOX PARAMS> } 
      Sphere { <SPHERE PARAMS> } 
      Sphere { <SPHERE PARAMS> } 
    }
    Plane { <PLANE PARAMS> } 
  }
}
```
Adding Materials

Group {
    numObjects 3
    Material { <BLUE> }
    Group {
        numObjects 3
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
    }
    Group {
        numObjects 2
        Material { <BROWN> }
        Group {
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
        }
        Group {
            Material { <GREEN> }
            Box { <BOX PARAMS> }
            Material { <RED> }
            Sphere { <SPHERE PARAMS> }
            Material { <ORANGE> }
            Sphere { <SPHERE PARAMS> }
        }
    }
    Plane { <PLANE PARAMS> }
}
Adding Transformations
Hierarchical Transformation of Objects

Transforms position logical groupings of objects within the scene
Simple Example with Transforms

```xml
Group {
    numObjects 3
    Transform {
        ZRotate { 45 }
        Group {
            numObjects 3
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
        }
        Transform {
            Translate { -2 0 0 }
            Group {
                numObjects 2
                Group {
                    Box { <BOX PARAMS> }
                    Box { <BOX PARAMS> }
                    Box { <BOX PARAMS> }
                }
                Group {
                    Box { <BOX PARAMS> }
                    Sphere { <SPHERE PARAMS> }
                    Sphere { <SPHERE PARAMS> }
                }
            }
        }
    }
    Plane { <PLANE PARAMS> }
}
```
Separating types of transformation

Note that we have treated translations, rotations, etc. as separate
But they are all represented by 4x4 matrices and there is no technical reason not to combine them into the resulting matrix
It’s just simpler for the human programmer, and corresponds to the handle of 3D modeling/animation packages
Demo applet

http://www.cs.brown.edu/exploratories/...
Questions?
Plan

Hierarchical Modeling

*OpenGL matrix stack*

Scene graph

Forward kinematics

Inverse Kinematics

Interpolating rotations
Hierarchical modeling in OpenGL

Commands to change current transformation
- `glTranslate`, `glScale`, etc.

Affects the **state**, i.e. all following commands will undergo this transformation

Utilities to maintain a matrix stack (to revert to previous state)

Difference between model and view matrix
Model vs. Projection matrix

It is almost the same to rotate the camera or the objects

Main difference:
- Lighting

This is why OpenGL has two transforms: model and projection

```c
glMatrixMode( GL_MODELVIEW );
```

Tells OpenGL that next matrix commands deal with the objects
typically used for modeling & animation

```c
glMatrixMode(GL_PROJECTION);
```

Tells OpenGL we deal with the camera
typically used to change viewpoint & focal length
Managing the state

To reset everything: `glLoadIdentity();`

OpenGL stores a stack of matrices

- You don’t need to remember, OpenGL remembers
- `glPushMatrix()`
- `glPopMatrix`

- Typical use: push matrix when you start rendering a group
- Pop once you are done

- Geeky detail: the max stack size is typically bigger for...
  Modelview
Questions?
Plan

Hierarchical Modeling
OpenGL matrix stack
**Scene graph**
Forward kinematics
Inverse Kinematics
Interpolating rotations
Scene Graph

Convenient Data structure for scene representation

- Transformations
- Materials, color
- Multiple instances

Basic idea: Hierarchical Tree

Useful for manipulation/animation

- Especially for articulated figures

Useful for rendering too

- Ray tracing acceleration, occlusion culling
Scene Graph Representation

Basic idea: Tree
Comprised of several node types:

- Shape: 3D geometric objects
- Transform: Affect current transformation
- Property: Appearance, texture, etc.
- Group: Collection of subgraphs
Scene Graph Representations

In fact, generalization of a tree
Directed Acyclic Graph (DAG)
Why?

- Allows multiple instantiations
- Cycle forbidden
  - because infinite recursions
Traversal

Depth first

- Top to bottom, left to right
Traversal State

The State is updated during traversal

- Transformations, properties
- Influence of nodes can be complex
- E.g. bottom to top
Overview of Inventor

A suite of tools
- Viewer(s)
- Utilities

An API
- C++ set of classes for 3D display and manipulation

File format
- ASCII or binary
- Later became VRML
  - Virtual Reality Modeling Language
  - Internet 3D format

Originally SGI (Silicon Graphics)
- http://portal.acm.org/citation.cfm?id=166127

Now TGS
Questions?
Plan

Hierarchical Modeling
OpenGL matrix stack
Scene graph
Forward kinematics
Inverse Kinematics
Interpolating rotations
Animation

Hierarchical structure is essential for animation

- Eyes move with head
- Hands move with arms
- Feet move with legs
- ...

Without such structure the model falls apart:
Forward Kinematics

Describes the positions of the body parts as a function of the joint angles.

Each joint is characterized by its degrees of freedom (dof)

- Usually rotation for articulated bodies

1 DOF: knee

2 DOF: wrist

3 DOF: arm
Skeleton Hierarchy

Each bone transformation described relative to the parent in the hierarchy:

\[
x_h, y_h, z_h, q_h, f_h, s_h
\]

Derive world coordinates \( V_w \) for a point with local coordinates \( V_s \)?
Draw by Traversing a Tree

Assumes drawing procedures for thigh, calf, and foot use joint positions as the origin for a drawing coordinate frame.

```c
glLoadIdentity();
glPushMatrix();
glTranslatef(...);
glRotate(...);
drawHips();
glPushMatrix();
glTranslate(...);
glRotate(...);
drawThigh();
glTranslate(...);
glRotate(...);
drawCalf();
glTranslate(...);
glRotate(...);
drawFoot();
glPopMatrix();
```
Forward Kinematics

Transformation matrix for a point $v_s$ is a matrix composition of all joint transformation between the point and the root of the hierarchy.

Note that the natural parameters of the degrees of freedom (e.g. angle) have a non-linear effect.

$$v_w = T(x_h, y_h, z_h) R(q_h, f_h, s_h) \ TR(q_t, f_t, s_t) \ TR(q_c) \ TR(q_f, f_f) \ v_s$$

$$v_w = S\left(\begin{array}{c}
  x_h, y_h, z_h, \theta_h, \phi_h, \sigma_h, \theta_t, \phi_t, \sigma_t, \theta_c, \phi_c, \theta_f, \phi_f
\end{array}\right) v_s = S(p) v_s$$
Articulated Models

Articulated models:
- rigid parts
- connected by joints

They can be animated by specifying the joint angles as functions of time.
Procedural Animation

A stopwatch with second and minute hands. Hands rotate together as a function of time. The hands are animated by varying the time parameter.
Questions?
Plan

Hierarchical Modeling
OpenGL matrix stack
Scene graph
Forward kinematics
Inverse Kinematics
Interpolating rotations
Inverse Kinematics

Forward Kinematics

- Given the skeleton parameters (position of the root and the joint angles) $p$ and the position of the point in local coordinates $v_s$, what is the position of the point in the world coordinates $v_w$?
- Not too hard, we can solve it by evaluating $S(p)v_s$

Inverse Kinematics

- Given the position of the point in local coordinates $v_s$ and the desired position $\tilde{v}_w$ in world coordinates, what are the skeleton parameters $p$?
- Much harder requires solving the inverse of the non-linear function:
- Underdetermined problem with many solutions $p$ such that $S(p)v_s = \tilde{v}_w$
It’s underconstrained

e.g., count degrees of freedom:
- we specify one point (3 equations)
- We usually need more than 3 angles

Simples geometric example (in 3D):
specify hand position, need elbow & shoulder
- The set of possible elbow location is a circle in 3D
How to tackle these problems?

Deal with non-linearity: Iterative solution (steepest descent)
- Compute Jacobian of position wrt angles
- Good news: in an interactive contest, we want to move e.g. the hand and we can use the previous frame as an initial guess

Deal with ill-posedness: Pseudo-inverse (18.06!)
- Solution that displaces things the least

Deal with ill-posedness: Prior on “good pose”
- More advanced

Additional potential issues: bounds on joint angles, etc.
Questions?
Plan

Hierarchical Modeling
OpenGL matrix stack
Scene graph
Forward kinematics
Inverse Kinematics

Interpolating rotations
Interpolating Orientations in 3-D

Critical in particular for camera animation

Rotation matrices

Given rotation matrices $M_i$ and time $t_i$, find $M(t)$ such that $M(t_i) = M_i$
Flawed Solution

Interpolate each entry independently

Example: $M_0$ is identity and $M_1$ is $90^\circ$ around $x$-axis

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & -1 & 0
\end{bmatrix}
= 
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0.5 & 0.5 \\
0 & -0.5 & 0.5
\end{bmatrix}
\]

Is the result a rotation matrix?
No, it does not preserve rigidity
(angles and lengths)
3D Rotations

How many degrees of freedom for 3D orientations?

3 degrees of freedom:
- direction of rotation and angle
- or 3 Euler Angles
Euler Angles

An Euler angle is a rotation about a single axis.
Any orientation can be described by composing three rotations, one around each coordinate axis.
Roll, pitch and yaw (perfect for flight simulation)

http://www.fho-emden.de/~hoffmann/gimbal09082002.pdf
Interpolating Euler Angles

Natural orientation representation: interpolate independently 3 angles for 3 degrees of freedom. However, leads to unnatural interpolation:

Rotation of 90° around Z, then 90° around Y = 120° around (1, 1, 1)

But rotation of 30° around Z then 30° around Y ≠ 40° around (1, 1, 1)
Gimbal Lock

Two or more axis align resulting in a loss of rotation degrees of freedom.

http://www.fho-emden.de/~hoffmann/gimbal09082002.pdf

Many slides by Jovan Popovic and Barbar Cutler  6.837 Fredo Durand MIT EECS
Euler Angles in the Real World

Apollo inertial measurement unit
To “prevent” lock, they added a fourth Gimbal!

http://www.hq.nasa.gov/office/pao/History/alsj/gimbals.html

Figure 2, 1-24. IMU Gimbal Assembly
Questions?
Solution: Quaternion Interpolation

Interpolate orientation on the unit sphere in 4D

- Logical and easy, isn’t it?

By analogy:
1-, 2-, 3-DOF rotations as constrained points on 1, 2, 3-spheres in 2D, 3D and 4D
1D Sphere and Complex Plane

1 angle
- But messy to handle because modulo $2\pi$

Solution:
- Represent rotation by point on circle
- Use interpolation in 2D plane
- Project back to circle

And we can say that the 2D plane is the complex plane
- Orientation = complex argument of the number
- Not strictly necessary, but extends more easily to 3D rotations

- Interestingly, composition of rotation $\Leftrightarrow$ complex multiplication
  - Trivial with exponential notation $re^{i\theta}$
Velocity Issue: lerp vs. slerp

Linear Interpolation (lerp) interpolates the straight line between the two orientations
→ lerp motion does not have uniform velocity:

\[
\text{lerp}(q_0, q_1, t) = q(t) = q_0 (1 - t) + q_1 t
\]

Spherical Linear Interpolation (slerp) interpolates along the arc lines by adding a sine term:

\[
\text{slerp}(q_0, q_1, t) = q(t) = \frac{q_0 \sin((1 - t)\omega) + q_1 \sin(t\omega)}{\sin(\omega)},
\]

interpolate along arc line rather than secant