

Department of Computer Science **Hierarchical Models**

- Objects are defined in their own coordinate system
- They are then “transformed” and placed in their proper place in the scene
- Transformations:
  - Translation (moving from one spot to another)
  - Rotating (around any of the axis)
  - Scaling (making the object larger or smaller in any direction)

5/3/2007 1

Department of Computer Science **The Graphics Pipeline**

$$\begin{bmatrix} x_v \\ y_v \\ z_v \\ 1 \end{bmatrix} = [Projection] \cdot \begin{bmatrix} World \\ to \\ Camera \end{bmatrix} \cdot [Transformation] \cdot \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

view plane local object coords

5/3/2007 2

Department of Computer Science **Transformations in OpenGL**

- OpenGL maintains 3 kinds of matrices:
  - *Modelview*
    - Handles all object transformations
    - Handles world to camera transformation
  - *Projection*
    - Handles projection
    - Equivalent to the camera view plane
  - *Viewport*
    - Handles view plane to view port (2D → 2D)
- All matrix operations in OpenGL are performed on the “current” matrix as defined by the OpenGL *matrix mode*
- `glMatrixMode()` is used to select the mode

5/3/2007 3

Department of Computer Science **Matrix Stacks**

- In OpenGL, “current” matrices are really matrix stacks
- Before adding a localized transformation, push the current transformation matrix on the stack, thus saving a copy of it.
 

```
glPushMatrix()
```
- When you’re done, pop the stack, thus restoring the previous transformation matrix.
 

```
glPopMatrix()
```

5/3/2007 4

Department of Computer Science **Hierarchical Models**

- Suppose we need to model an object comprised of components all placed relative to other components
  - Example:
    - A two-armed robot
    - `/usr/local/pub/ncs/graphics/OpenGL/ExamplesLab3/robot`
- Even in a local coordinate system, it may be tedious to keep track of the transformations of the various components

5/3/2007 5

Department of Computer Science **Hierarchical Models**

- Example: a two-armed robot

```

graph TD
    Base[Robot Base] --> LA1[Lower Arm]
    Base --> LA2[Lower Arm]
    LA1 --> UA1[Upper Arm]
    LA1 --> UA2[Upper Arm]
    LA2 --> UA3[Upper Arm]
    LA2 --> UA4[Upper Arm]
    
```

5/3/2007 6

**Hierarchical Models**

- We know how to transform of each component with respect to another component
  - Use the matrix stack in order to calculate the local coordinates of each component

5/3/2007 7

**Hierarchical Models - Robot**

```

Define camera orientation
Push Matrix // Saves view matrix
Define transformation for robot as a whole
Push Matrix // Saves whole object position/orientation matrix
Define transformations for positioning robot base in scene
Draw robot base
Push Matrix // Saves robot base
Define transformations for left arm w/r/t the center of the robot
Draw arm
Pop Matrix // Restores robot base matrix
Push Matrix // Saves robot base
Define Transformations of right Arm with respect to robot base
Draw arm
Pop Matrix // Restores robot base
Pop Matrix // Restores position/orientation matrix
Pop Matrix // Restores pre-robot (i.e., orig view) matrix
    
```

5/3/2007 8

**Hierarchical Models - drawArm**

```

Define transformations for robot lower arm w/r/t the center of the robot
Push matrix
Define transformations needed to generate lower arm from glutCube
Draw lower arm
Pop matrix
Define transformations of upper arm w/r/t the lower arm
Push matrix
Define transformations needed to generate upper arm from glutCube
Draw upper arm
Pop matrix
    
```

5/3/2007 9

**Hierarchical Models - drawBase**

```

Push Matrix
Define transformations to generate base from gluCylinder
Draw Base
Pop Matrix
    
```

5/3/2007 10

**Hierarchical Models**

- Advantages
  - More intuitive to specify
  - Allows for individual control of each component
  - Reuse

5/3/2007 11

**OpenGL Matrix Stack Example**

```

glMatrixMode( GL_MODELVIEW )
glLoadIdentity();
gluLookAt( 2.0, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0 1.0, 0.0 );

glPushMatrix();
glTranslatef( 1.0, 2.0, 3.0 );
glScalef( 2.0, 2.0, 2.0 );
glRotatef( 45.0, 0.0, 0.0, 1.0 );
glRotatef( 30.0, 0.0, 1.0, 0.0 );
// code to draw object1
glPopMatrix();

glPushMatrix();
glTranslatef( 10.0, 10.0, 10.0 );
// code to draw object2
glPopMatrix();
    
```


5/3/2007 12

 **So you'd like to use multiple models...**

- Display lists: a group of OpenGL commands that have been stored for later execution
- Some example code we could make into a list:
 

```
glColor3f( 1, 1, 1 );
glutSolidTeapot(0.25);
glPushMatrix();
glColor3f( 1, 0, 0 );
glTranslatef(-1.0, -0.5, 0.0);
glutSolidTeapot(0.25);
glPopMatrix();
glPushAttrib(GL_ALL_ATTRIB_BITS);
glTranslatef(0.0, -0.5, 0.0);
glColor3f( 0, 0, 1 );
glutSolidTeapot(0.25);
glPopAttrib();
```


5/3/2007 13

 **So you'd like to use multiple models...**

```
glNewList(TEAPOTS, GL_COMPILE);
glColor3f( 1, 1, 1 );
glutSolidTeapot(0.25);
glPushMatrix();
glColor3f( 1, 0, 0 );
glTranslatef(-1.0, -0.5, 0.0);
glutSolidTeapot(0.25);
glPopMatrix();
glPushAttrib(GL_ALL_ATTRIB_BITS);
glTranslatef(0.0, -0.5, 0.0);
glColor3f( 0, 0, 1 );
glutSolidTeapot(0.25);
glPopAttrib();
glEndList();
```


- Can call list with: `glCallList(TEAPOTS)`

5/3/2007 14

 **Other Helpful Commands**


- GLuint **glGenLists** (GLsizei range) : will give you a new range of ID's that haven't been used
- GLboolean **glIsList** (GLuint listID) : tells you if this list index is used
- GLvoid **glDeleteLists** (GLuint firstlist, GLsizei range) : deletes a range of list indices starting with firstlist
- GLvoid **glCallLists** (GLsizei n, GLenum type, GLvoid \*lists\_indices) : used to call several lists in a row (good for lists of lists)

5/3/2007 15

 **Animation**

- Motion and changes over time
- Create individual images (frames) to be played back at a constant rate
  - 24 frames / sec = film
  - 30 frames / sec = video and TV
  - As fast as your machine can handle = interactive animation.
- Each image reflects changes in the scene that occurs within each time frame
- Examples:
  - Early Computer Animation: [Wally B](#)
  - Artistic Breakthrough: [Luxo Jr.](#)
  - Newer: The Ark Film


5/3/2007 16

 **Animation**

- What can change in an animation:
  - Shape of objects
  - Position of objects
  - Transformation on objects and object components
  - Lighting
  - Camera
  - Colors
  - Textures
  - Etc.
- "There's no particular mystery in animation... It's really very simple and like anything that is simple, It's about the hardest thing in the world to do"

-- Bill Tytla, Walt Disney Studios, 1937

5/3/2007 17

 **Why Animation is Difficult**

- Consider a 5 minute feature
- 5 min x 60 sec x 24 frames = 7200 frames
- For each frame of 7200, you must determine the "state" of your scene
- You also have to generate an image for each frame
- Now consider a 90-minute feature film
- 90 min x 60 sec x 24 frames = 129,600 frames!

5/3/2007 18

Department of Computer Science **Animation Pipeline**

```

    graph TD
      Storyboard[Storyboard] --> Modeling[Modeling]
      Storyboard --> MotionControl[Motion Control]
      Storyboard --> Rendering[Rendering]
      Storyboard --> PostProduction[Post-Production]
      Modeling --> MotionControl
      MotionControl --> Rendering
  
```

5/3/2007 19

Department of Computer Science **Animation Pipeline**

- Storyboard
  - Series of rough cartoon-like drawings
  - Depicts staging, camera angles, pacing, division into scenes
  - One storyboard for each major piece of action

```

    graph TD
      Storyboard[Storyboard] --> Modeling[Modeling]
      Storyboard --> MotionControl[Motion Control]
      Storyboard --> Rendering[Rendering]
      Storyboard --> PostProduction[Post-Production]
      Modeling --> MotionControl
      MotionControl --> Rendering
  
```

5/3/2007 20

Department of Computer Science **Animation Pipeline**

- Modeling
  - Creating the objects in your scene
    - Geometry
    - Shading
    - Texturing

```

    graph TD
      Storyboard[Storyboard] --> Modeling[Modeling]
      Storyboard --> MotionControl[Motion Control]
      Storyboard --> Rendering[Rendering]
      Storyboard --> PostProduction[Post-Production]
      Modeling --> MotionControl
      MotionControl --> Rendering
  
```

5/3/2007 21

Department of Computer Science **Animation Pipeline**

- Motion control
  - Defining the motion of objects over time
  - Defining other changes in your scene over time
    - Camera motion
    - Lighting
    - Texturing, etc.

```

    graph TD
      Storyboard[Storyboard] --> Modeling[Modeling]
      Storyboard --> MotionControl[Motion Control]
      Storyboard --> Rendering[Rendering]
      Storyboard --> PostProduction[Post-Production]
      Modeling --> MotionControl
      MotionControl --> Rendering
  
```

5/3/2007 22

Department of Computer Science **Animation Pipeline**

- Rendering
  - Create an image for each frame in your animation
    - Local Illumination Models
    - Radiosity
    - Ray Tracing
    - Other rendering methods

```

    graph TD
      Storyboard[Storyboard] --> Modeling[Modeling]
      Storyboard --> MotionControl[Motion Control]
      Storyboard --> Rendering[Rendering]
      Storyboard --> PostProduction[Post-Production]
      Modeling --> MotionControl
      MotionControl --> Rendering
  
```

5/3/2007 23


Department of Computer Science **Animation Pipeline**

- Post-production
  - Editing
  - Add sound, effects, narration, title
  - Assemble onto a given medium (film, video, etc.)

```


    graph TD
      Storyboard[Storyboard] --> Modeling[Modeling]
      Storyboard --> MotionControl[Motion Control]
      Storyboard --> Rendering[Rendering]
      Storyboard --> PostProduction[Post-Production]
      Modeling --> MotionControl
      MotionControl --> Rendering
  
```

5/3/2007 24

 **Motion Control**


- A problem unique to animation
- How to define position and orientation for each object in scene in frame of the animation?
- Issues:
  - How much control?
  - How many degrees of freedom?

5/3/2007 25

 **Motion Control Techniques**


- Keyframing / Tweening / Interpolation
  - Define location and orientation at a number of given *keyframes*
  - System *interpolates* to get motion for frames between the keys
- Motion is defined by animator
- Method most often used in “traditional” animation

5/3/2007 26

 **Motion Control Techniques**


- Procedural methods
- Motion is generated by some procedure or algorithm
  - Dynamic (physically based)
  - Heuristics
  - Behavioral motion

5/3/2007 27

 **Dynamic (Physically-Based) Motion Control**


- Objects have physical attributes (weight, mass, etc)
- Motion is determined by laws of physics
  - Rigid body motion
  - Shape deformation
  - Cloth modeling
- Need a physics simulator
- Animator has little control

5/3/2007 28

 **Heuristic Motion Control**


- Like dynamic motion control, except that the physics is made up
  - What looks good vs. what is physically accurate
- Take liberties with physics
- Examples:
  - Particle Systems
  - Smoke, Fog, Fire

5/3/2007 29

 **Behavioral Motion Control**


- Motion defined by some set of heuristics
- Good for modeling motion of large groups of objects
  - Ex: flocking, herding, plant growth
  - Ex: Lord of the Rings battles
- Motion of one object is based on relationship to other objects in the group
- Once again: animator has little control

5/3/2007 30


**Articulated Character Motion**


- Defining motion for articulated figures (hierarchical models) is particularly challenging
  - Must define transformations for all joints in the hierarchy
- *Forward kinematics*
  - Animator defines transformation for all joints
  - End-effectors (fingers, feet, toes) end up where they may
  - Difficult to do even for the best animators
- *Inverse kinematics*
  - Animator defines position of end effectors
  - System determines "appropriate" transformations at each joint

5/3/2007 31


**Articulated Character Motion**


- Levels of control
  - *Low-level*
    - Animator defines all joint transformations
  - *Procedural*
    - Animator defines "motor programs"
      - Swing leg, wiggle finger
  - *Functional*
    - Animator defines "skill"
      - Walking, grasping
  - *Character*
    - Animator defines a "task"
      - Go to the fridge and get me a beer

5/3/2007 32

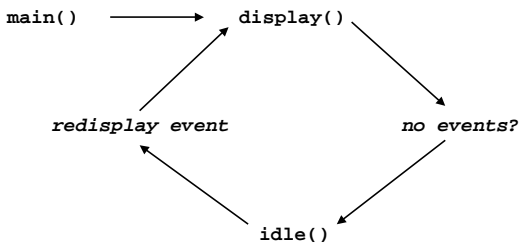

**Real-time Animation**

- Frames are generated on the fly
- Frame rate will depend upon
  - Speed of your machine
  - Complexity of your scene
  - Complexity of your rendering

5/3/2007 33


**Animation in OpenGL**


- OpenGL (GLUT) event loop



```


    graph TD
      main[main()] --> display[display()]
      display --> no_events[no events?]
      no_events --> idle[idle()]
      idle --> redisplay[redisplay event]
      redisplay --> display
    
```

5/3/2007 34


**Animation in OpenGL**

- Perform scene updates in `idle()` callback
- OpenGL also supports *double buffering*
  - Two frame buffers
    - Last frame shown in window
    - New frame being drawn on second buffer
    - Swap buffers

5/3/2007 35


**Areas of Animation Research**

- Procedural Animation
  - Fog, smoke, water, etc.
- Facial Animation
  - Facial models, lip sync
- Sound and Animation
  - Automated soundtracks / speech
- Real-time animation
  - Gaming
- Shape Morphing
- Motion Capture Systems

5/3/2007 36