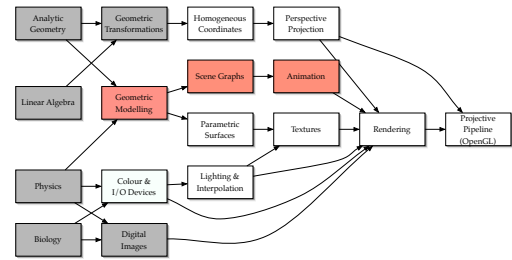


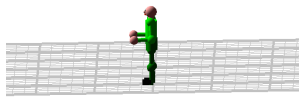
Animation

COMP 3003 Autumn 2005

Where We Are



Animation Example:



What is Animation?

- The *illusion* of motion:
 - making things appear to move
 - by drawing things too fast for the eye
- So we go back to what the eye can do

The Illusion of Motion

- The eye sees events in about $1/24$ sec.
- Anything shorter, the eye mostly misses
- So, we show more than 24 *frames* / sec.
 - the eye *interprets* this as motion
 - but each image is static
- In practice, it's more complex

Movies

- Movies used to run at 16 *frames per second*
 - so they looked jerky
- Modern movies run at 24 fps or better
- Each frame is a single image
 - computed "off-line" & stored on film
 - projected one at a time at fixed speed

Computers

- Generate images on the fly
 - it takes most of the time just to draw
 - most of the 1/24 sec you see nothing
 - you see some objects longer than others
 - last objects drawn look ghostly

Tearing

- Video output refreshes constantly
 - doesn't wait for frame to finish
 - so you get half a frame visible
- CRT's scan from top down
 - so you get a similar problem
 - we'll come back to this later

Double-buffering

- We can get rid of the first problem
 - Use **TWO** copies of the framebuffer
 - Draw one, show the other
 - When finished drawing, swap them
 - `glutSwapBuffers()`

Dropped Frames

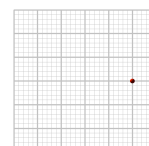
- Show a new frame every 1/30 second
- What if we are late?
 - the eye sees the same frame twice
 - motion no longer looks smooth
- Solution: render faster & we won't notice
- Result: we want 60 fps or better

Simulating Motion

- For each frame, we compute:
 - new transformations for each object
 - new lighting (sometimes)
 - new colour / textures (occasionally)
 - new geometry (rarely)

Time & Motion

- We treat *position* p as a function of *time*
- We *compute* p given a particular t
- Usually in parametric form



Simple Example

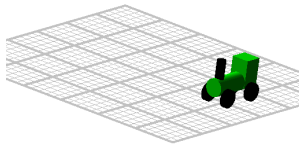
```
void animateTrain()
{ /* animateTrain() */
  for (timeStep = 0; timeStep < 600; timeStep++)
  { /* each frame */
    computePosition(timeStep);
    drawTrain();
  } /* each frame */
} /* animateTrain() */
```

Computing Position

- Given the *time step*, compute position:

```
void computePosition(int time)
{ /* computePosition() */
  theta = (time / 600) * 2 * PI;
  posn_x = cos(theta);
  posn_y = sin(theta);
  glTranslatef(posn_x, posn_y, 0.0);
} /* computePosition() */
```

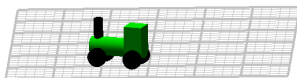
Oops ...



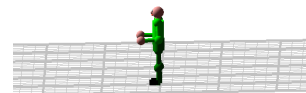
Position & Orientation

```
void computePosition(int time)
{ /* computePosition() */
  thetaDeg = (time / 600) * 360;
  thetaRad = (time / 600) * 2 * PI;
  posn_x = cos(thetaRad);
  posn_y = sin(thetaRad);
  glRotatef(thetaDeg, 0.0, 0.0, 1.0);
  glTranslatef(posn_x, posn_y, 0.0);
} /* computePosition() */
```

And the result



A Complex Example



Articulation

- Humans are different from toy trains
 - they are *articulated*
 - composed of *bones* (i.e. limbs)
 - connected by *joints*
 - so we will model them this way

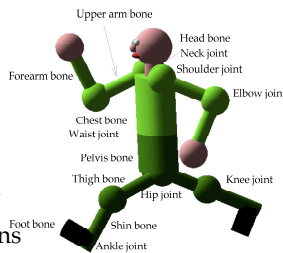
Bones

- How many bones do we have?
 - 208 (I think)
 - but many don't matter
- A *bone* is a separately transformed piece of the object - often a limb
- Even trains have *bones* - wheels, &c.



How many Bones?

- Number of bones depends on detail
- Sometimes need fingers, toes, &c.
- Even need "bones" for facial expressions



Bone Coordinates

- Each bone has its own coordinates
 - i.e. *local coordinate systems*
- But we know how to deal with these
- Bones are connected by *joints*
 - points where two bones meet

Joints

- *Joints* are points at which bones meet
 - one bone rotates around the joint
- Joint is fixed in stationary bone's coords
- Joint is *origin* of other bone's coords
 - to move the bone, rotate around joint
 - "simple" rotation matrix

Degrees of Freedom

- Not all rotations are possible at joints:
- *degrees of freedom* = number of rotations
- Elbow & knee: 1 *d.o.f.*, 135° range
- Shoulder / hip:
- Wrist / ankle:
- Waist / neck:

Direction of Axes

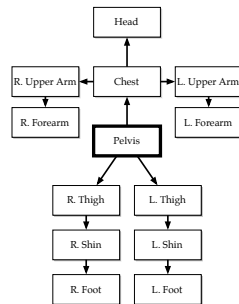
- Axes for bone are up to you
 - convenient if rotations are simple
 - e.g. elbow y-axis sticks out sideways
 - so all elbow rotations are y-rotations
- I like putting z-axis down centre of bone

Which Bone Rotates?

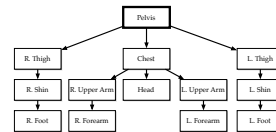
- Both, unless there is another force acting
- Ultimately, motion pushes against ground
- But the feet move a lot!
 - Human motion keeps body / head stable
 - So pelvis is treated as stationary
 - if it moves or rotates, the body does too

Bone Diagram

- Pelvis is master bone
 - all others hang off it
- So let's redraw this diagram:



Bone Hierarchy



- This hierarchy makes drawing easier
- Let's look at drawing the left hand

Drawing Left Hand

- Left hand is part of forearm
 - defined in forearm's local coordinates
 - transform to upper arm's LCS
 - then transform to chest's LCS
 - then to pelvis' LCS
 - finally to WCS

Left Hand Transformations

$$T_{pelvis} T_{chest} T_{left\ arm} T_{left\ forearm} (hand)$$

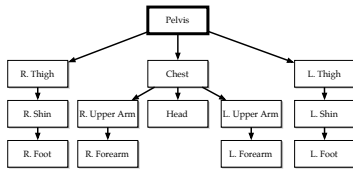
- Transformations inherited from hierarchy
 - we call them *hierarchical transformations*
 - diagram is the *transformation hierarchy*
- Right hand shares some of these

$$T_{pelvis} T_{chest} T_{right\ arm} T_{right\ forearm} (hand)$$

Drawing the Man

```

Apply T_pelvis
Draw Pelvis
Apply T_chest
Draw Chest
Apply T_L_arm
Draw L Upper Arm
Apply T_L_forearm
Draw L Forearm
Remove T_L_forearm
Remove T_L_arm
Apply T_head
Draw head
Remove T_head
Apply T_R_arm
Draw R Upper Arm
    
```



Traverse the hierarchy:
going down, apply matrix
going up, remove it

Matrix Stacks

- Since this is Computer Science
- we can represent this with *stacks*
 - *push* another matrix going down
 - *pop* a matrix going up
- and OpenGL has this built in

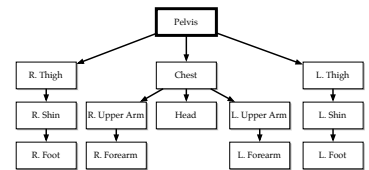
glPushMatrix()

- glPushMatrix() pushes current matrix
 - and gives you a copy to work with
- glPopMatrix() gets rid of copy
 - and reverts to last version on stack
- Like glBegin(), OpenGL doesn't balance them
 - make sure you do

Drawing the Man

```

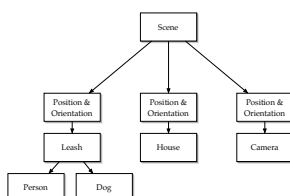
glPushMatrix()
Apply T_pelvis
Draw Pelvis
glPushMatrix()
Apply T_chest
Draw Chest
glPushMatrix()
Apply T_L_arm
Draw L Upper Arm
glPushMatrix()
Apply T_L_forearm
Draw L Forearm
glPopMatrix()
Apply T_head
Draw head
glPopMatrix(), &c. &c.
    
```



Traverse the hierarchy:
going down, apply matrix
going up, remove it

Scene Graphs

- General form of animation hierarchy
- Hierarchical description of a *scene*



Terminology

- A *frame* is a single image in a sequence
- A *bone* is a piece of an articulated object
- A *joint* is where two bones meet
- A *pose* is the set of rotations for a frame

Generating Poses

- How *do* we determine the poses?
 - motion capture
 - keyframe interpolation
 - physical simulation
 - inverse kinematics

Motion Capture

- Measure the joint angles of a person
 - then build a model of the person
- Commonly used in
 - movies (Gollum)
 - sports games (Michael Jordan)
- The ideal solution ?

Mocap Problems

- Motion only works for one character
 - looks wrong on smaller / larger
- Hard to measure joint angles
 - surface occlusion, markers off-centre
- Requires expensive hardware & people
- Only works for real animals / humans

Keyframe Interpolation

- Some frames are important (i.e. *key*)
 - artist specifies joint angles at keyframes
 - compute angles for frames in between
 - simple interpolation
- Requires well-trained artists

Physics Simulation

- Simulate Newtonian mechanics
- Much more complex than it sounds
 - and doesn't give humans control
- Best for inanimate objects
 - we must detect collisions
 - even for human motions

Inverse Kinematics

- Artist sketches approximate path
- We compute suitable joint angles
 - use physics simulation
 - try to minimize energy used
 - human motion is efficient
- Lots of work to be done here

Animation Problems

- Collision detection & response
- Smooth interpolation (*blending*) of poses
- Smooth *skinning* between joints
- Mapping to different characters
- Controls for artists