Robotic Animals
New Concepts

• 3 Dimensional World
• Full Screen Animation
• Multiple Frames of Reference
• Camera vs Object Motion
• Lighting
• Complex Hierarchical Drawing/Animation
3 Dimensional World

- 3 parameters \((x,y,z)\) – glVertex3f(…)
- Using perspective projection instead of orthographic - gluPerspective(…)
- Objects can occlude (block visibility) one another, we need a depth buffer to figure out when this is happening - glutInitDisplayMode(…) with GLUT_DEPTH, glEnable(GL_DEPTH_TEST), glClear(..) with GL_DEPTH_BUFFER_BIT
Full Screen Animation

• In the painting assignment we only drew one picture. This program should draw new pictures over and over as fast as it can.
• All painting goes in the display callback
• Add an idle callback that tells GLUT to git drawin' again - glutPostRedisplay()
• We don’t want the user to watch us drawing so we draw to a back buffer while the user looks at front, then swap them when we are ready to show them the new picture - glutSwapBuffers() -- implicitly flushes. Look for GLUT_DOUBLE for glutInitDisplayMode()
• We want to take up the whole screen -- use glutFullScreen() after glutCreateWindow(…)

•
Multiple Frames of Reference

- The arm rotates about the shoulder
- The forearm rotates about the elbow
- The hand rotates about the wrist
- OpenGL does all of the hard math for you as long as you tell it what frame you want to work in - glTranslate, glRotate, glPushMatrix, glPopMatrix, glScale…
- Ideally, you can do the whole assignment without sin() and cos() (you might use these for some special effects though)
- We want to update the states using some constants for the rate of each rotation and the amount of time that has passed before we draw the scene each time.
  \[ x += v \times dt \]
Camera vs Object Motion

- OpenGL doesn’t make a distinction here, but you should (to avoid getting lost)
- Scene is rendered from eye’s frame of reference
- Simulation is run from the body’s frame of reference
- Use gluLookAt(…) or other functions to move the simulated world’s origin with respect to the eye before drawing the world. This has same logical effect as moving the eye within the world.
Camera Setup

• In your reshape callback reset the projection matrix to be a *perspective* projection with the appropriate aspect ratio, nothing more. (45 degrees is a decent field of view)

• In your display callback, the first change to the modelview matrix should be the camera’s transformation (with *gluLookAt*), after that everything you draw will be relative to the shifted origin so it looks correct from the eye point. Save the modelview matrix anytime you think you’ll need its state back later
Lighting  (magic for now)

- Turn on lighting using glEnable() with GL_LIGHTING and GL_LIGHT0
- Specify material properties with glMaterial (how much light an object reflects)
- Specify light properties with glLight (how much light is emitted and from where)
- Enable GL_NORMALIZE -- fixes normal vectors on geometry from glut
- Enable GL_COLOR_MATERIAL -- makes glColor work like painting assignment
Complex Hierarchical Drawing Using Current State Variables

```cpp
glPushMatrix();
    drawTower(tower_height);
    glTranslate(0,tower_height,0);
    glRotate(arm_angle,1,0,0);
    drawControlBox();
    drawArm(arm_length);
    glTranslate(0,0,arm_length);
    glPushMatrix();
        glRotate(pulley_angle,1,0,0);
        drawPulley();
    glPopMatrix();
    glTranslate(-cable_length,0,0);
    drawBarrel();
glPopMatrix();

glPushMatrix();
    glTranslate(first_barrel,0,0);
    for(l=0; l<4; l++) {
        drawBarrel();
        glTranslate(barrel_spacing,0,0);
    }
glPopMatrix();
```
“I still don’t really feel like I understand OpenGL in general”

- At other schools there are whole classes on OpenGL or at least a few lecture sessions about it.
- cmps160 is more concept-based, you only learn to program with OpenGL in the lab.
- I learned OpenGL myself from examples and documentation. However, if you learn best through colorful PowerPoint™ presentations, here is a good starter covering everything up through this assignment: http://www.cs.virginia.edu/~gfx/Courses/2004/Intro.Spring.04/Lectures/lecture04.ppt
Requirements

• Display a 3D scene with at least two camera angles
• Draw an animal with the following parts:
  – Body
  – Head
  – 4 Legs, each with multiple segments (thigh, calf)
  – 1 Tail with multiple segments so it can curl (10)
• Make your animal walk in a circle, jump, or stand on two legs like a horse
• The movement must be fairly complex and have parts that move at different speeds
Implementation Suggestion

- Familiarize yourself with drawing shapes and drawing them together
- Plan out (on paper) what your animal will look like
- Make a few small functions (initSimulation(), updateSimulation(), drawBody(...), drawLeg(...), placeLight(), etc.)
- You may need to “fix” the example code (i.e. the camera and/or viewing frustum)
- Draw a simplistic form of your animal
- Add simulation code
- Add more complexities and controls
- For complex animations a reverseRotation() function or a rotationDirection variable might be useful
“Simulation code”

**CONSTANTS**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>armSpinRate</td>
<td>10</td>
</tr>
<tr>
<td>elbowSpinRate</td>
<td>35</td>
</tr>
</tbody>
</table>

(all in degrees per second)

**VARIABLES (and initialization)**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>armSpin</td>
<td>0</td>
</tr>
<tr>
<td>elbowSpin</td>
<td>0</td>
</tr>
<tr>
<td>lastTicks</td>
<td>getTicks()</td>
</tr>
</tbody>
</table>

**UPDATE (per-frame)**

```plaintext
nowTicks = getTicks()
dt = (nowTicks - lastTicks)/1000.0
armSpin += armSpinRate*dt
elbowSpin += elbowSpinRate*dt
lastTicks = nowTicks
```

```plaintext
dt = (nowTicks - lastTicks)/1000.0
armSpin += armSpinRate*dt
elbowSpin += elbowSpinRate*dt
lastTicks = nowTicks
```
Primitives

- `glutSolidSphere()`
- `glutSolidCube()`
- `glutSolidCone()`
- `gluCylinder()` – needs a quadric made with `gluNewQuadric()`
- `gluDisk()`
- `glVertex3f()`

- You might also want to look at `glScale()` for getting shapes to be the right size
Graphical Extras

Ground
A really big quad

Water
A really big semi-transparent quad

Starfield
for 100 stars {
    theta = rand in 0-2*pi
    phi = rand in 0-2*pi
    Push matrix
    Rotate theta on x
    Rotate phi on y
    Translate star distance on z
    Draw point (with color based on sin(time) ?)
    Pop matrix
}
Resources

- **OpenGL Programming Guide** (The Red Book)
  http://www.glprogramming.com/red/
- **Viewing**
  http://www.glprogramming.com/red/chapter03.html
- **Lighting**
  http://www.glprogramming.com/red/chapter06.html
- **PyOpenGL Man Pages** (Great OpenGL API doc)
  http://pyopengl.sourceforge.net/documentation/manual/index.xml
- **NeHe OpenGL Tutorials**
  http://nehe.gamedev.net/