Lighting
- computing brightness based on property of material and light position(s)
- computation is performed per-vertex

Geometry Database
- application-specific data structure for holding geometric information
- depends on specific needs of application
- triangle soup, points, mesh with connectivity information, curved surface

Model/View Transformation
- modeling transformation
- map all geometric objects from local coordinate system into world coordinates
- viewing transformation
- map all geometry from world coordinates into camera coordinates

Perspective Transformation
- perspective transformation
- projecting the geometry onto the image plane
- projective transformations and model/view transformations can all be expressed with 4x4 matrix operations

Scan Conversion
- scan conversion
- turn 2D drawing primitives (lines, polygons etc.) into individual pixels (discretizing/sampling)
- interpolate color across primitive
- generate discrete fragments
Texture Mapping
• texture mapping
• “gluing images onto geometry”
• color of every fragment is altered by looking up a new color value from an image

Depth Test
• depth test
• remove parts of geometry hidden behind other geometric objects
• perform on every individual fragment
• other approaches (later)

Blending
• blending
• final image: write fragments to pixels
• draw from farthest to nearest
• no blending – replace previous color
• blending: combine new & old values with arithmetic operations

Framebuffer
• framebuffer
• video memory on graphics board that holds image
• double-buffering: two separate buffers
• draw into one while displaying other, then swap to avoid flicker

Texture Mapping
• texture mapping
• “gluing images onto geometry”
• color of every fragment is altered by looking up a new color value from an image

Geometry Database
Model/ViewTransform.
Lighting
PerspectiveTransform.
Clipping
ScanConversion
Texturing

Pipeline Advantages
• modularity: logical separation of different components
• easy to parallelize
• earlier stages can already work on new data while later stages still work with previous data
• similar to pipelining in modern CPUs
• but much more aggressive parallelization possible (special purpose hardware!)
• important for hardware implementations
• only local knowledge of the scene is necessary

Pipeline Disadvantages
• limited flexibility
• some algorithms would require different ordering of pipeline stages
• hard to achieve while still preserving compatibility
• only local knowledge of scene is available
• shadows, global illumination difficult

OpenGL
• API to graphics hardware
• based on IRIS GL by SGI
• designed to exploit hardware optimized for display and manipulation of 3D graphics
• implemented on many different platforms
• low level, powerful flexible
• pipeline processing
• set state as needed

OpenGL (briefly)
• modularity: logical separation of different components
• easy to parallelize
• earlier stages can already work on new data while later stages still work with previous data
• similar to pipelining in modern CPUs
• but much more aggressive parallelization possible (special purpose hardware!)
• important for hardware implementations
• only local knowledge of the scene is necessary

Graphics State
• set the state once, remains until overwritten
• glColor3f(1.0, 1.0, 0.0) → set color to yellow
• glSetClearColor(0.0, 0.0, 0.2) → dark blue bg
• glEnable(LIGHT0) → turn on light
• glEnable(GL_DEPTH_TEST) → hidden surf.

Geometry Pipeline
• tell it how to interpret geometry
• glBegin(mode of geometric primitives>)
• mode = GL_TRIANGLES, GL_POLYGON, etc.
• feed it vertices
• glVertex3f(1.0, 0.0, -1.0)
• glVertex3f(1.0, 0.0, -1.0)
• glVertex3f(0.0, 1.0, -1.0)
• tell it you’re done
• glEnd()

GLUT
void display()
void reshape(int w, int h);
void keyboard(unsigned char key, int x, int y);
void mouse(int button, int state, int x, int y);
void idle();
void display();

GLUT Callback Functions
// you supply these kind of functions
void reshape(int w, int h);
void keyboard(unsigned char key, int x, int y);
void mouse(int button, int state, int x, int y);
void idle();
void display();

// register them with glut
glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutMouseFunc(mouse);
 glutIdleFunc(idle);
 glutDisplayFunc(display);

more OpenGL as course continues
GLUT Example 1
#include <GLUT/glut.h>
void display(){
  glClearColor(0,0,0,1);
  glClear(GL_COLOR_BUFFER_BIT);
  glRotatef(0.1, 0,0,1);
  glRotatef(0.1, 0,0,1);
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_POLYGON);
  glVertex3f(0.25, 0.75, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glVertex3f(0.75, 0.25, -0.5);
  glVertex3f(0.25, 0.25, -0.5);
  glVertex3f(0.75, 0.25, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glVertex3f(0.25, 0.75, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glEnd();
  glutSwapBuffers();
}

glutCreateWindow("glut1");
glutDisplayFunc( display );
glutMainLoop();
return 0; // never reached

GLUT Example 2
#include <GLUT/glut.h>
void display(){
  glRotatef(0.1, 0,0,1);
  glClearColor(0,0,0,1);
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_POLYGON);
  glVertex3f(0.25, 0.75, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glVertex3f(0.75, 0.25, -0.5);
  glVertex3f(0.25, 0.25, -0.5);
  glVertex3f(0.75, 0.25, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glVertex3f(0.25, 0.75, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glEnd();
  glutSwapBuffers();
}

glutCreateWindow("glut2");
glutDisplayFunc( display );
glutMainLoop();
return 0; // never reached

GLUT Example 3
#include <GLUT/glut.h>
bool animToggle = true;
float angle = 0.1;
void display() {
  glutTimerFunc(50, doIdle, 0);
  glRotatef(angle, 0,0,1);
  glClearColor(0,0,0,1);
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_POLYGON);
  glVertex3f(0.25, 0.75, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glVertex3f(0.75, 0.25, -0.5);
  glVertex3f(0.25, 0.25, -0.5);
  glVertex3f(0.75, 0.25, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glVertex3f(0.25, 0.75, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glEnd();
  glutSwapBuffers();
}

void idle() {
  if (animToggle) {
    glRotatef(angle, 0,0,1);
    glutTimerFunc(50, doIdle, 0);
  }
  else if ('r' == key) {
    angle = -angle;
  }
  glutPostRedisplay();
}

Redrawing Display
• display only redrawn by explicit request
• glutPostRedisplay() function
• default window resize callback does this
• idle called from main loop when no user input
• good place to request redraw
• will call display next time through event loop
• should return control to main loop quickly
• continues to rotate even when no user action

GLUT Example 4
#include <GLUT/glut.h>
bool animToggle = true;
float angle = 0.1;
void display() {
  glutTimerFunc(50, doIdle, 0);
  glRotatef(angle, 0,0,1);
  glClearColor(0,0,0,1);
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_POLYGON);
  glVertex3f(0.25, 0.75, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glVertex3f(0.75, 0.25, -0.5);
  glVertex3f(0.25, 0.25, -0.5);
  glVertex3f(0.75, 0.25, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glVertex3f(0.25, 0.75, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glEnd();
  glutSwapBuffers();
}

void idle() {
  if (animToggle) {
    glutTimerFunc(50, doIdle, 0);
    glRotatef(angle, 0,0,1);
    glutTimerFunc(50, doIdle, 0);
  }
  else if ('r' == key) {
    angle = -angle;
  }
  glutPostRedisplay();
}

Keyboard/Mouse Callbacks
• again, do minimal work
• consider keypress that triggers animation
• do not have loop calling display in callback!
• what if user hits another key during animation?
• instead, use shared/global variables to keep track of state
• yes, OK to use globals for this!
• then display function just uses current variable value

Readings for Next Four Lectures
• FCG Chap 6 Transformation Matrices
  - except 6.1.6, 6.3.1
• FCG Sect 13.3 Scene Graphs
• RB Chap Viewing
  - Viewing and Modeling Transforms until Viewing Transformations
  - Examples of Composing Several Transformations through
    Building an Articulated Robot Arm
• RB Appendix Homogeneous Coordinates and Transformation Matrices
  - unit Perspective Projection
• RB Chap Display Lists