Correction: Vector-Vector Multiplication
- multiply: vector * vector = scalar
- dot product, aka inner product $u \cdot v = (u_1, v_1) + (u_2, v_2) + (u_3, v_3)$
- geometric interpretation: lengths, angles
- can find angle between two vectors $\theta = \cos^{-1}\left(\frac{u \cdot v}{\|u\|\|v\|}\right)$

Correction: Dot Product Example
- $u_1, u_2, v_1, v_2$
- $u_1 = (6, 1, 2)$
- $v_1 = (1, 7)$
- $u_2 = (2, 3)$

Review: Working with Frames
- $p = (o + x\mathbf{i} + y\mathbf{j})$
- $F_1 \rightarrow p = (3, -1)$
- $F_2 \rightarrow p = (-1.5, 2)$
- $F_3 \rightarrow p = (1, 2)$

Rendering
- goal: transform computer models into images
- may or may not be photo-realistic
- interactive rendering: fast, but limited quality
- offline rendering: ray tracing, global illumination

Rendering Pipeline
- what is the pipeline?
- abstract model for sequence of operations to transform geometric model into digital image
- abstraction of the way graphics hardware works
- underlying model for application programming interfaces (APIs) that allow programming of graphics hardware
- OpenGL
- Direct 3D
- actual implementation details of rendering pipeline will vary

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
### OpenGL (briefly)

- started in 1989 by Kurt Akeley
- based on IRIS_GL by SGI
- API to graphics hardware
- 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

### 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

### Graphics State

- set the state once, remains until overwritten
  - glColor3f(1.0, 0.0, 0.0) \(\to\) set color to yellow
  - glClearColor(0.0, 0.0, 0.0, 0.0) \(\to\) black color
  - glEnable(LIGHT0) \(\to\) turn on light
  - glEnable(GL_DEPTH_TEST) \(\to\) 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()

### Code Sample

```c
void display()
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 1.0, 0.0);
    glBegin(GL_TRIANGLES);
    glVertex3f(-1.0, 0.0, -1.0);
    glVertex3f(1.0, 0.0, -1.0);
    glVertex3f(0.0, 1.0, -1.0);
    glEnd();
    glFlush();
}
```

- more OpenGL as course continues
GLUT: OpenGL Utility Toolkit

- developed by Mark Kilgard (also from SGI)
- simple, portable window manager
- opening windows
- handling graphics contexts
- handling input with callbacks
  - keyboard, mouse, window reshape events
  - timing
  - idle processing, idle events
- designed for small-medium size applications
- distributed as binaries
- free, but not open source

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 special(int 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);

void glutDisplayFunc(void (*func)(void));
void glutKeyboardFunc (void (*func)(unsigned char key, int x, int y));
void glutMouseFunc (void (*func)(int button, int state, int x, int y));
void glutIdleFunc (void (*func)(void));

GLUT Draw World

int main(int argc, char **argv)
{
    glutInit (&argc, argv);
    glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize( 640, 480 );
    glutCreateWindow( "OpenGL/GLUT" );
    glutDisplayFunc(DrawWorld);
    glutIdleFunc(Idle);
    glClearColor( 1,1,1 );
    glutMainLoop();
    return 0; // never reached
}

GLUT Draw World

void DrawWorld() {
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glClear( GLUT_COLOR_BUFFER_BIT );
    if(keypress) {
        glRotatef(angle,0,0,1);
        Idle();
        glutSwapBuffers();
    }

    glClearColor(0.0,0.0,0.0,1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(angle,0,0,1);
    glLineWidth(3);
    glBegin(GL_TRIANGLES);
    glVertex3d(1,0,0);
    glVertex3d(0,1,0);
    glVertex3d(-1,0,0);
    glEnd();
    Idle();
    glutSwapBuffers();
    glutPostRedisplay();
}

Idle Function

void Idle() {
    angle += 0.05;
    glutPostRedisplay();
}

- called from main loop when no user input
- should return control to main loop quickly
- update value of angle variable here
- then request redraw event from GLUT
- draw function will be called next time through
- continues to rotate even when no user action

Keyboard/Mouse Callbacks

- do minimal work
- request redraw for display
- example: keypress triggering animation
- do not create loop in input callback!
- what if user hits another key during animation?
- shared/global variables to keep track of state
- display function acts on current variable value

Week 2 Lab

- labs start Tuesday
- project 0
  - http://www.openca.ubc.ca/~cs314/vjan2007/a0
  - make sure you can compile OpenGL/GLUT
  - very useful to test home computing environment
  - template: spin around obj files
  - todo: change rotation axis
  - do handin to test configuration, but not graded

Remote Graphics

- OpenGL does not work well remotely
  - very slow
  - only one user can use graphics at a time
- current X server doesn't give priority to console, just does first come first served
- problem: FCFS policy = confusion/chaos
- solution: console user gets priority
  - only use graphics remotely if nobody else is logged on
    - with 'who' command, "O" is console person
  - stop using graphics if asked by console user via email
  - or console user can reboot machine out from under you

Event-Driven Programming

- main loop not under your control
  - vs. batch mode where you control the flow
- control flow through event callbacks
  - redraw the window now
  - key was pressed
  - mouse moved
  - callback functions called from main loop when events occur
    - mouse/key/keyboard state setting vs. redrawing

Remote Graphics

- OpenGL does not work well remotely
  - very slow
  - only one user can use graphics at a time
- current X server doesn't give priority to console, just does first come first served
- problem: FCFS policy = confusion/chaos
- solution: console user gets priority
  - only use graphics remotely if nobody else is logged on
    - with 'who' command, "O" is console person
  - stop using graphics if asked by console user via email
  - or console user can reboot machine out from under you