News

• Labs start next week
• Reminder: my office hours Wed/Fri 11-12
  • in your 011 lab, not my X661 office
• Leftover handouts will be in 011 lab
Today’s Readings

• today
  • RB Chap Introduction to OpenGL
  • RB Chap State Management and Drawing Geometric Objects
  • RB App Basics of GLUT (Aux in v 1.1)
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
  • until Perspective Projection
• RB Chap Display Lists
Correction: Vector-Vector Multiplication

- multiply: vector \(*\) vector = scalar
- dot product, aka inner product

\[
\begin{bmatrix}
  u_1 \\
  u_2 \\
  u_3 \\
\end{bmatrix}
\cdot
\begin{bmatrix}
  v_1 \\
  v_2 \\
  v_3 \\
\end{bmatrix}
= (u_1 * v_1) + (u_2 * v_2) + (u_3 * v_3)
\]

- geometric interpretation
  - lengths, angles
  - can find angle between two vectors

\[
\mathbf{u} \cdot \mathbf{v} = \mathbf{u} \mathbf{v} \cos \theta
\]
Correction: Dot Product Example

\[
\begin{bmatrix}
u_1 \\
u_2 \\
u_3 \\
v_1 \\
v_2 \\
v_3 \\
\end{bmatrix} = (u_1 \cdot v_1) + (u_2 \cdot v_2) + (u_3 \cdot v_3)
\]

\[
\begin{bmatrix}
6 \\
1 \\
2 \\
\end{bmatrix} \cdot \begin{bmatrix}
1 \\
7 \\
3 \\
\end{bmatrix} = (6 \cdot 1) + (1 \cdot 7) + (2 \cdot 3) = 6 + 7 + 6 = 19
\]
Review: Working with Frames

\[ p = o + xi + yj \]

\( F_1 \) \( p = (3, -1) \)

\( F_2 \) \( p = (-1.5, 2) \)

\( F_3 \) \( p = (1, 2) \)
More: Working with Frames

\[ p = o + xi + yj \]

\[ F_1 \quad p = (3,-1) \]
\[ F_2 \quad p = (-1.5,2) \]
\[ F_3 \quad p = (1,2) \]
More: Working with Frames

\[ \mathbf{p} = \mathbf{o} + x\mathbf{i} + y\mathbf{j} \]

\[ F_1 \quad \mathbf{p} = (3, -1) \]

\[ F_2 \quad \mathbf{p} = (-1.5, 2) \]

\[ F_3 \quad \mathbf{p} = (1, 2) \]
More: Working with Frames

\[
p = o + xi + yj
\]

\[
F_1 \quad p = (3,-1)
\]

\[
F_2 \quad p = (-1.5,2)
\]

\[
F_3 \quad p = (1,2)
\]
Rendering

• goal
  • transform computer models into images
  • may or may not be photo-realistic

• interactive rendering
  • fast, but limited quality
  • roughly follows a fixed patterns of operations
    • rendering pipeline

• offline rendering
  • ray tracing
  • global illumination
Rendering

• tasks that need to be performed (in no particular order):
  • project all 3D geometry onto the image plane
    • geometric transformations
  • determine which primitives or parts of primitives are visible
    • hidden surface removal
  • determine which pixels a geometric primitive covers
    • scan conversion
  • compute the color of every visible surface point
    • lighting, shading, texture mapping
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
Rendering Pipeline

Geometry Database ➔ Model/View Transform. ➔ Lighting ➔ Perspective Transform. ➔ Clipping

Scan Conversion ➔ Texturing ➔ Depth Test ➔ Blending ➔ Frame-buffer
Geometry Database

- 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
Lighting

- lighting
  - compute brightness based on property of material and light position(s)
  - computation is performed *per-vertex*
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
Clipping

- clipping
  - removal of parts of the geometry that fall outside the visible screen or window region
  - may require *re-tessellation* of geometry
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
  • “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
  • video memory on graphics board that holds image
  • double-buffering: two separate buffers
    • draw into one while displaying other, then swap to avoid flicker
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 (briefly)
OpenGL

- 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
Graphics State

• set the state once, remains until overwritten
  • **glColor3f(1.0, 1.0, 0.0)** ◊ set color to yellow
  • **glClearColor(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_TRIANGLE, 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()`
Open GL: Geometric Primitives

```c
glPointSize( float size);
glLineWidth( float width);
glColor3f( float r, float g, float b);
```

...
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_POLYGON);
    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);
    glEnd();
    glFlush();
}

• more OpenGL as course continues
GLUT
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 Draw World

int main(int argc, char **argv)
{
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_RGB |
                        GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize( 640, 480 );
    glutCreateWindow( "openGLDemo" );
    glutDisplayFunc( DrawWorld );
    glutIdleFunc(Idle);
    glClearColor( 1,1,1 );
    glutMainLoop();

    return 0;       // never reached
}
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/keyboard state setting vs. redrawing
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 but, 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 glutIdleFunc (void (*func)());
void glutReshapeFunc (void (*func)(int width, int height));
Display Function

```c
void DrawWorld() {
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    glClear( GL_COLOR_BUFFER_BIT );
    angle += 0.05;                //animation
    glRotatef(angle,0,0,1);        //animation
    ... // redraw triangle in new position
    glutSwapBuffers();
}
```

- directly update value of angle variable
  - so, why doesn't it spin?
- only called in response to window/input event!
Idle Function

```c
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
Labs
Week 2 Lab

• labs start Tuesday
• project 0

• 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 logged on
    - with ‘who’ command, “:0” is console person
  - stop using graphics if asked by console user via email
  - or console user can reboot machine out from under you