Lecture 7: OpenGL
Input and Interaction
Todays lecture

• What is OpenGL?
• How does it work?
  – Primitives: Points, vertices, lines, polygons
  – Input and interaction
• Code examples

Common primitives in OpenGL

Input devices
Open Graphics Library
API for drawing 2D and 3D graphics
Introduced in 1992
Version 4.3 released in 2012
Platform independent (UNIX, Windows, Mac OS…)
Often hardware supported in graphics cards
Language binding to C, C++, Java, Fortran, Python, Perl,...
Toolkits

- OpenGL is a “low level” graphics library
- GLU (OpenGL Utility Library)
  - Support for higher level graphics, such as spheres, cylinders, NURBS, etc.
  - Delivered with OpenGL
  - Methods begin with glu, e.g.
    
    ```
    gluSphere(...);
    gluLookAt(...);
    ```
Toolkits

• GLUT (OpenGL Utility Toolkit)
  – Simplifies window handling
  – User interface functions
  – Available from opengl.org
  – Methods begin with glut, e.g.
    glutInitWindowSize( 700, 700 );

• GLEW (OpenGL Extension Wrangler Library)
  – Facilitates usage of OpenGL Extensions
  – (http://www.opengl.org/registry/)
Mesa (http://www.mesa3d.org/)

- Mesa is a fully open source 3-D graphics library with an API which is very similar to that of OpenGL.
- You can consider Mesa “to be OpenGL”; you can use OpenGL documentation, for example.
OpenGL advantages

- Industry Standard
- Stable
  - Well controlled specification
  - Backward compatibility
- Reliable and portable
  - Consistent visual display results
OpenGL advantages

• Evolving
  – Extension mechanism (http://www.opengl.org/registry/)

• Scalable
  – Run on systems ranging from cell phones and PDA:s to PCs, workstations, and supercomputers

• Documentation
  – Numerous books, web material, and sample code is readily available
State machine

- OpenGL is designed as a state machine
- Inputs geometric primitives, outputs bitmaps
- The state machine converts the inputs to an output image
- The result depends on the current state
- Examples of states: Colors, shading, texture,...
The OpenGL Machine

http://www.opengl.org
Primitives in OpenGL

- Primitives are “basic building blocks” for graphics in OpenGL.
- Some primitive types
  - GL_POINTS
  - GL_LINES (Used in Lab 1!)
  - GL_LINESTRIP
  - GL_TRIANGLES
  - GL_QUADS
  - GL_POLYGON

Complex shapes can be built by using many primitives!
• Primitive defining statements all starts with `glBegin(<primitive_type>)` and ends with `glEnd()`
• Vertices are defined using `glVertex*()`
  • where * can be `2f`, `2d`, `3f`, `3d`, ...
• Ex.: void `glVertex2f(GLfloat vx, GLfloat vy);`
• `glVertex2fv(GLfloat *v)` (pointer to array)
• `GLfloat` typedefined as `float` (32 bit)
• `GLdouble` typedefined as `double` (64 bit)
• Colors, normals, texture coordinates can be specified for each vertex
Primitives in OpenGL

GL_POINTS

GL_LINES

GL_LINE_STRIP

GL_LINE_LOOP
Primitives in OpenGL

- GL_POINTS
- GL_POLYGON
- GL_QUADS
- GL_TRIANGLES
Primitives in OpenGL

GL_POINTS

GL_TRIANGLE_STRIP

GL_QUAD_STRIP
glBegin(GL_LINES);
glVertex2f(-0.5, -0.5); // 1
glVertex2f(0.5, -0.5); // 2
glVertex2f(0.5, 0.5); // 3
glVertex2f(-0.5, 0.5); // 4
glEnd();
glBegin(GL_LINE_STRIP);
glVertex2f(-0.5,-0.5); // 1
glVertex2f( 0.5,-0.5); // 2
glVertex2f( 0.5, 0.5); // 3
glVertex2f(-0.5, 0.5); // 4
glEnd();
glBegin(GL_LINE_LOOP);
glVertex2f(-0.5, -0.5);  // 1
glVertex2f(0.5, -0.5);    // 2
glVertex2f(0.5, 0.5);     // 3
glVertex2f(-0.5, 0.5);    // 4
glEnd();
```
glBegin(GL_TRIANGLES);
glVertex3f(-0.5, -0.5, 0.0); // 1
glVertex3f(0.5, -0.5, 0.0);  // 2
glVertex3f(0.25, 0.5, 0.0); // 3
glVertex3f(-0.5, 1.25, 0.0); // 4
glVertex3f(0.5, 1.25, 0.0); // 5
glVertex3f(0.25, 0.75, 0.0); // 6
glEnd();
```
```
glBegin(GL_TRIANGLE_STRIP);
glVertex3f(-0.5, -0.5, 0.0); // 1
glVertex3f(0.5, -0.5, 0.0); // 2
glVertex3f(0.25, 0.5, 0.0); // 3
glVertex3f(0.5, 0.75, 0.0); // 4
glEnd();
```
Front and back rendering

• Each polygon has two sides, front and back
• OpenGL can render the two differently
• The ordering of vertices determines which is the front side:
  – By default, when looking at the front side, the vertices go counterclockwise (GL_CCW)
  – This is basically the right-hand rule
  – You can change this with glFrontFace(GL_CW);

```c
// culling
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);
```
glColor3f(1.0, 0.0, 0.0); // red
glBegin(GL_TRIANGLES);
glVertex3f(...);
...
glEnd();
Color per vertex

```cpp
glBegin(GL_TRIANGLES);
setColor3f(...);
vertex3f(...);
setColor3f(...);
vertex3f(...);
...
 glEnd();
```
A very simple program (Primitives.cpp)

#include <GL/glut.h>

void initGLUT(int &argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(700, 700);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("My Program");
}

void initGL()
{
    glClearColor( 0.0, 0.0, 0.0, 0.0 );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glOrtho( 0.0, 1.0, 0.0, 1.0, -1.0, 1.0 );
    glMatrixMode ( GL_MODELVIEW );
}

Could use gluPerspective(...), or gluLookAt(...) instead!
void draw() {
    float point2[2] = {0.5, 0.75};
    glClear( GL_COLOR_BUFFER_BIT );
    glColor3f( 1.0, 1.0, 1.0 );
    glBegin( GL_POLYGON );
        glVertex2f( 0.25, 0.25 );
        glVertex2f( 0.75, 0.5 );
        glVertex2fv( point2 );
    glEnd();
    glFlush();
}
int main(int argc, char **argv) {
    initGLUT(argc, argv);
    initGL();

    // Set the display callback
    glutDisplayFunc(draw);

    // Start the GLUT main event loop.
    glutMainLoop();
    return 0;
}

Primitives.cpp
Coding time!
Input and interaction

- Event driven: CPU waits on the device before it does anything
- Examples:
  - keyboard
  - Mouse
  - Joystick
Callback functions

- Used for input and interaction
- The user submits a pointer to a function that should be called when the corresponding event occurs
- GLUT provides an easy-to-use interface
Callback functions

```c
glutMouseFunc(function);  // click mouse
glutMotionFunc(function);  // move mouse
glutPassiveMotionFunc(function);  // no button
glutReshapeFunc(function);  // window resize
glutKeyboardFunc(function);  // keyboard
glutSpecialFunc(function);  // arrows, pgup
glutJoystickFunc(function);  // joystick
glutIdleFunc(function);  // animation
glutDisplayFunc(function);  // draw primitives
```
int main(int argc, char **argv) {
    initGLUT(argc, argv);
    initGL();

    //Set the display callback
    glutDisplayFunc(draw);

    //Set the keyboard callback
    glutKeyboardFunc(keyboard);

    //Start the GLUT main event loop.
    glutMainLoop();
    return 0;
}
//keyboard callback
void keyboard(unsigned char key, int x, int y) {
    switch(key){
    case 'r':
        r=1.0; g=0.0; b=0.0;
        break;
    case 'g':
        r=0.0; g=1.0; b=0.0;
        break;
    }
    glutPostRedisplay();
}
void draw() {
...

    //draw a polygon
    glColor3f( r, g, b );
    glBegin( GL_POLYGON );
        glVertex2f( 0.25, 0.25 );
        glVertex2f( 0.75, 0.5 );
        glVertex2fv( point2 );
    glEnd();
...
}
Keyboard interaction

'C' Coding time! 'G'

Program1

Red triangle

Program1

Green triangle
int main(int argc, char **argv) {
    ...

    // Set the keyboard callback
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
    glutPassiveMotionFunc(passiveMotion);
    glutMotionFunc(motion);

    ...
}
Mouse interaction

// mouse callback
void mouse(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        printf("Left mouse button pressed");
    }
}
Mouse interaction

```
// mouse motion callback
void motion(int x, int y) {
    pos_x = x / WINDOW_SIZE_X;
    pos_y = 1 - (y / WINDOW_SIZE_Y);
    glutPostRedisplay();
}
```
In the display function

```cpp
void draw() {
...
    //draw a point
    glColor3f( 0.8, 0.8, 0.2 );
    glPointSize(10.0);
    glBegin( GL_POINTS );
        glVertex2f( pos_x, pos_y );
    glEnd();
...
}
```
Mouse interaction

Coding time!
idle tasks

```c
int main(int argc, char **argv) {
    ...

    glutKeyboardFunc(keyboard);

    glutMouseFunc(mouse);
    glutPassiveMotionFunc(motion);

    glutIdleFunc(idle);

    ...
}
```
Idle tasks

// idle callback
void idle() {
    t+=1;
    glutPostRedisplay();
}
void draw() {
    ...
    glColor3f( 0.5+0.5*sin(0.001*t),
               0.3,
               0.5+0.5*cos(0.001*t));

    // draw a polygon
    glBegin( GL_POLYGON );
    glVertex2f( 0.25, 0.25 );
    ...
    glEnd();
    ...
}

Coding time!
OpenGL in GUI applications

• Basic keyboard and mouse interaction sufficient for small applications.
• Often, we need more (menus, buttons, sliders,... )
GLUI
(http://www.cs.unc.edu/~rademach/glui/)

- Minimal library for GUI:s, really easy to use
- All rendering done with OpenGL
- Good for simple applications
- Limited functionality
• Multi-platform (Windows, OS X, Linux)
• Open Source
• Large feature set
• OpenGL supported through `wxGLCanvas` class.
Qt (http://qt.nokia.com/)

- Multi-platform (Windows, OS X, Linux)
- Open Source (since 2008)
- Large feature set
- OpenGL support.
Questions so far...?
Interaction with more degrees of freedom

Haptic interfaces provide multiple degrees of freedom and force feedback. Enables exciting interaction possibilities…