2-D OpenGL

Fall Semester
Topic Outline

• Program Structure
  – Boilerplate settings
  – Registering callbacks
  – The main event loop

• Drawing Commands
  – Colors
  – Vertices
  – Geometric primitives
  – Animation

• Interaction
  – Mouse clicked
  – Mouse dragged
  – Key pressed
  – Window resized

• Integrating OpenGL and MPI
  – Manager versus Worker
  – Send/receive signals
  – Modifying the script
2-D OpenGL

Boilerplate Settings (1)

```c
#include <GL/glut.h>
...
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(w, h);
    glutInitWindowPosition(100, 50);
    glutCreateWindow("Mandelbrot Set");
```
Boilerplate Settings (2)

- See also, the Red Book.

```c
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
// single buffering, flickers

glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
// double buffering, eliminates flicker

glShadeModel(GL_SMOOTH);
// per vertex coloring of geometric primitives
```
Callback Functions (1)

- Name of function passed to function pointer parameter.

```c
main:
    glutDisplayFunc(display);
    glutIdleFunc(NULL);
    glutMouseFunc(mouse);
    glutMotionFunc(motion);
    glutKeyboardFunc(keyfunc);
    glutReshapeFunc(reshape);
```
Callback Functions (2)

• Function header must match expected signature:
  – Same parameter list.
  – Same return type.

• Names are irrelevant, only the order of types matters.

```c
void display(void)
void mouse(int button,int state,int xscr,int yscr)
void motion(int xscr,int yscr)
void keyfunc(unsigned char key,int xscr,int yscr)
void reshape(int wscr,int hscr)
```
Main Event Loop

- The command `glutMainLoop` takes control of all execution.
- Whenever the window needs to be redrawn the registered display function is automatically called by the main loop.
- As the user interacts with the window:
  - Mouse clicks trigger the registered mouse function.
  - Mouse drags trigger the registered motion function.
  - Key presses trigger the registered keyboard function.
  - A window resize triggers the registered reshape function.
- When nothing else is happening the registered idle function is called over and over and over again; passing `NULL` disables this.
Colors

- The command `glClearColor`, called in `main`, sets the color for clearing the window using the RGBA color model.
- The command `glClear`, called in `display`, actually clears the screen. In single buffering mode this is why the window flickers whenever it’s redrawn.
- The command `glColor3f` sets the current drawing color using the RGB color model (i.e., no alpha value for transparency).

| Transparency: 0.0 is opaque and 1.0 is transparent |
| RGB values: 0.0 is none and 1.0 is full          |
| Color values: 0.0 0.0 0.0 0.0 is black, 1.0 1.0 1.0 1.0 is white |
Vertices

- The command \texttt{gluOrtho2D}, called in \texttt{reshape}, sets the OpenGL coordinate system while \texttt{glutInitWindowSize}, called in \texttt{main}, sets the window coordinate system. These are \textit{not}!!! the same.

- To convert from window coordinates to OpenGL coordinates:

\begin{verbatim}
for(k=0;k<h;k++)
  for(j=0;j<w;j++)
  {
    ...
    x=xmin+(xmax-xmin)*(j)/w;
    y=ymin+(ymax-ymin)*(h-k)/h; // inverted
  }
\end{verbatim}
Geometric Primitives (1)

- Vertices build primitives: points, lines, line_strip, line_loop, triangles, triangle_strip, triangle_fan, quads, quad_strip, polygon

```c
    glBegin(GL_POINTS);
    for(k=0;k<h;k++)
        for(j=0;j<w;j++)
            ...
            glVertex2f(x,y);
    glEnd();
```
Geometric Primitives (2)

- The GL_LINES primitive would only connect vertices in pairs, not in sequence; GL_LINE_LOOP connects the last to the first.
- The GL_POLYGON primitive fills in the interior of the polygon.
Animation

- In double buffering mode `glutSwapBuffers` replaces `glFlush` in the display function. Then, call `glutPostRedisplay` in the idle function to manually redraw the window for each frame.

```c
// draw something based on the value of xpos
glutSwapBuffers();

oxpos++;

glutPostRedisplay();
```
Mouse Clicked and Mouse Dragged

```c
void mouse(int button,int state,int xscr,int yscr)
{
    if(button==GLUT_LEFT_BUTTON)
    {
        if(state==GLUT_DOWN)
        {
            ...
        }
    }

    void motion(int xscr,int yscr)
    {
        printf("(x,y)=%d,%d\n",xscr,yscr);
    }
```
Key Pressed

- An **unsigned** variable uses the sign-bit to represent larger values instead of negative numbers. The **exit** function is from **stdlib.h** and zero indicates normal program termination.

```c
void keyfunc(unsigned char key,int xscr,int yscr)
{
    if(key=='q')
    {
        exit(0);
    }
}
```
Window Resized

```c
void reshape(int wscr,int hscr)
{
    w=wscr; h=hscr;
    glViewport(0,0,(GLsizei)w,(GLsizei)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    // aspect ratio calculation
    gluOrtho2D(xmin,xmax,ymin,ymax);
    glMatrixMode(GL_MODELVIEW);
}
```
Integrating OpenGL and MPI (1)

- Easy scheme. Only the Manager calls OpenGL commands.

```c
main:
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    if(rank==0)
    {
        glutInit(&argc,argv);
        ...
        glutMainLoop();
```
Integrating OpenGL and MPI (2)

- Workers loop while waiting for signals from the Manager.

```c
main:
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    if(rank==0)
        ...
    else
        while(1)
            {
                MPI_Recv(...
```
Integrating OpenGL and MPI (3)

- Manager’s callbacks send signals to the Workers.

```c
keyfunc:
  if (key==‘q’)
  {
    loop:
      MPI_Send(... // kill signal
      MPI_Finalize();
    exit(0);
  }
```