TA (Amy Platt) office hours

M and W, 1-2

GS 929b
CRT display (getting rare!)
[ H&B, pp 36-44]
CRT Displays

- Phosphors glow when hit by electron beam.
- Color is adjusted via intensity of beam delivered to each of R, G, and B phosphor.
- CRT display phosphors glow for limited time--need to be refreshed.
- Raster displays refresh by scanning from top to bottom in left right order.
- Timing is used to make screen elements correspond to memory elements.
CRT Displays

- Typical refresh rate is 75 per second
- May have many phosphor dots corresponding to one memory element (old stuff), but more usually one per phosphor trio.
- Memory elements called pixels
- Refresh method creates architectural and programming issues (e.g. double buffering), defines “real time” in animation.
Flat Panel TFT* Displays

[ H&B, pp 44-47]

*Thin film transistor
From http://www.atip.or.jp/fpd/src/tutorial
From http://www.atip.or.jp/fpd/src/tutorial
3D displays

Use some scheme to control what each eye sees
Color, temporal + shutter glasses, polarization + glasses
OpenGL and GLUT

[ H&B, §2.9, pp 73-80]

Demo and discussion of example program

http://www.cs.arizona.edu/classes/cs433/fall05/triangle.c
OpenGL and GLUT

• Layer between your program and lower levels (hardware, low level display issues)
• Provides primitives
  – points
  – lines
  – polygons
  – bitmaps, fonts
• Provides standard graphics facilities
  – We will learn how some of these work. Some assignments will therefore have some routines “out of bounds”
  – GLUT simplifies interactive program development with intuitive callbacks and additional facilities (menus, window management).
OpenGL and GLUT

- Initialization code from the example

```c
/* initialize GLUT system */
glutInit(&argc, argv);

glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
glutInitWindowSize(400,500);  /* width=400pixels height=500pixels */
win = glutCreateWindow("Triangle");  /* create window */

/* From this point on the current window is win */

/* set background to black */
glClearColor((GLclampf)0.0,(GLclampf)0.0,(GLclampf)0.0,(GLclampf)0.0);
gluOrtho2D(0.0,400.0,0.0,500.0);  /* how object is mapped to window */
```
OpenGL and GLUT

- Window display callback. You will likely also call this function. Window repaintiong on expose and resizing is done for you

```c
/* set window's display callback */
glutDisplayFunc(display_CB);
```
static void display_CB(void)
{
    glClear(GL_COLOR_BUFFER_BIT);        /* clear the display */

    /* set current color */
    glColor3d(triangle_red, triangle_green, triangle_blue);

    /* draw filled triangle */
    glBegin(GL_POLYGON);

    /* specify each vertex of triangle */
    glVertex2i(200 + displacement_x, 125 - displacement_y);
    glVertex2i(100 + displacement_x, 375 - displacement_y);
    glVertex2i(300 + displacement_x, 375 - displacement_y);

    glEnd();           /* OpenGL draws the filled triangle */
    glFlush();         /* Complete any pending operations */

    glutSwapBuffers(); /* Make the drawing buffer the frame buffer and vice versa */
}
OpenGL and GLUT

• User input is through callbacks, e.g.,

/* set window's key callback */
glutKeyboardFunc(key_CB);

/* set window's mouse callback */
glutMouseFunc(mouse_CB);

/* set window's mouse move with button pressed callback */
glutMotionFunc(mouse_move_CB);
static void key_CB(unsigned char key, int x, int y)
{
    if(key == 'q') exit(0);
}

/* */

/* Function called on mouse click */
static void mouse_CB(int button, int state, int x, int y)
{
    /*
       Code which responses to the button, the state (press, release), and where
       the pointer was when the mouse event occured (x, y).
       
       See example on-line for sample code.
    */
}

/* */

/* Function called on mouse move while depressed. */
static void mouse_move_CB(int x, int y)
{
    /* See example on-line for sample code. */
}
OpenGL and GLUT

- GLUT makes pop-up menus easy. We will save development time by using (perhaps abusing) this facility.

```c
/* Create a menu which is accessed by the right button. */
submenu = glutCreateMenu(select_triangle_color);
glutAddMenuEntry("Red", KJB_RED);
glutAddMenuEntry("Green", KJB_GREEN);
glutAddMenuEntry("Blue", KJB_BLUE);
glutAddMenuEntry("White", KJB_WHITE);
glutCreateMenu(add_object_CB);
glutAddMenuEntry("Triangle", KJB_TRIANGLE);
glutAddMenuEntry("Square", KJB_SQUARE);
glutAddSubMenu("Color", submenu);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```
OpenGL and GLUT

- Ready for the user!

```c
/* start processing events... */
glutMainLoop();
```

- For the rest of the code see
  
  [http://www.cs.arizona.edu/classes/cs433/fall05/triangle.c](http://www.cs.arizona.edu/classes/cs433/fall05/triangle.c)
Displaying lines

• Assume for now:
  – lines have integer vertices
  – lines all lie within the displayable region of the frame buffer

• Other algorithms will take care of these issues.
Displaying lines

• Assume for now:
  – lines have integer vertices
  – lines all lie within the displayable region of the frame buffer

• Other algorithms will take care of these issues.

• Consider lines of the form \( y = mx + c \), where \( 0 < m < 1 \)

• Other cases follow by symmetry

• (Boundary cases, e.g. \( m=0, m=1 \) also work in what follows, but are often considered separately, because they can be done very quickly as special cases).
Displaying lines

- Variety of naive (poor) algorithms:
  - step x, compute new y at each step by equation, rounding
  - step x, compute new y at each step by adding m to old y, rounding
Bresenham’s algorithm

[ H&B, pp 95-99]

• Plot the pixel whose y-value is closest to the line
• Given \((x_k, y_k)\), must **choose** from either \((x_k+1, y_k+1)\) or \((x_k+1, y_k)\)---recall we are working on case \(0<m<1\)
• Idea: compute value that will determine this choice that is easy to update and cheap to compute (no floating point operations if endpoints are integral).
To integral endpoint

\[ y_{k+1} \]

Paint this one?

\[ y_k \]

Or this one?

To integral endpoint

\[ x_k \quad x_{k+1} \]
$y_{k+1}$

$y_k$

$d_2$

$d_1$

$x_k$

$x_{k+1}$

Paint this one?

Or this one?

To integral endpoint

Actual point on line $y = mx + b$
Bresenham’s algorithm

- Determiner is $d_1 - d_2$
  
  $d_1 - d_2 < 0 \Rightarrow \text{plot at } y_k$ (same level as previous)
  
  otherwise $\Rightarrow \text{plot at } y_k + 1$ (otherwise)

[Diagram showing the determination process for plotting points along a line]

Actual point on line $y = mx + b$
(Current point is, \((x_k, y_k)\) line goes through \((x_k + 1, y)\) )

\[d_1 = y - y_k \quad \text{and} \quad d_2 = (y_k + 1) - y\]

So \[d_1 - d_2 = (y - y_k) - ((y_k + 1) - y)\]

Plugging in \[y = m(x_k + 1) + b\]

Gives:

\[d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1\]
Avoiding Floating Point

From the previous slide

\[ d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1 \]

Recall that,

\[ m = \frac{(y_{end} - y_{start})}{(x_{end} - x_{start})} = \frac{dy}{dx} \]

So, for integral endpoints we can avoid floating point if we scale by a factor of \( dx \). Use determiner \( P_k \).

\[ p_k = (d_1 - d_2)dx \]
\[ = (2m(x_k + 1) - 2y_k + 2b - 1)dx \]
\[ = 2(x_k + 1)dy - 2y_k(dx) + 2b(dx) - dx \]
\[ = 2(x_k)dy - 2y_k(dx) + \text{constant} \]
Incremental Update

From previous slide

\[ p_k = 2(x_k)dy - 2y_k(dx) + \text{constant} \]

Finally, express the next determiner in terms of the previous, and in terms of the decision on the next \( y \).

\[ p_{k+1} = 2(x_k + 1)dy - 2y_{k+1}(dx) + \text{constant} \]
\[ = p_k + 2dy - 2(y_{k+1} - y_k) \]

Either 1 or 0 depending on decision on \( y \)
Bresenham algorithm

- \( p_{k+1} = p_k + 2 \, dy - 2 \, dx \, (y_{k+1}-y_k) \)

- Exercise: check that \( p_0 = 2 \, dy - dx \)

- Algorithm (for the case that \( 0 < m < 1 \)):
  - \( x=x_{\text{start}}, y=y_{\text{start}}, p=2 \, dy - dx, \textbf{mark} \ (x, y) \)
  - until \( x=x_{\text{end}} \)
    - \( x=x+1 \)
      - \( p>0 \ ? y=y+1, \textbf{mark} \ (x, y), p=p+2 \, dy - 2 \, dx \)
      - \( \text{else} \ y=y, \textbf{mark} \ (x, y), p=p+2 \, dy \)

- Some calculations can be done once and cached.
Issues

- End points may not be integral due to clipping (or other reasons)
- Brightness is a function of slope.
- Discretization problems “aliasing” (related to previous point).
Line drawing--simple line (Bresenham) brightness issues

8 pixels per $8\sqrt{2}$ length

8 pixels for 8 length
(Brighter)
Line drawing--discretization artifacts (often called aliasing)