

---

# CIS 454 Computer Graphics

## Lecture 9, 10/03/2006

---

Li Shen  
Computer and Information Science  
UMass Dartmouth

---

## Notes

- HW1 sample solution in the portal
    - Submit hard copy of your code and sample runs
    - Do not put code in rtf, doc files
-

---

## Working with Callbacks

- Learn to build interactive programs using GLUT callbacks
    - Mouse
    - Keyboard
    - Reshape
  - Introduce menus in GLUT
- 

---

## The Mouse Callback

```
glutMouseFunc(mymouse)
```

```
void mymouse(GLint button, GLint state, GLint x, GLint y)
```

- Issues
  - Positioning (OpenGL vs Screen):  $y = h - y$ ;
  - Obtain window size
  - Terminate program
  - Use mouse position

[square.c](#)

---

---

## The Motion Callback

- We can draw squares (or anything else) continuously as long as a mouse button is depressed by using the motion callback
    - `glutMotionFunc(drawSquare)`
  - We can draw squares without depressing a button using the passive motion callback
    - `glutPassiveMotionFunc(drawSquare)`
- 

---

## The Keyboard Callback

```
glutKeyboardFunc(mykey)
void mykey(unsigned char key,
            int x, int y)
```

- Returns ASCII code of key depressed and mouse location

```
void mykey()
{
    if(key == 'Q' | key == 'q')
        exit(0);
}
```

---

---

## The Reshape Callback

`glutReshapeFunc(myreshape)`

`void myreshape(int w, int h)`

- Returns width and height of new window (in pixels)
  - A redisplay is posted automatically at end of execution of the callback
  - GLUT has a default reshape callback but you probably want to define your own
  - The reshape callback is good place to put viewing functions because it is invoked when the window is first opened
- 

---

## Menus

- GLUT supports pop-up menus
  - A menu can have submenus
- Three steps
  - Define entries for the menu
  - Define action for each menu item
    - Action carried out if entry selected
  - Attach menu to a mouse button

[paint.c](#)

---

## Defining a simple menu

- In `main.c`

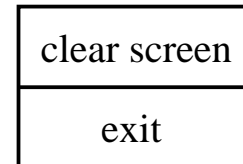
```

menu_id = glutCreateMenu(mymenu);
glutAddmenuEntry("clear Screen", 1);

gluAddMenuEntry("exit", 2);

glutAttachMenu(GLUT_RIGHT_BUTTON);

```



entries that appear when  
right button depressed

identifiers

paint.c

## Menu actions

- Menu callback

```

void mymenu(int id)
{
    if(id == 1) glClear();
    if(id == 2) exit(0);
}

```

- Note each menu has an id that is returned when it is created
- Add submenus by

```
glutAddSubMenu(char *submenu_name, submenu id)
```

entry in parent menu

---

# Better Interactive Programs

---

---

## Outline

- Learn to build more sophisticated interactive programs using
    - Picking
      - Select objects from the display
      - Three methods
    - Rubberbanding
      - Interactive drawing of lines and rectangles
    - Display Lists
      - Retained mode graphics
-

---

## Picking

- Identify a user-defined object on the display
  - In principle, it should be simple because the mouse gives the position and we should be able to determine to which object(s) a position corresponds
  - Practical difficulties
    - Pipeline architecture is feed forward, hard to go from screen back to world
    - Complicated by screen being 2D, world is 3D
    - How close do we have to come to object to say we selected it?
- 

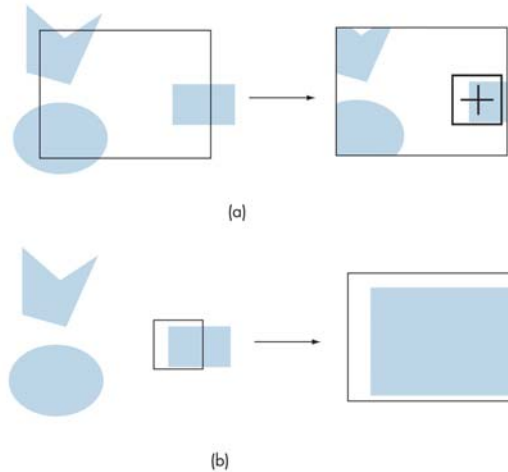
---

## Three Approaches

- Hit list
    - Clipping region: most general approach but most difficult to implement
  - Use back or some other buffer to store object ids as the objects are rendered
  - Rectangular maps
    - Easy to implement for many applications
    - See [paint](#) program in text
-

## Hit List

- Adjust clipping region
- Keep tracking of which primitives are rendered
- Include those into a hit list



## Rendering Modes

- OpenGL can render in one of three modes selected by `glRenderMode(mode)`
  - `GL_RENDER`: normal rendering to the frame buffer (default)
  - `GL_FEEDBACK`: provides list of primitives rendered but no output to the frame buffer
  - `GL_SELECTION`: Each primitive in the view volume generates a *hit record* that is placed in a *name stack* which can be examined later

[pick.c](#)

---

## Selection Mode Functions

- `glSelectBuffer()`: specifies name buffer
  - `glInitNames()`: initializes name buffer
  - `glPushName(id)`: push id on name buffer
  - `glPopName()`: pop top of name buffer
  - `glLoadName(id)`: replace top name on buffer
- 
- id is set by application program to identify objects
- 

---

## Using Selection Mode

- Initialize name buffer
  - Enter selection mode (using mouse)
  - Render scene with user-defined identifiers
  - Reenter normal render mode
    - This operation returns number of hits
  - Examine contents of name buffer (hit records)
    - Hit records include id and depth information
-

## Selection Mode and Picking

- As we just described it, selection mode won't work for picking because every primitive in the view volume will generate a hit
- Change the viewing parameters so that only those primitives near the cursor are in the altered view volume
  - Use `gluPickMatrix` (see text for details)

## Name Buffer

- Hit record
  - Number of names on the name stack where there was a hit
  - Scaled minimum and maximum depths for the hit primitive: useful for 3D

```
void drawObjects(GLenum mode)
{
    if(mode == GL_SELECT) glLoadName(1);
    glColor3f(1.0, 0.0, 0.0);
    glRectf(-0.5, -0.5, 1.0, 1.0);
    if(mode == GL_SELECT) glLoadName(2);
    glColor3f(0.0, 0.0, 1.0);
    glRectf(-1.0, -1.0, 0.5, 0.5);
}
```

```
void processHits (GLint hits, GLuint buffer[])
{
    unsigned int i, j;
    GLuint names, *ptr;

    printf ("hits = %d\n", hits);
    ptr = (GLuint *) buffer;
    for (i = 0; i < hits; i++)
    { /* for each hit */
        names = *ptr;
        ptr+=3;
        for (j = 0; j < names; j++)
        { /* for each name */
            if(*ptr==1) printf ("red rectangle\n");
            else printf ("blue rectangle\n");
            ptr++;
        }
        printf ("\n");
    }
}
```

---

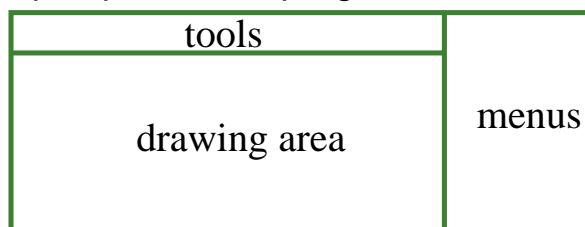
## Using Another Buffer and Colors

- For a small number of objects, we can assign a unique color (often in color index mode) to each object
  - We then render the scene to a color buffer other than the front buffer so the results of the rendering are not visible
  - We then get the mouse position and use `glReadPixels()` to read the color in the buffer we just wrote at the position of the mouse
  - The returned color gives the id of the object
- 

---

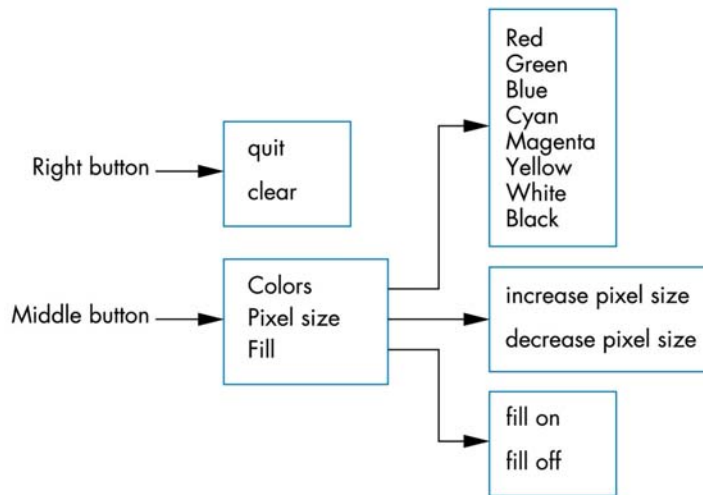
## Using Regions of the Screen

- Many applications use a simple rectangular arrangement of the screen
  - Example: paint/CAD program



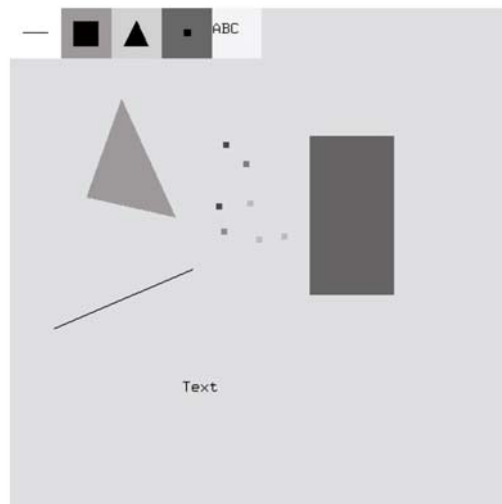
- Easier to look at mouse position and determine which area of screen it is in than using selection mode picking
-

# Paint Program



# Output from Paint Program

paint.c



---

## Three Approaches for Picking

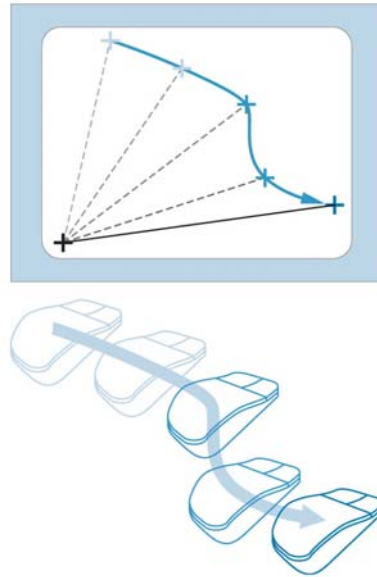
- Hit list
    - Clipping region: most general approach but most difficult to implement
  - Use back or some other buffer to store object ids as the objects are rendered
  - Rectangular maps
    - Easy to implement for many applications
    - See [paint](#) program in text
- 

---

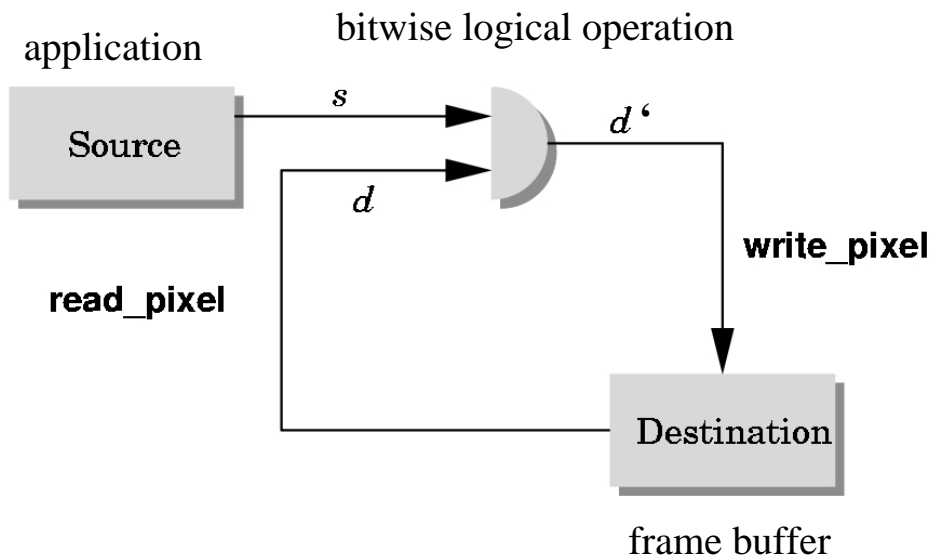
## Outline

- Learn to build more sophisticated interactive programs using
    - Picking
      - Select objects from the display
      - Three methods
    - [Rubberbanding](#)
      - Interactive drawing of lines and rectangles
    - Display Lists
      - Retained mode graphics
-

# Rubberbanding



# Writing Modes



---

## XOR write

- Usual (default) mode: source replaces destination ( $d' = s$ )
    - Cannot write temporary lines this way because we cannot recover what was “under” the line in a fast simple way
  - Exclusive OR mode (XOR) ( $d' = d \oplus s$ )
    - $x \oplus y \oplus x = y$
    - Hence, if we use XOR mode to write a line, we can draw it a second time and line is erased!
- 

---

## Rubberbanding

- Switch to XOR write mode
  - Draw object
    - For line can use first mouse click to fix one endpoint and then use motion callback to continuously update the second endpoint
    - Each time mouse is moved, redraw line which erases it and then draw line from fixed first position to to new second position
    - At end, switch back to normal drawing mode and draw line
    - Works for other objects: rectangles, circles
-

---

## After Class

- [HW1](#) solution in the portal
  - Read Chapter 3
  - Work on [HW2](#)
-