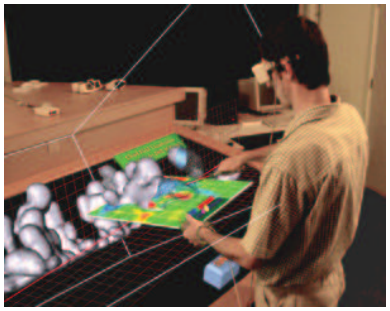
Computer Graphics (Input and Interaction)

Thilo Kielmann
Fall 2003
Vrije Universiteit, Amsterdam
kielmann@cs.vu.nl

http://www.cs.vu.nl/~graphics/

Adv: Scientific Visualization



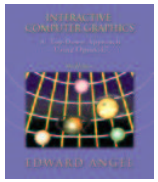
Course given by Michal Koutek
starts Tue, Oct 28, 13:45-15:30, at the ICWall

http://www.nat.vu.nl/~koutek/scivis/

Excursion to the CAVE: October 15, 10:30-12:30

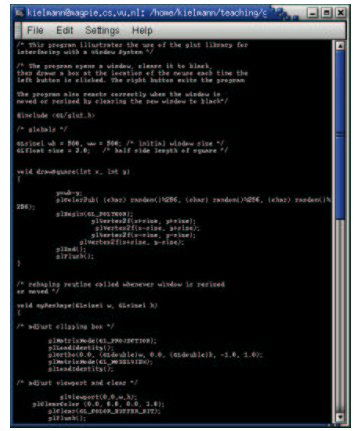
Sign up now!
Check WWW page for details (how to get there. . .)

A Word in advance...



read
background
material

connect the concepts



understand things REALLY

Outline for today

- GLUT and the window system
- Output modes and display lists
- Input devices
- Programming event-driven input
- Picking
- Animating interactive programs

Reminder: GLUT and the Window System

```
#include <GL/glut.h>
int main(int argc, char** argv){
  glutInit(&argc,argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(0,0);
  glutCreateWindow("Sierpinski Gasket");
  glutDisplayFunc(display); /* register display func. */

  myinit();       /* application-specific inits */
  glutMainLoop(); /* enter event loop */
  return 0;
}
```

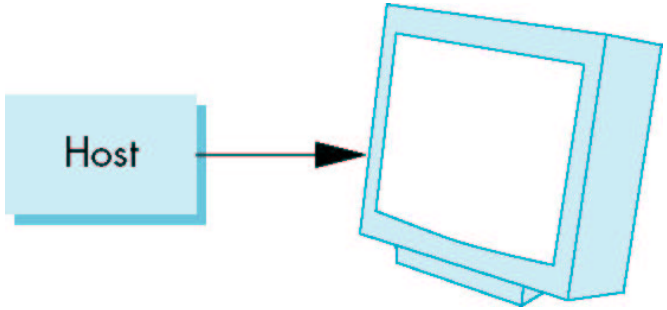

`myinit()` – Application-specific

```
void myinit(void)
{
  glClearColor(1.0, 1.0, 1.0, 1.0); /* white background */
                                /* ^----- opaque background */
  glColor3f(1.0, 0.0, 0.0); /* draw in red */

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0.0, 500.0, 0.0, 500.0);
  glMatrixMode(GL_MODELVIEW);
}
```


`display()` – Application-specific

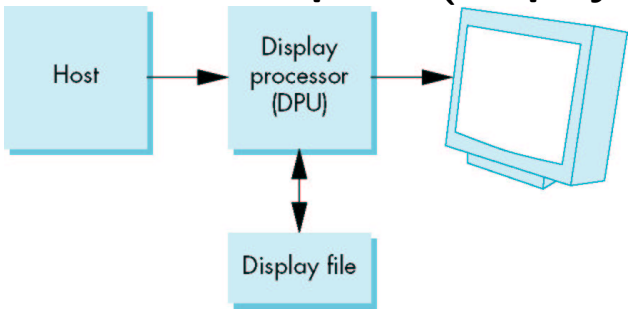
```
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    divide_triangle(v[0], v[1], v[2], n);
    glFlush();
}
```

Output: Immediate Mode Graphics



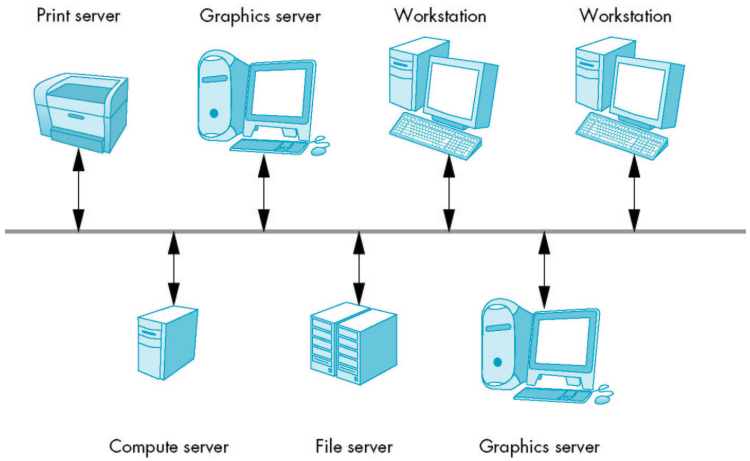
- Early: host has to write at 50–85 Hz to refresh screen
- Recent: frame buffer refreshing screen, host writes buffer

Retained Mode Graphics (Display Lists)

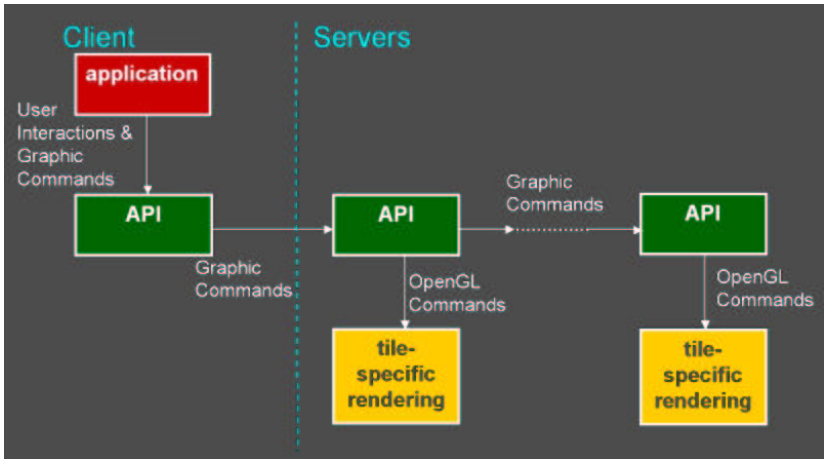


- Like JIT-compilation for graphics operations
- Reduce traffic between Host and DPU
- Speed up both Host and DPU operation

Display Lists and Remote Displays



Display Lists and The ICWall



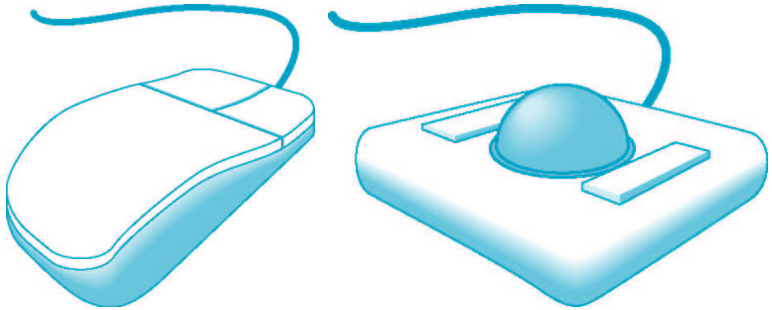
Display Lists: Summary for Today

- OpenGL provides display lists (described in Chapter 3)
- With a simple PC and simple programs, you will hardly notice any effect. . .
- . . . unless you use high-end hardware like the ICWall
- We will get back to display lists along with Chapter 9 (scene graphs)

Input and Interaction

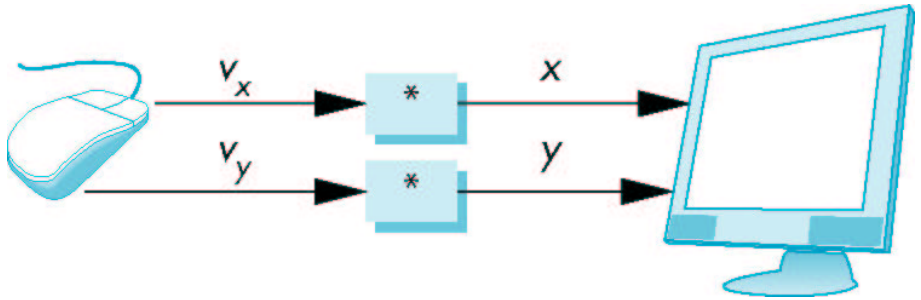
- Interaction with the user requires input devices
- physical input devices
- logical input devices
- input modes

Physical Input Devices



- pointing device (mouse, trackball)
- keyboard device (keyboard)

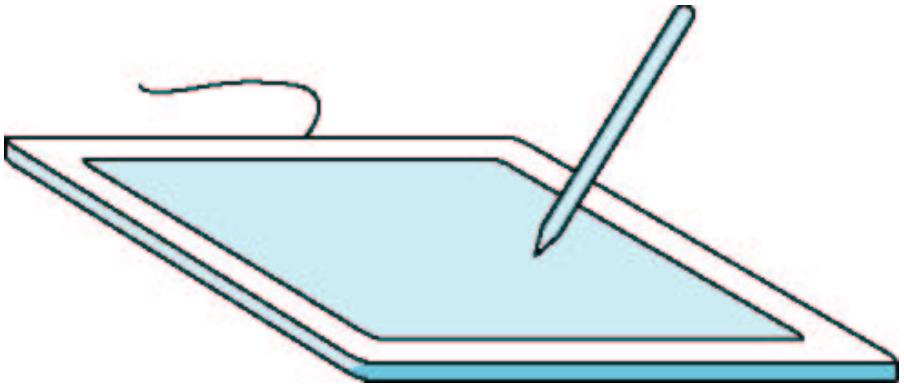
Relative Positioning



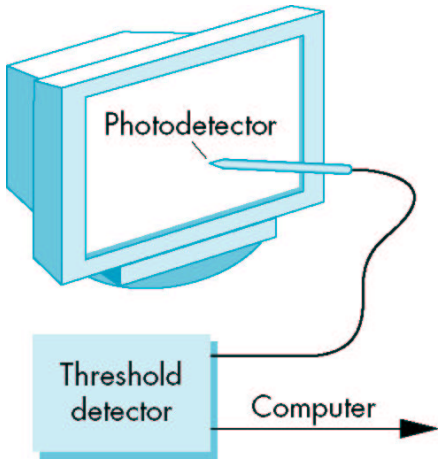
Device driver (or application) integrates mouse movement ticks (velocity) over time to compute new position.

Absolute Positioning

Data tablet:

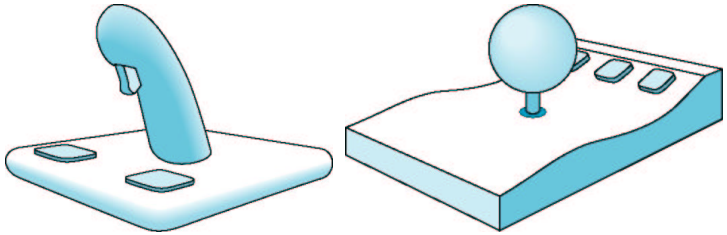


Lightpen (Absolute Positioning)



Problems: dark screen areas, ergonomics

Joystick and Spaceball



Movement interpreted as velocity/acceleration
provides mechanical resistance

Logical Devices

- String (keyboard)
- Locator (mouse)
- Pick (select item)
- Choice (menu)
- Dial ("analog" input: sliders)
- Stroke (array of locations, mouse movement)

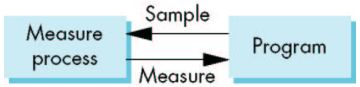
Measure and Trigger

- Measure:
  - ⋆ input data
  - ⋆ mouse position, typed string, . . .
- Trigger:
  - ⋆ signaling the application
  - ⋆ mouse button, return key, . . .

Input Modes

Request mode:

Sample mode:



Event mode:

Advantages of the Event Mode

- Program does not need to request/sample explicitly (no polling):
  - ⋆ No waste of CPU time while there is no input.
  - ⋆ No problem with location of polling statements in the code, and how often polling should occur. . .
- Multiple input devices (types of events) can be served, even without knowing how many are active.

Handling Events: Callback Functions

```
#include <GL/glut.h>
int main(int argc, char** argv){
  glutInit(&argc,argv);

  ...
  glutDisplayFunc(display);
  glutKeyboardFunc(keyboard);
  glutMouseFunc(mouse);
  ...

  glutMainLoop(); /* enter event loop */
  return 0;
}
```

Programming Event-driven Input

Mouse callback: (colors.c)

```
void mouse(int btn, int btn_state, int x, int y){
  if (btn==GLUT_LEFT_BUTTON && btn_state==GLUT_DOWN){
    state++;
    state = state%7;
  }
  if (btn==GLUT_MIDDLE_BUTTON && btn_state==GLUT_DOWN){
    state = 0;
  }
  if (btn==GLUT_RIGHT_BUTTON && btn_state==GLUT_DOWN){
    exit();
  }
  glutPostRedisplay();
}
```

Main program: `glutMouseFunc(mouse);`

Example Square Drawing

```
void drawSquare(int x, int y)   // (square.c)
{
  y=wh-y; /* translate window to user coordinates */
  glColor3ub( (char) random()%256,
              (char) random()%256, (char) random()%256);
  glBegin(GL_POLYGON);
  glVertex2f(x+size, y+size);
  glVertex2f(x-size, y+size);
  glVertex2f(x-size, y-size);
  glVertex2f(x+size, y-size);
  glEnd();
  glFlush();
}
```

Register: `glutMotionFunc(drawSquare);`

Mouse-related Callbacks

`glutMouseFunc`	button up/down
`glutMotionFunc`	movement with button pressed
`glutPassiveMotionFunc`	movement w/o button pressed

Reshape Events

```
void myReshape(GLsizei w, GLsizei h){    (square.c)
   /* adjust clipping box */
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0.0, (GLdouble)w, 0.0, (GLdouble)h, -1.0, 1.0);
   glMatrixMode(GL_MODELVIEW);
   /* adjust viewport and clear */
   glViewport(0,0,w,h);
   glClear(GL_COLOR_BUFFER_BIT);
   glFlush();
   /* set global size for use by drawing routine */
   ww = w;   wh = h;
}
```

Main program: `glutReshapeFunc(myReshape);`

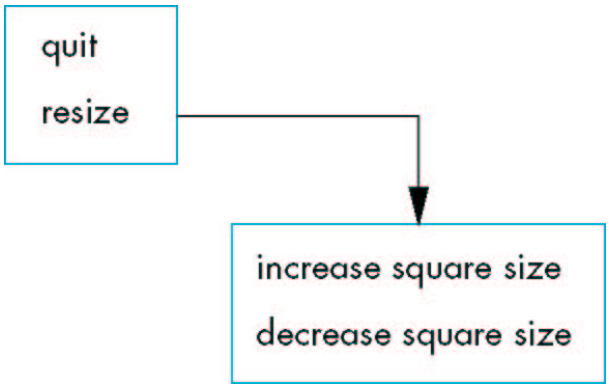
(Also called when window is displayed the first time.)

Keyboard Events

```
void keyboard(unsigned char key, int x, int y){
  if ( key == 'q' ||
       key == 'Q' ||
       key == 27  // ESC
     ) exit();
}

glutKeyboardFunc(keyboard);
```


Menus

Example for the **square** program:



Sample Menu

In the main program:

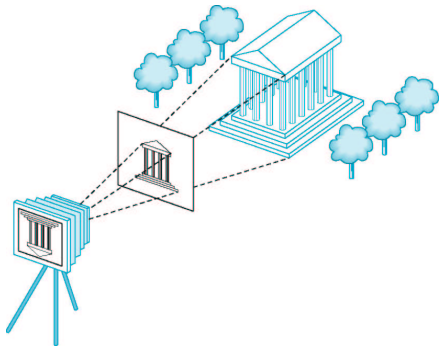
```
sub_menu = glutCreateMenu(size_menu);
glutAddMenuEntry("increase square size", 1);
glutAddMenuEntry("decrease square size", 2);
glutCreateMenu(top_menu);
glutAddMenuEntry("quit",1);
glutAddSubMenu("resize", sub_menu);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```


Sample Menu: Callbacks

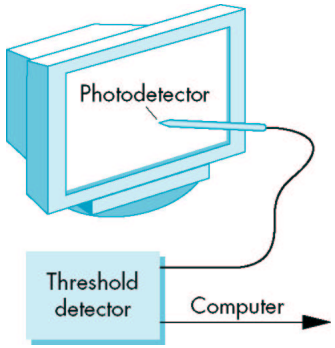
```
void size_menu(int id){
  if (id==1) size *= 2;
  if (id==2) size /= 2;
}

void top_menu(int id){
  if (id==1) exit();
}
```

Picking

Selecting a displayed object by the picking device.
Problem: getting back from 2D-screen coordinate to 3D-application coordinate:

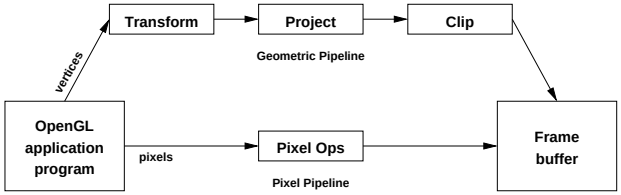


Picking with Lightpen (once upon a time)



Lightpen generates interrupt when light ray comes along.
⇒ Program knows the object currently being displayed.

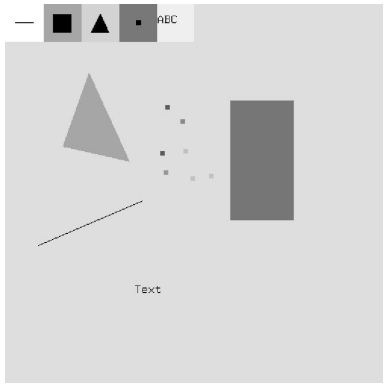
Picking with Rendering Pipelines



Problems:

- Operations are hard to reverse (esp. in hardware).
- The application abstracts from screen coordinates **on purpose**.
- The logical grouping of elementary operations (vertices) to application objects unknown to OpenGL.

Picking for Simple Applications



Application computes (2D) bounding boxes for its objects. (see Chapter 3.8 “A Simple Paint Program”)

Picking with OpenGL's Selection Mode

Idea:

1. Re-render the scene (walk through all objects).
   No graphical output is produced while doing so.
   (a) Application indicates when an object starts and ends.
   (b) OpenGL checks which objects are close to the (mouse) position.
2. Application interprets which object to select.

Render Mode vs. Selection Mode

Default:
`glRenderMode(GL_RENDER)`
Rendering output goes to frame buffer.

For picking:
`glRenderMode(GL_SELECT)`
Rendering output goes to (user-supplied) **select buffer**.

Select buffer stores objects that **hit** the picking position.

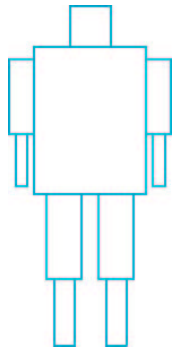
Identifying Objects: the Name Stack

```
void glInitNames();   // initialize name stack

void glPushName(GLuint name); // "names" are unsigned integers

void glPopName();

void glLoadName(GLuint name); // overwrite top element
                              // "here starts object 'name'"
```

With simple objects, only one stack element is enough.

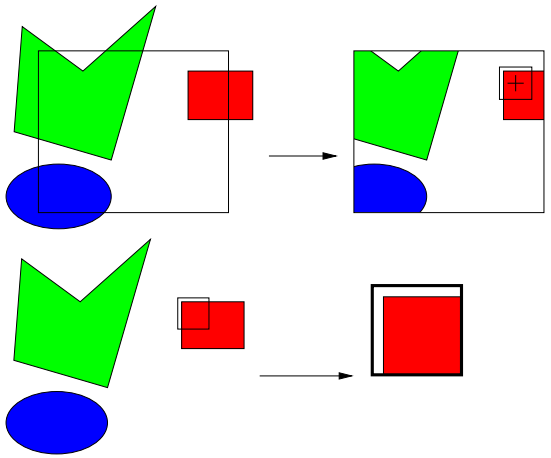
With complex objects (see Chapter 9), the whole stack trace needs to be captured – and the application needs to decide later, what is to be selected.

Example of a Complex Object



Application needs to decide whether the lower-left arm, or the whole robot should be selected.

Checking Proximity to Mouse by Clipping



Proximity Check

1. Use `gluPickMatrix` to restrict clipping region to small area around the mouse position.

2. Re-render in selection mode

   → Objects within the (small) clipping region generate a **hit** in the select buffer

Use $N \times N$ pixels area rather than single pixel to allow for human imprecision. . .

Putting the Pieces Together. . .

```
void display(){      // pick.c
  glClear(GL_COLOR_BUFFER_BIT);
  draw_objects(GL_RENDER);
  glFlush();
}

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

Picking in the Mouse Callback (1)

```
void mouse(int button, int state, int x, int y){
   GLuint selectBuf[SIZE];  GLint hits;  GLint viewport[4];
   if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
     glGetIntegerv (GL_VIEWPORT, viewport); // get current viewport
     glSelectBuffer (SIZE, selectBuf);
     glRenderMode(GL_SELECT);

     glInitNames(); glPushName(0); // init name stack

     glMatrixMode (GL_PROJECTION); glPushMatrix (); // save old state

     glLoadIdentity ();
     /*  create 5x5 pixel picking region near cursor location */
     gluPickMatrix ((GLdouble) x, (GLdouble) (viewport[3] - y),
                    5.0, 5.0, viewport);
     gluOrtho2D (-2.0, 2.0, -2.0, 2.0);  // same as without picking
```

Picking in the Mouse Callback (2)

```
    drawObjects(GL_SELECT);

    glMatrixMode (GL_PROJECTION);
    glPopMatrix ();  // restore old state
    glFlush ();

    hits = glRenderMode (GL_RENDER); // switch back and get
                                     // number of hits

    processHits (hits, selectBuf);   // still to be implemented
    glutPostRedisplay();             // ask GLUT for re-displaying
  }
}
```


The Select Buffer Data Structure

The select buffer is an array of **GLuint** values.
The data of all **hits** (returned by `glRenderMode`),
is stored consecutively.
Values for each hit:

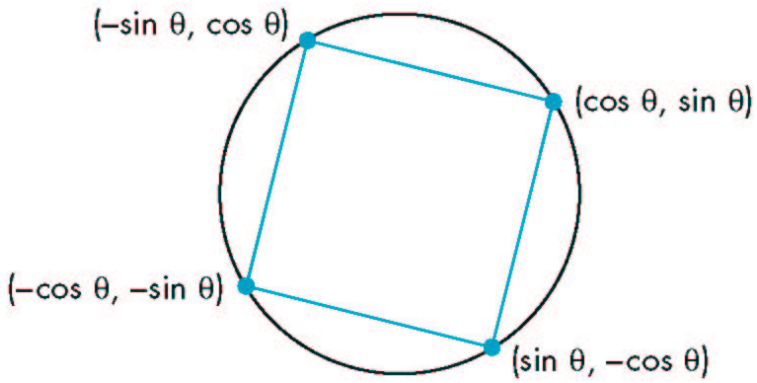
number of names n	(the depth of the name stack)
minimum depth	for all vertices of this hit
maximum depth	for all vertices of this hit
name 1	
name 2	
. . .	
name n	

Finally, `processHits()`

```
void processHits (GLint hits, GLuint buffer[]){
   unsigned int i, j;
   GLuint ii, jj, 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++;
      }
   }
}
```


Animating Interactive Programs

The rotating square:



Dislaying the Rotating Square

```
void display(){
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_POLYGON);
  thetar = theta*((2.0*3.14159)/360.0);
       /* convert degrees to radians */
  glVertex2f( cos(thetar), sin(thetar));
  glVertex2f(-sin(thetar), cos(thetar));
  glVertex2f(-cos(thetar),-sin(thetar));
  glVertex2f( sin(thetar),-cos(thetar));
  glEnd();
}
```


The Idle Function

```
double time;

void idle(){     // (single.c)
  double t = Wallclock();   // from glutil.h (VU-made)
  if ( t < time + 0.01 ) return;
  time = t;
  theta+=2;
  if ( theta>=360.0) theta -= 360.0;
  glutPostRedisplay();
}
```

Double Buffering

- screen image is refreshed 50-85 times per second
- drawing into the frame buffer is not an atomic action
  - ⋆ (and takes longer than 1/50 sec)
- the flickering we see is from partially drawn images
- solution: double buffering
  - ⋆ front buffer is used for display
  - ⋆ back buffer is used for drawing

Double Buffering (OpenGL)

- initialize with:
  `glutInitDisplay(GLUT_RGB | GLUT_DOUBLE)`
- add to `display()`:
  `glutSwapBuffers()`

(single_double) 

Double Buffering

- . . . isn't always the solution
- see chapter 3.9.3
- we will get back to buffering issues with Chapter 7

Direct Frame Buffer Access

- OpenGL allows to directly access the frame buffer
- Applications:
  - ⋆ popup-menus (restore the background without long re-rendering)
  - ⋆ rubberbanding (better done with raster operations)
- We get back to raster operations with Chapter 7.

Summary

- GLUT and the window system
- Output modes and display lists
- Input devices
- Programming event-driven input
- Picking
- Animating interactive programs

Summary

- Next week: affine transformations (simple math)