Computer Graphics
(Basic OpenGL, Input and Interaction)

Thilo Kielmann
Fall 2008
Vrije Universiteit, Amsterdam
kielmann@cs.vu.nl

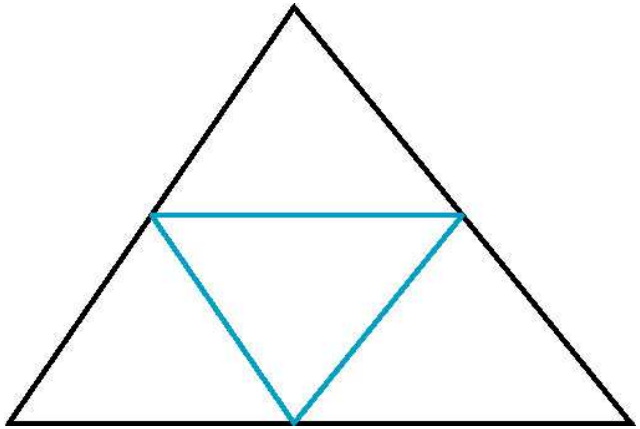
http://www.cs.vu.nl/~graphics/

Outline for today

- 3D graphics
- Input devices
- Programming event-driven input
- Picking
- Animating interactive programs

⇒ **3D graphics**

Sierpinski Gasket by Triangle Bisection



Drawing a Triangle

```
void triangle( point2 a, point2 b, point2 c){
  glBegin(GL_TRIANGLES);
     glVertex2fv(a);
     glVertex2fv(b);
     glVertex2fv(c);
  glEnd();
}
```

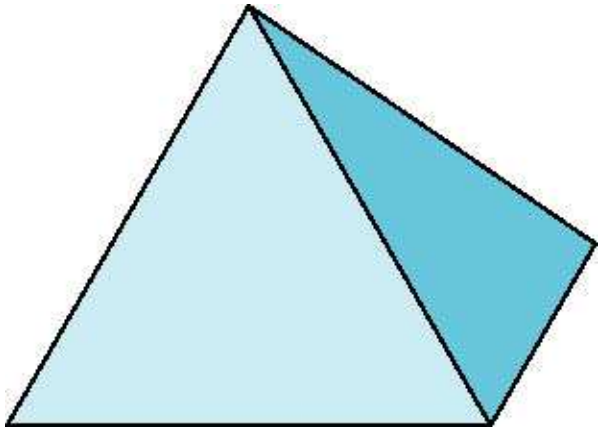
Dividing Triangles

```
void divide_triangle(point2 a, point2 b, point2 c, int m){
    point2 v0, v1, v2;
    int j;
    if(m>0) {
        for(j=0; j<2; j++) v0[j]=(a[j]+b[j])/2;
        for(j=0; j<2; j++) v1[j]=(a[j]+c[j])/2;
        for(j=0; j<2; j++) v2[j]=(b[j]+c[j])/2;
        divide_triangle(a, v0, v1, m-1);
        divide_triangle(c, v1, v2, m-1);
        divide_triangle(b, v2, v0, m-1);
    }
    else(triangle(a,b,c));
}
```


Display it

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

The 3-Dimensional Gasket



`myinit()` for 3D gasket

```
void myinit(void){

  glClearColor(1.0, 1.0, 1.0, 1.0); /* white background */

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(-500.0, 500.0, -500.0, 500.0, -500.0, 500.0);
  glMatrixMode(GL_MODELVIEW);
}
```

Drawing a 3D-Triangle

```
void triangle( point3 a, point3 b, point3 c){
  glBegin(GL_TRIANGLES);
    glVertex3fv(a);
    glVertex3fv(b);
    glVertex3fv(c);
 glEnd();
}
```

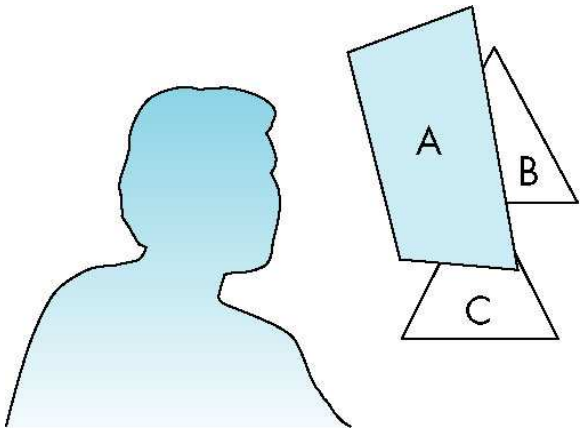

Dividing 3D-Triangles

```
void divide_triangle(point3 a, point3 b, point3 c, int m){
  point3 v0, v1, v2;
  int j;
  if(m>0){
    for(j=0; j<3; j++) v0[j]=(a[j]+b[j])/2;
    for(j=0; j<3; j++) v1[j]=(a[j]+c[j])/2;
    for(j=0; j<3; j++) v2[j]=(b[j]+c[j])/2;
    divide_triangle(a, v0, v1, m-1);
    divide_triangle(c, v1, v2, m-1);
    divide_triangle(b, v2, v0, m-1);
  }
  else(triangle(a,b,c));
}
```

`display()` for 3D with triangles

```
void display(void){  /* be n the recursion level */
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0,0.0,0.0);
  divide_triangle(v[0],v[2],v[3], n);
  glColor3f(0.0,1.0,0.0);
  divide_triangle(v[0],v[1],v[2], n);
  glColor3f(0.0,0.0,1.0);
  divide_triangle(v[1],v[2],v[3], n);
  glColor3f(0.0,0.0,0.0);
  divide_triangle(v[0],v[1],v[3], n);
  glFlush();
}
```

Hidden Surface Removal



Let's add Hidden-Surface Removal

```
int main(int argc, char **argv) {
  if ( argc < 2 ) { printf("synopsis: %s <recursion depth>\n",argv[0]); }
  else{
    n=atoi(argv[1]);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH );
    glutInitWindowSize(500, 500);
    glutCreateWindow("3D Gasket, Triangles, hidden-surface removal");
    glutDisplayFunc(display);
    glEnable(GL_DEPTH_TEST);
    myinit();
    glutMainLoop();
  }
  return 0;
}
```


. . . and don't forget:

```
void display(void){
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  ...
}
```

with

without

Outline for today

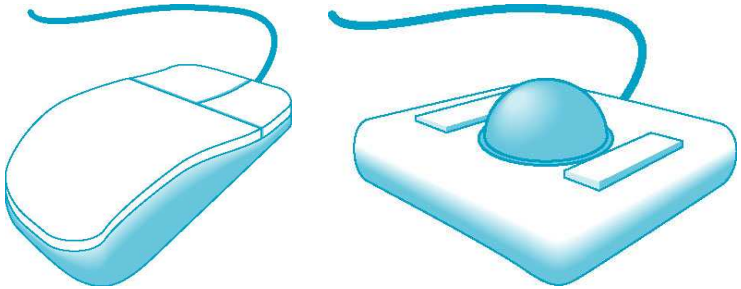
- 3D graphics
- Input devices
- Programming event-driven input
- Picking
- Animating interactive programs

⇒ **Input devices**

Input and Interaction

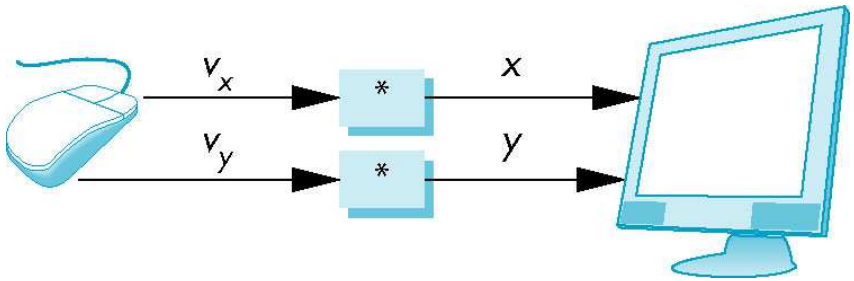
- Interaction with the user requires input devices
- physical input devices
- input modes

Physical Input Devices



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

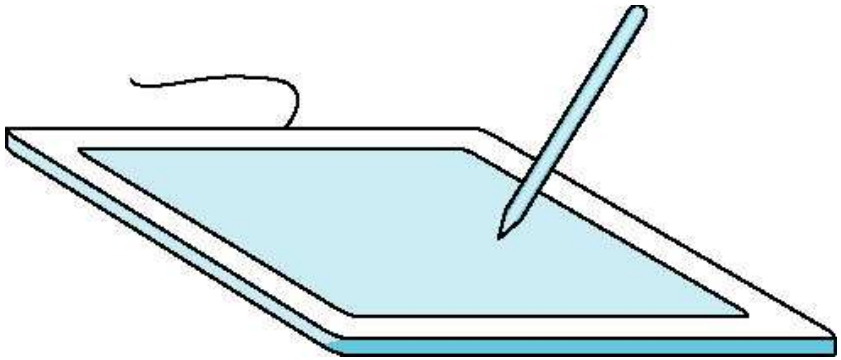
Relative Positioning



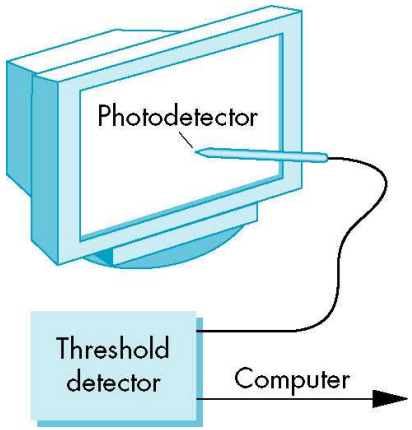
Device driver (or application) integrates mouse movement ticks (velocity) over time to compute new position.

Absolute Positioning

Data tablet:

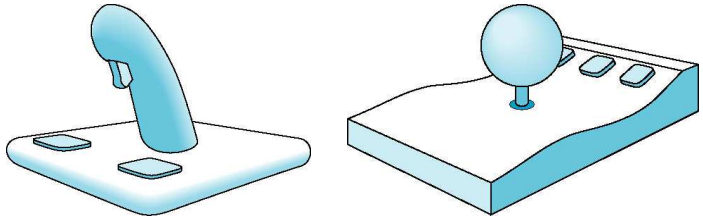


Lightpen (Absolute Positioning)



Problems: dark screen areas, ergonomics

Joystick and Spaceball

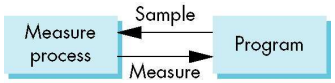


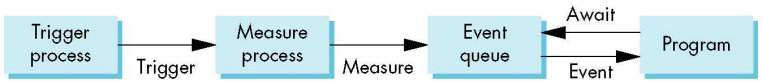
Movement interpreted as velocity/acceleration
provides mechanical resistance

Measure and Trigger

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

Input Modes





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 (and which) are active.

Outline for today

- 3D graphics
- Input devices
- Programming event-driven input
- Picking
- Animating interactive programs

⇒ **Programming event-driven input**

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(0);
  }
  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 without 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;
}
```

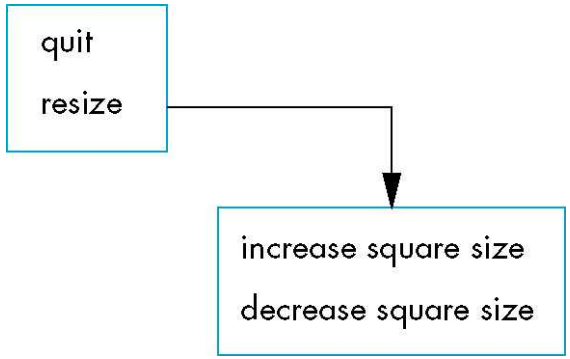
Main program: `glutReshapeFunc(myReshape);`
(Also called when window is displayed the first time.)

Keyboard Events

```
void keyboard(unsigned char key, int x, int y){
  if ( key == 'q' ||
       key == 'Q' ||
       key == 27  // ESC
     ) exit(0);
}

glutKeyboardFunc(keyboard);
```


Menus

Example for the **square** program:



Sample Menu

In the main program:

```
sub_menu = glutCreateMenu(size_menu);          // creates and selects ‘‘current’’ menu
glutAddMenuEntry("increase square size", 1);   // add to current menu
glutAddMenuEntry("decrease square size", 2);
glutCreateMenu(top_menu);                      // create new ‘‘current’’ menu
glutAddMenuEntry("quit",1);
glutAddSubMenu("resize", sub_menu);            // add sub menu to menu
glutAttachMenu(GLUT_RIGHT_BUTTON);             // attach current menu to mouse button
```

```
void size_menu(int id){
   if (id==1) size *= 2;
   if (id==2) size /= 2;
}
void top_menu(int id){
   if (id==1) exit(0);
}
```

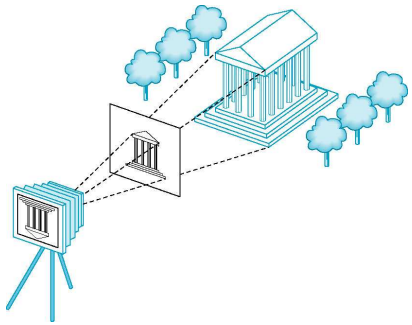

Outline for today

- 3D graphics
- Input devices
- Programming event-driven input
- Picking
- Animating interactive programs

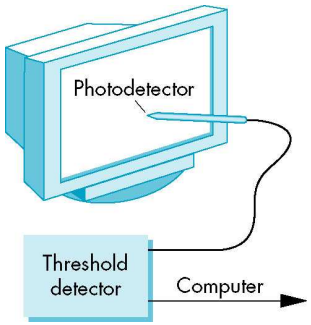
⇒ **Picking**

Picking

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

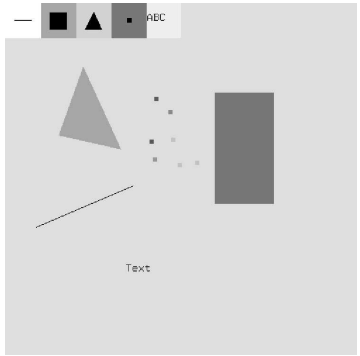


Picking with Lightpen (once upon a time)



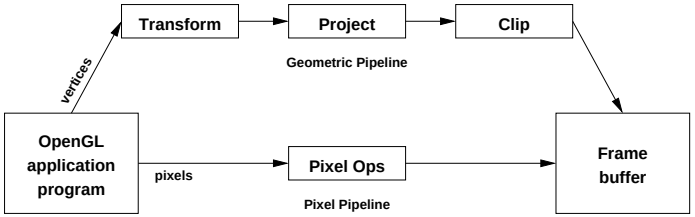
Lightpen generates interrupt when light ray comes along. ⇒ Program knows the object currently being displayed.

Picking for Simple Applications (2D)



Application computes (2D) bounding boxes for its objects.

Picking 3D Objects with the 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 is unknown to OpenGL.

Picking with OpenGL's Selection Mode

Idea:

1. When user clicks into the window, re-render the scene (walk through all objects).
   No graphical output is produced while doing so. (special "selection" mode)

(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.

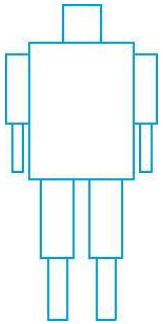
Identifying Objects: the Name Stack

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

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

void glPopName();

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

With simple objects, only one stack element is enough.

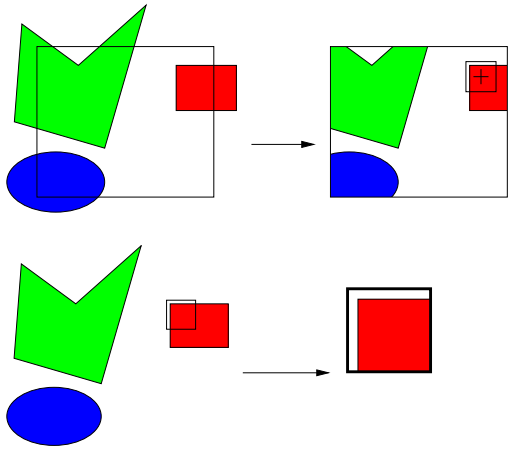
With complex objects (see Chapter 10), the whole stack trace needs to be captured – and the application needs to decide later, what is to be selected.

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.
   (We apply a "magnifying glass" to the viewport.)

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++;
      }
   }
}
```

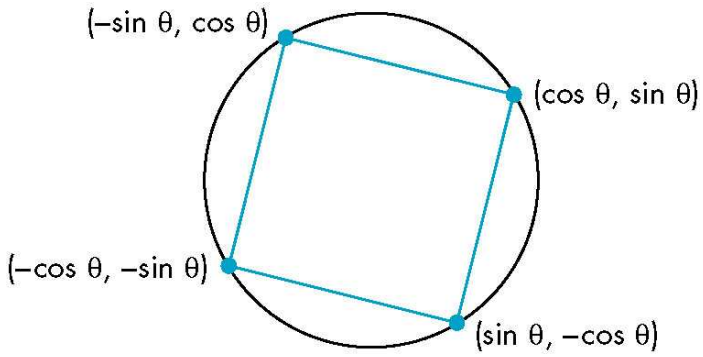

Outline for today

- 3D graphics
- Input devices
- Programming event-driven input
- Picking
- Animating interactive programs

⇒ **Animating interactive programs**

Animating Interactive Programs

A rotating square:



Displaying 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();
}
```

This is the “poor man’s solution”: use glRotate() instead! (once we will have learned about it)

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) 

Summary

What to remember:

- Hidden surface removal
- Programming event-driven input
- Picking
- Double buffering

Next lecture:

- Affine transformations (simple math)