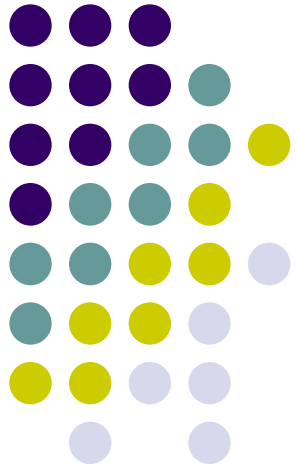
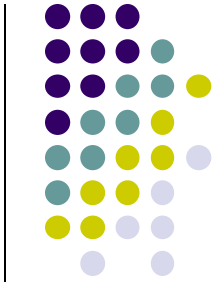
Introduction to OpenGL Programming

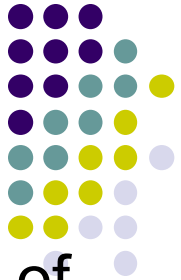


Frame Buffer



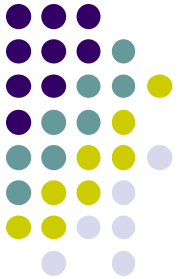
- The computer program stores the image in a two-dimensional array in RAM of pixel values (called a ***frame buffer***). *The display hardware produces the image line-by-line (called raster lines).*
- The frame buffer is not a device. It is simply a chunk of RAM memory that has been allocated for this purpose.
- A program modifies the display by writing into the frame buffer, and thus instantly altering the image that is displayed.
- OpenGL is used to write such programs and send their commands to the graphics accelerators (*graphic cards*). The accelerator rasterizes the results of the commands and sends them to the frame buffer.

Graphic cards



- Graphic card relieves the computer's CPU from much of the mundane repetitive effort involved in maintaining the frame buffer.
- A typical graphic card will provide assistance for a number of operations (*graphical pipeline*) including the following:
  - **Transformations**
  - **Clipping**
  - **Projection**
  - **Shading and Coloring**
  - **Texturing**
  - **Hidden – surface elimination**

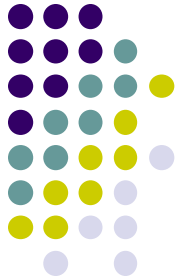
SGI and GL



- Silicon Graphics (SGI) revolutionized the graphics workstation by implementing the pipeline in hardware (1982)
- To access the system, application programmers used a library called GL
- With GL, it was relatively simple to program three dimensional interactive applications

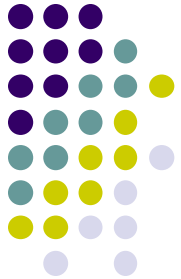
What is OpenGL

http://www.opengl.org/



- OpenGL is a software API to graphics hardware.
  - designed as a steamlined hardware-independent interface to be implemented on many different hardware platforms
  - Intuitive, procedural interface with C/C++ binding
  - No windowing commands !
  - No high-level commands for describing models of three-dimensional objects
  - The OpenGL Utility Library (GLU) provides many of the modeling features, such as quadric surfaces and NURBS curves and surfaces

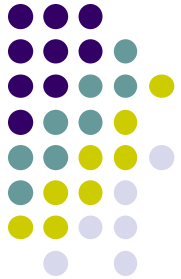
OpenGL - Basic Attributes



The success of GL lead to OpenGL (1992), a platform-independent API that was

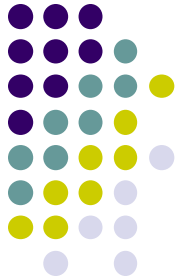
- Easy to use
- Close enough to the hardware to get excellent performance
- Focus on rendering
- Omitted windowing and input to avoid window system dependencies

OpenGL Libraries



- OpenGL core library
  - OpenGL32 on Windows
  - GL on most unix/linux systems (libGL.a)
- OpenGL Utility Library (GLU)
  - Provides functionality in OpenGL core but avoids having to rewrite code
- Links with window system
  - GLX for X window systems, WGL for Windows
  - Cross-platform GUI libraries: GLUT, SDL, FLTK, QT, …

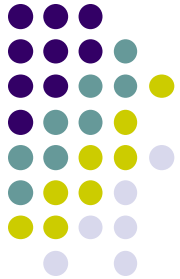
Windowing with OpenGL



- OpenGL is independent of any specific window system
- OpenGL can be used with different window systems
  - X windows (GLX)
  - MFC
  - …
- GLUT provide a portable API for creating window and interacting with I/O devices

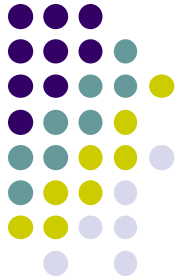
GLUT

http://www.opengl.org/documentation/specs/glut/spec3/spec3.html



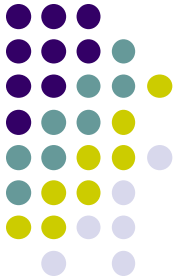
- OpenGL Utility Toolkit (GLUT)
  - Provides functionality common to all window systems
    - Open a window
    - Get input from mouse and keyboard
    - Menus
    - Event-driven
  - Code is portable but GLUT lacks the functionality of a good toolkit for a specific platform
    - No slide bars, buttons, …

OpenGL as a state machine



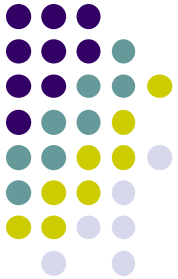
- GL State Variables- can be set and queried by OpenGL. Remains unchanged until the next change.
  - Projection and viewing matrix
  - Color and material properties
  - Lights and shading
  - Line and polygon drawing modes
  - ...
- OpenGL functions are of two types
  - Primitive generating
    - Can cause output if primitive is visible
    - How vertices are processed and appearance of primitive are controlled by the state
  - State changing
    - Transformation functions
    - Attribute functions

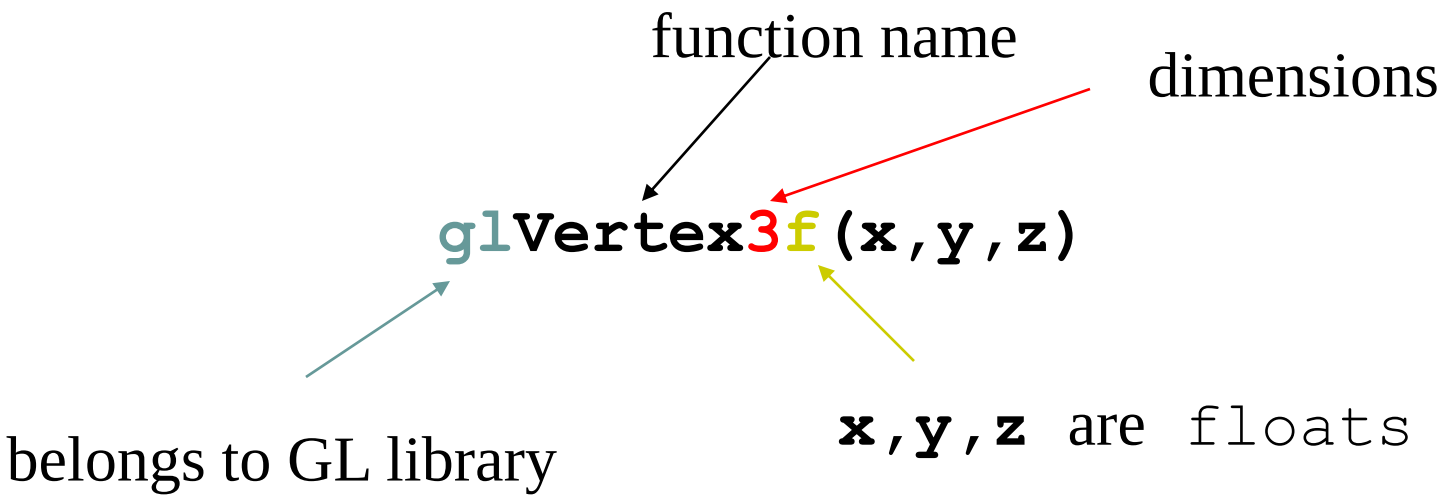
OpenGL Syntax



- Functions have prefix **gl** and initial capital letters for each word
  - **glClearColor(), glEnable(), glPushMatrix()** …
- **glu** for **GLU** functions
  - **gluLookAt(), gluPerspective()** …
- Constants begin with **GL_**, use all capital letters
  - **GL_COLOR_BUFFER_BIT, GL_PROJECTION, GL_MODELVIEW** …
- Extra letters in some commands indicate the number and type of variables
  - **glColor3f(), glVertex3f()** …
- OpenGL data types
  - **GLfloat, GLdouble, GLint, GLenum,** …**, for compatibility**
- Underlying storage mode is the same
- Easy to create overloaded functions in C++ but issue is efficiency

OpenGL function format

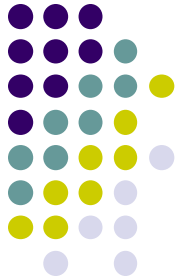





```
glVertex3fv(p)
```

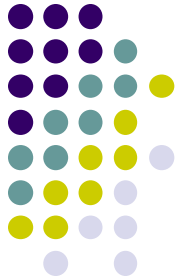
p is a pointer to an array

OpenGL #defines



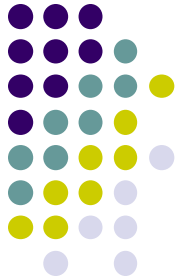
- Most constants are defined in the include files **`gl.h`**, **`glu.h`** and **`glut.h`**
  - Note **`#include <GL/glut.h>`** should automatically include the others
  - Examples
  - **`glBegin(GL_POLYGON)`**
  - **`glClear(GL_COLOR_BUFFER_BIT)`**
- include files also define OpenGL data types: **`GLfloat`**, **`GLdouble`**,….

GLUT



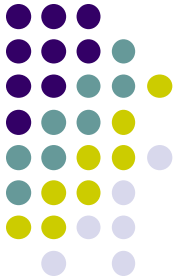
- Developed by Mark Kilgard
- Hides the complexities of differing window system APIs
  - Default user interface for class projects
- Glut routines have prefix **`glut`**
  - **`glutCreateWindow()`** …
- Has very limited GUI interface
- Glui is the C++ extension of glut

Glut Routines



- **Initialization:** `glutInit()` processes (and removes) commandline arguments that may be of interest to glut and the window system and does general initialization of Glut and OpenGL
  - Must be called before any other glut routines
- **Display Mode**: The next procedure, `glutInitDisplayMode(),` performs initializations informing OpenGL how to set up the frame buffer.
  - Display Mode — Meaning
  - **GLUT_RGB** — Use RGB colors
  - **GLUT_RGBA** — Use RGB plus alpha (for transparency)
  - **GLUT_INDEX** — Use indexed colors (not recommended)
  - **GLUT_DOUBLE** — Use double buffering (recommended)
  - **GLUT_SINGLE** — Use single buffering (not recommended)
  - **GLUT_DEPTH** — Use depthbuffer (for hidden surface removal.)

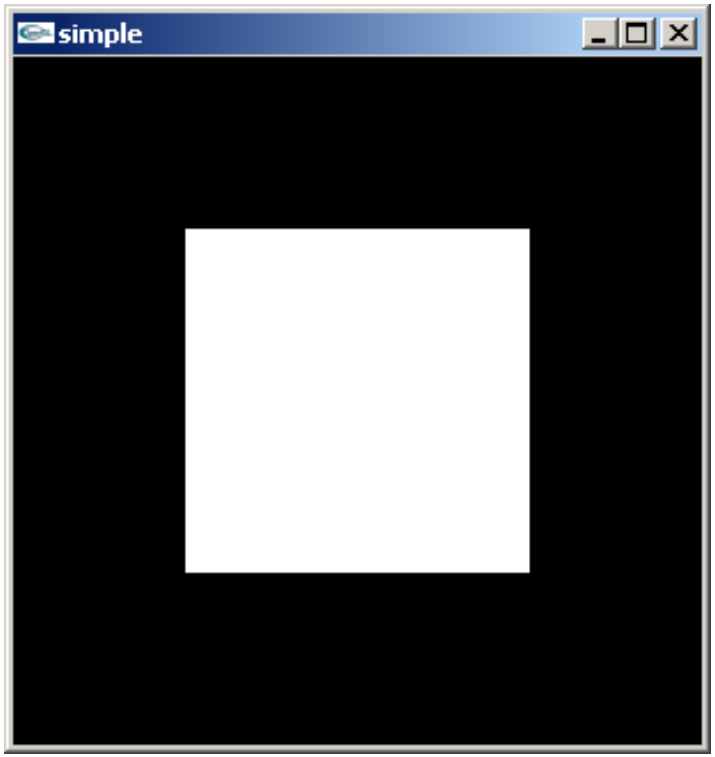
Glut Routines



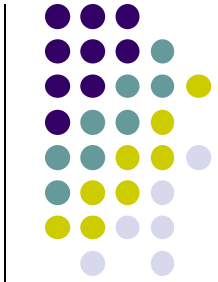
- Window Setup
  - `glutInitWindowSize(int width, int height)`
  - `glutInitWindowPosition(int x, int y)`
  - `glutCreateWindow(char* title)`

A Simple Program

Generate a square on a solid background



simple.c

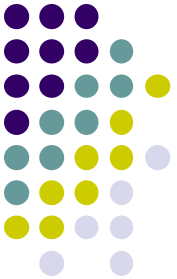


```
#include <GL/glut.h>
void init()
{
        glClearColor (0.0, 0.0, 0.0, 1.0);

        glColor3f(1.0, 1.0, 1.0);

        glMatrixMode (GL_PROJECTION);
        glLoadIdentity ();
        glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
}
void mydisplay(){
      glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POLYGON);
                glVertex2f(-0.5, -0.5);
                glVertex2f(-0.5, 0.5);
                glVertex2f(0.5, 0.5);
                glVertex2f(0.5, -0.5);
        glEnd();
        glFlush();
}
```

Simple.c (continue)



```
int main(int argc, char** argv){
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE |
     GLUT_RGB);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(0,0);
  glutCreateWindow("simple");
  glutDisplayFunc(mydisplay);
  init();
  glutMainLoop();
}
```

Closer Look at the main()

```
#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("simple");
   glutDisplayFunc(mydisplay);

   init();

   glutMainLoop();
}
```

includes `gl.h`

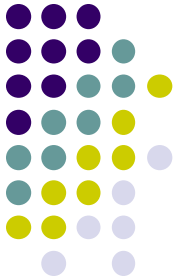
define window properties

display callback

set OpenGL state

enter event loop

Closer Look at the init()



```
void init()
{
   glClearColor (0.0, 0.0, 0.0, 1.0);

   glColor3f(1.0, 1.0, 1.0);

   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
}
```

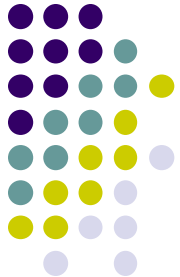
black clear color

opaque window

fill/draw with white

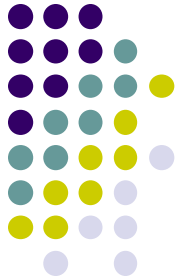
viewing volume

Event Handling



- Virtually all interactive graphics programs are event driven
- GLUT uses callbacks to handle events
  - Windows system invokes a particular procedure when an event of particular type occurs.
  - MOST IMPORTANT: display event
    - Signaled when window first displays and whenever portions of the window reveals from blocking window
    - **`glutDisplayFunc(void (*func)(void))`** registers the display callback function
- Running the program: **`glutMainLoop()`**
  - Main event loop. Never exit()

More Callbacks

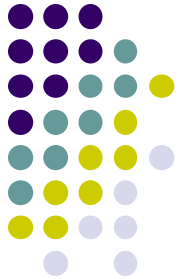


- `glutReshapeFunc(void (*func)(int w, int h))` indicates what action should be taken when the window is resized.
- `glutKeyboardFunc(void (*func)(unsigned char key, int x, int y))` and `glutMouseFunc(void (*func)(int button, int state, int x, int y))` allow you to link a keyboard key or a mouse button with a routine that's invoked when the key or mouse button is pressed or released.
- `glutMotionFunc(void (*func)(int x, int y))` registers a routine to call back when the mouse is moved while a mouse button is also pressed.
- `glutIdleFunc(void (*func)(void))` registers a function that's to be executed if no other events are pending - for example, when the event loop would otherwise be idle

OpenGL Drawing

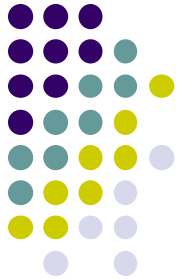
- OpenGL coordinate system has different origin from the window system
  - **GLUT** uses the convention that *the origin is in the upper left corner and coordinates are given as integers*. This makes sense for Glut, because its principal job is to communicate with the window system, and most window systems (X-windows, for example) use this convention.
  - **OpenGL** uses the convention that coordinates are (generally) floating point values and the origin is in the lower left corner. Recalling the OpenGL goal is to provide us with an idealized drawing surface this convention is mathematically more elegant.

Clear the Window



- `glClear(GL_COLOR_BUFFER_BIT)`
  - clears the frame buffer by overwriting it with the background color.
  - Background color is a state set by `glClearColor(GLfloat r, GLfloat g, GLfloat b, GLfloat a)` in the `init()`.

Drawing Attributes: Color



- `glColor3f(GLfloat r, GLfloat g, GLfloat b)` sets the drawing color
  - `glColor3d(), glColor3ui()` can also be used
  - Remember OpenGL is a state machine
  - Once set, the attribute applies to all subsequent defined objects until it is set to some other value
  - `glColor3fv()` takes a flat array as input
- There are more drawing attributes than color
  - Point size: `glPointSize()`
  - Line width: `glLinewidth()`
  - Dash or dotted line: `glLineStipple()`
  - Polygon pattern: `glPolygonStipple()`
  - …

Drawing Commands

- Simple Objects `glRectf()`
- Complex Objects
  - Use construct `glBegin(mode)` and `glEnd()` and a list of vertices in between
    -
```
glBegin(mode)
    glVertex(v0);
    glVertex(v1);
    ...
glEnd();
```
- Some other commands can also be used between `glBegin()` and `glEnd()`, e.g. `glColor3f()`.
- Example