Computer Graphics for Visualization

Why Computer Graphics?

- Visualization = data transformation + rendering
- Computer graphics forms the foundation of visualization
- Graphics: process of generating images or pictures using computers
  - 3D Graphics
  - Interactive Graphics

Graphics Challenge!

- How to model visual world
  - A variety of objects
    Different shapes, sizes, and dimensions
  - A variety of colors
    Visible light spectra (violet to red)
  - A variety of materials
    Shine, dull, glow
  - A variety of lighting
    Ambient, diffuse, specular

Computer Graphics APIs

- OpenGL
- DirectX (or Direct3D)
- Mesa
- GKS (graphical kernel system)
- PHIGS (Programmer’s hierarchical interactive graphics system)

Topics in OpenGL

- OpenGL basics
- Drawing geometric objects
- Viewing
- Color
- Lighting
- Some special topics

What Is OpenGL?

- A standard, hardware-independent interface to graphics hardware
  - Introduced in 1992 by SGI
  - Most widely used 3D graphics API
  - Portable across a wide array of platforms
  - Well documented

- Current Version: OpenGL 4.0
  - Older versions: 1.'s, 2.'s and 3.'s

- No commands for window management
  - Does not create window
  - Does not take user input (such as mouse click)

What Is OpenGL?

- Provides a powerful but primitive set of rendering commands
  - points, lines and polygons
- No high-level rendering commands
  - Ultimate control over modeling 3D objects
  - Assembler language of computer graphics
- Foundation for high-performance graphics
  - Many APIs built on the top of OpenGL

    VTK, VMD, Java3D, VRML, Open Inventor

What Is OpenGL?

- GL routine has a prefix `gl`
  - **glColor()**
- Header file for GL-library calls
  - `#include <GL/gl.h>`
- Software information and download
  - http://www.opengl.org

OpenGL Command Syntax

- OpenGL commands
  - ➢ prefix **gl** and initial capital letters for each word making up the command line

    glClearColor()

    glVertex()

- OpenGL defined constants
  - ➢ Begin with GL_, use all capital letters, and use underscore to separate words

    GL_COLOR_BUFFER_BIT

    GL_TRIANGLES

OpenGL Command Syntax

- Suffixes in commands
  - **void glVertex{234}{sifd)[v]**(TYPE *coords*);

    2 or **3** or **4** means the number of arguments to be given

    s or **i** or **f** or **d** means data type

    v means a pointer to a vector or array of three values

  - **glVertex3f(2.0, 4.0, 1.0);**

    Three floating-point numbers for three arguments

  - GLfloat dvect[3]= (2.0, 4.0, 1.0);

    glVertex3fv(dvect);

    Representation of three arguments by a vector *dvect*

OpenGL - Related Libraries

- Libraries for extending different window and operating systems to support OpenGL

- Different OpenGL extensions
  - GLX: X Window
  - AGL: Apple Mac
  - PGL: IBM OS/2 Warp
  - WGL: Microsoft Windows NT and Windows 95

OpenGL - Related Libraries

- OpenGL Utility Library: GLU
  - Routines for special tasks

    Matrices for viewing orientations and projections

    Polygon tessellation

    Rendering surfaces

  - `prefix glu`
  - `#include <GL/glu.h>`

OpenGL - Related Libraries

- OpenGL Utility Toolkit: GLUT
  - Window-system independent
  - `prefix glut`
  - `#include <GL/glut.h>`

- Window management
  - Creating window and handling input events

- Modeling 3D objects
  - High level drawing commands built on top of OpenGL

Window Management

- **Initializing and Creating a Window**

- void **glutInit**(int **argc*, char ***argv*);
  - Initializes the GLUT
  - Appears before any other GLUT routine

- void **glutInitDisplay**(unsigned int *mode*);
  - Specifies a display mode (buffer or color mode)
  - A double-buffered and RGBA color mode window:

    glutInitDisplay(GLUT_DOUBLE | GLUT_RGBA);

Window Management

- void **glutInitWindowPosition**(int *x*, int *y*);
  - Specifies the location of the upper-left corner of the window
- void **glutInitWindowSize**(int *width*, int *height*);
  - Specifies window's size in pixels
- void **glutCreateWindow**(char **name*);
  - Opens window with previously set characteristics (display mode, size, etc)
  - Window is not displayed until **glutMainLoop ( )** is called

Window Management

- **Handling window and input events**
  - Callback functions to specify specific events, e.g, mouse click
  - Register these functions before entering the main loop

- void **glutDisplayFunc**(void (*func)(void));
  - Specifies the function that is called whenever the contents of the window need to be redrawn

Window Management

- void **glutMouseFunc**(void (**func*)(int *button*, int *state*, int *x*, int *y*));
  - ➢ Specifies the function, *func*, that's called when a mouse *button* is pressed or released

- void **glutMotionFunc**(void(**func*)(int *x*, int *y*));
  - ➢ Specifies the function, *func*, that's called when the *mouse pointer* moves with the mouse button being pressed

Window Management

- void **glutKeyboardFunc**(void(**func*)(unsigned int *key*, int *x*, int *y*));
  - Specifies the function, *func*, that's called when a *key* is pressed
- void **glutReshapeFunc**(void(**func*)(int *width*, int *height*));
  - Specifies the function that's called whenever the window is resized or moved
  - *Func* reestablishes the rectangular region as a new rendering canvas and adjust coordinate system

Window Management

- **Managing a background process**

- void **glutIdleFunc**(void (**func*)(void));
  - ➢ Specifies the function, *func*, to be executed if no other events are pending
  - ➢ If NULL is passed in, execution of the function is disabled

- void **glutPostRedisplayFunc**(void);
  - ➢ Marks the current window as needing to be redrawn
  - ➢ At the next opportunity, the callback function registered by **glutDisplayFunc()** is called

Window Management

- **Running the program**
  - GLUT program enters "an event-processing loop"

- void **glutMainLoop**(void);
  - Enters the GLUT processing loop, never to return
  - Registered callback functions will be called when the corresponding events occur

Drawing 3D Objects with GLUT

- GLUT has many high-level drawing routines

- Two flavors of model
  - Wireframe without surface normal

    void **glutWireCube**(Gldouble *size*);

    void **glutWireSphere**(Gldouble *radius*, Glint *slices*, Glint *stacks*);

  - Solid with shading and surface normal

    void **glutSolidCube**(Gldouble *size*);

    void **glutSolidSphere**(Gldouble *radius*, Glint *slices*, Glint *stacks*);

- Other examples

  torus, icosahedron, octahedron, cone, teapot

Important OpenGL Operations

- Clearing the window
  - ➢ Clear the color buffer filled with the last picture before drawing

    ```
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    ```

- Specifying a color
  - ➢ Set the color to red (RGB mode) before any drawing

    ```
    glColor3f(1.0, 0.0, 0.0);
    ```

- Forcing completion of drawing
  - ➢ Force previous commands to begin execution

    ```
    void glFlush(void);
    ```

  - ➢ Particularly useful in client-server system

Example 1: OpenGL Program

- **Draws a red sphere in a white window**

```
#include <GL/glut.h>

void display (void)
{
   glClearColor(1.0, 1.0, 1.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 0.0, 0.0);
   glutSolidSphere(0.4, 50, 40);
   glFlush();
}
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(500, 500);
   glutInitWindowPosition(100, 100);
   glutCreateWindow(”A red sphere in a white window");
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```

Simplifed Example 1

- **Using default settings for window and drawing color**

```
#include <GL/glut.h>

void display (void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glutSolidSphere(0.4, 50, 40);
   glFlush();
}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutCreateWindow(”A white sphere in the black window");
   glutDisplayFunc(display);
   glutMainLoop();
}
```

Example 2: Keyboard Input

- Draws a different object when different ***key*** is pressed

t	for triangle
c	for circle
s	for square

- Drawing view remains unchanged with change in window size

Menus

- GLUT provides one important widget: menus
  - Pop-up menus
- Three steps in defining a menu
  - Decide what entries are in the menu
  - Tie specific actions to the rows
  - Tie each menu to a mouse button
- Relevant functions
  - glutCreateMenu()
  - glutSetMenu()
  - glutAddMenuEntry
  - glutAttachMenu()
  - glutAddSubMenu

SubWindows and Multiple Windows

- Create a top-level window *name* and returns an identifier for it
  - glutCreateWindow(name)

- When a window is created, it becomes the current window, which can be changed by
  - glutSetWindow(id)
  - Each window has its own properties, called context.

- Creates a subwindow of *parent* and returns its id. The subwindow has its origin at (*x*,*y*) and has size *width by height* in pixels
  - glutCreateSubWindow(parent, x, y, width, height)
  - glutPostWindowRedisplay(winid)

OpenGL as a State Machine

- Can be put into various states (modes) that remain in effect until they are changed
  - Current color
  - Current viewing and projection transformations
  - Position and characteristics of light sources

- State variables are queryable

  glGetFloatv(GL_CURRENT_COLOR, *params*);

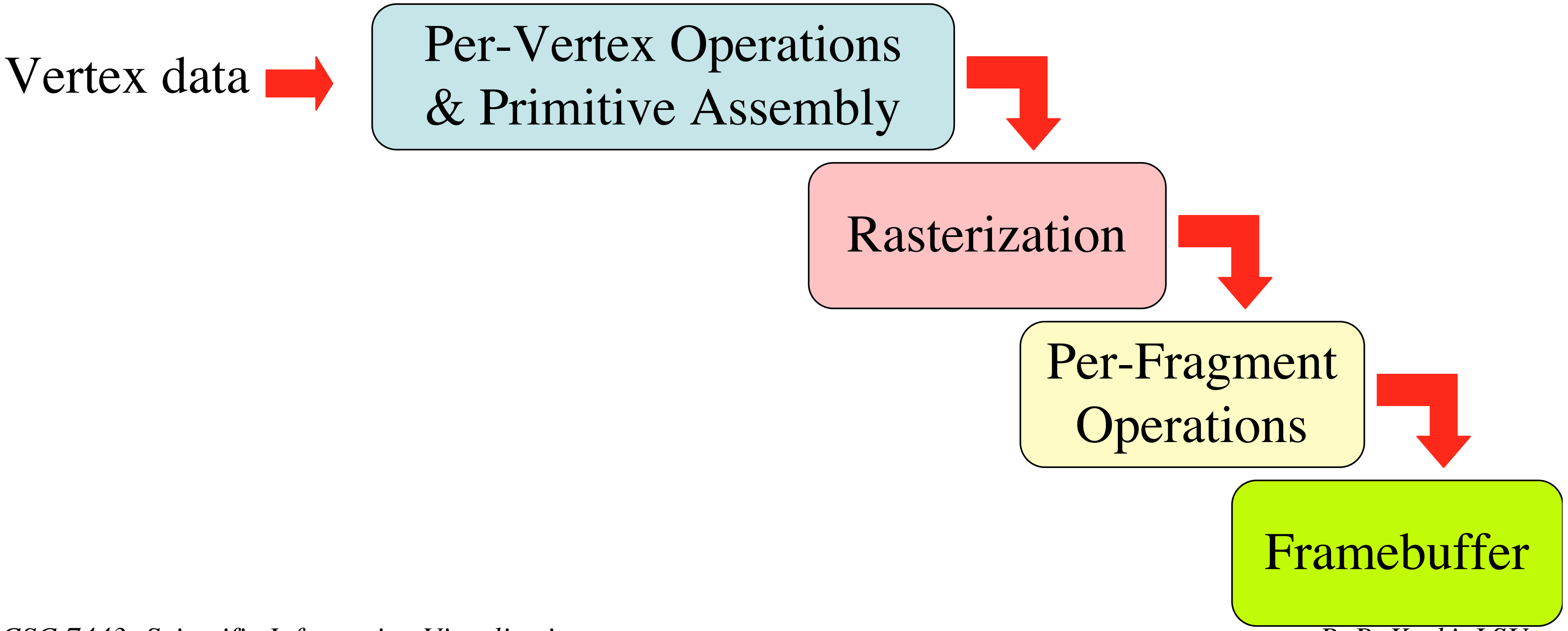
- By default, these states either have some values or are inactive

- Many states can be turned on and off with

  glEnable() and **glDisable()**

Graphics Pipeline

- OpenGL rendering pipeline
  - a series of processing stages from vertex data to display



Stages in Rendering Process

- **Vertex data:** Data for geometric objects consist of vertices
- **Per-vertex operations**: Translations and rotations are performed for some vertices. Positions in the 3D world are projected onto positions on the screen. Lighting calculations are performed using the vertices, surface normal, light sources, and material properties.
- **Primitive assembly**: Clipping eliminates portions of geometry, which fall outside the screen.
- **Rasterizations**: Conversion of geometric data into **fragments**. Each fragment square corresponds to a pixel in the framebuffer. Color and depth (z coordinate) values are assigned.
- **Per-fragment operations**: Hidden surface removal using the depth buffer (z buffer) or alpha blending for transparent materials.
- **Framebuffer**: A collection of buffers that store data for screen pixels (screen is, for example, 1024 pixels wide and 1024 pixels high) such as color, depth information for hidden surface removal, etc.

OpenGL Basics: Summary

- OpenGL and related libraries
- Window Management
- Basic structure of OpenGL program
- OpenGL as a state machine
- Graphics pipeline