CS-423: Computer Graphics
OpenGL and GLUT

Introduction

- Library to address pixel on screen
- Provides shading, rendering, texture mapping and lighting

History

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

OpenGL

- The success of IrisGL lead to OpenGL (1992), a platform-independent API that was
  - Easy to use
  - Close enough to the hardware to get excellent performance
  - Focused on rendering
  - Omitted windowing and input to avoid window system dependencies

OpenGL Portability

- Portability
  - Different platforms (frame buffer, video cards, CPU, etc.)
  - Operating system independent
  - Can take code from one machine to another with minimal effort
- How it works?
  - E.g. glVertex3f()
    - Microsoft will provide their own .dll and .lib
    - Linux will provide their own .dll and .lib
- To be OpenGL compliant each platform must have a basic set of functions

4/21/2023 5

5

OpenGL UI Interfacing

- OpenGL does not have UI capabilities
  - Dialog Boxes
  - Windows
  - Mouse
  - Context Menu
  - Event Mechanism
- OpenGL is focused on Graphics only
- Use MFC, VB or GLUT for GUI handling

4/21/2023 6

6

GLUT

- OpenGL Utility Toolkit (GLUT)
  - Provides functionality common to all window systems
    - Open a window
    - Mouse and Keyboard inputs
    - Menus
    - Event-driven
  - Code is portable but GLUT lacks the functionality of a good toolkit for a specific platform

7

Lack of Object Orientation

- OpenGL is not object oriented so that there are multiple functions for a given logical function
  - glVertex3f
  - glVertex2i
  - glVertex3dv
- Underlying storage mode is the same
- Easy to create overloaded functions in C++ but issue is efficiency

8

Adding New Source Code Files

- Then add to the top of this file the lines:
- // Standard includes
- #include <stdlib.h>
- #include <stdio.h>
- #include <string.h> // string-related functions
- #include <time.h> // time-related functions
- #include <math.h> // math functions (sin, cos, etc.)
- // OpenGL includes
- #include <GL/glut.h>
- #include <GL/gl.h>
- #include <GL/glu.h>
- Important: include <GL/glut.h> before all the other GL include files! It shouldn't matter, but under Windows it apparently does.

OpenGL function format

glu

function name

dimensions

`glVertex3f(x,y,z)`

belongs to GL library

`x,y,z` are `floats`

`glVertex3fv(p)`

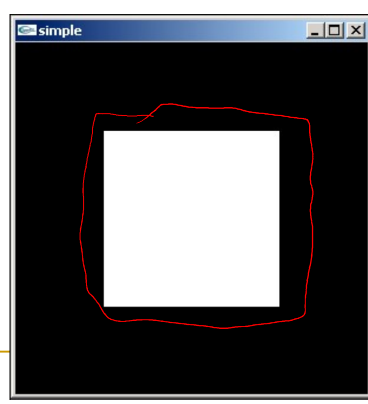
`p` is a pointer to an array

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,….

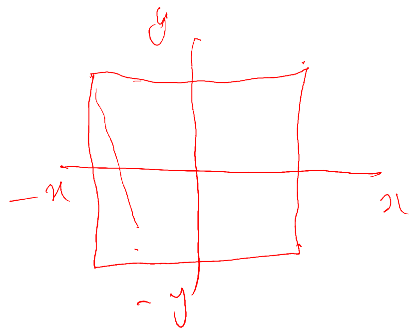
A Simple Program

- Generate a square on a solid background



Simple.cpp

```
#include <GL/glut.h>
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();
}
int main(int argc, char** argv)
{
glutCreateWindow("simple");
glutDisplayFunc(mydisplay);
glutMainLoop();
}
```



Event Loop

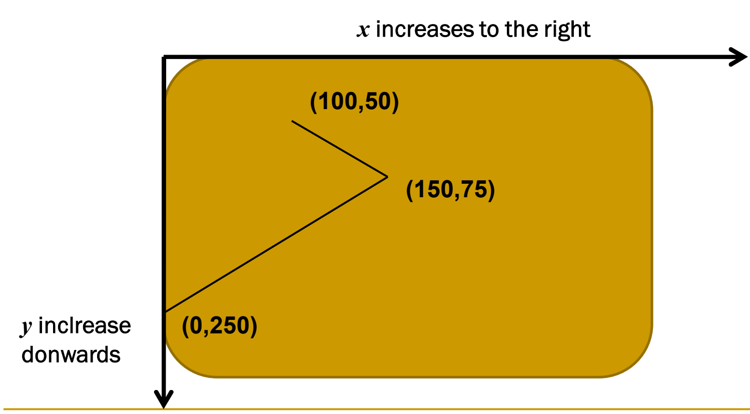
- Note that the program defines a display callback function named mydisplay
  - Every glut program must have a display callback
  - The display callback is executed whenever OpenGL decides the display must be refreshed, for example when the window is opened
  - The main function ends with the program entering an event loop
- Can have other callback functions as well

GETTING STARTED

- You need an environment consisting of
  1. Hardware to display pictures (usually a CRT display generally called a screen) and
  2. A library of software tools that your program can use.
- Initialization of the hardware
  1. Establishes the display mode (to graphics)
  2. Setting up a coordinate system on the display

COORDINATE SYSTEMS – THREE POSSIBILITIES (1/3)

1. The entire screen is used for drawing



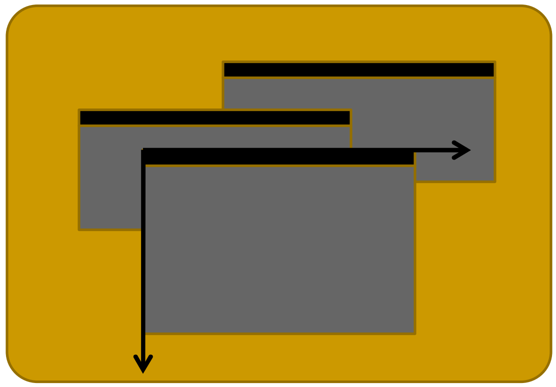
COORDINATE SYSTEMS – THREE POSSIBILITIES (2/3)

2. A window-based system is used.

    Different rectangular windows of different sizes at various locations are supported.

    Initialization involves creating and opening a new window.

    A coordinate system is attached to each window. X increases to the right and y increases downwards



X-Windows on Linux, Solaris, Ultrix etc
Windows Application Programming Interface (API) on Windows X
Quick Drawn on Apple Macintosh system

17

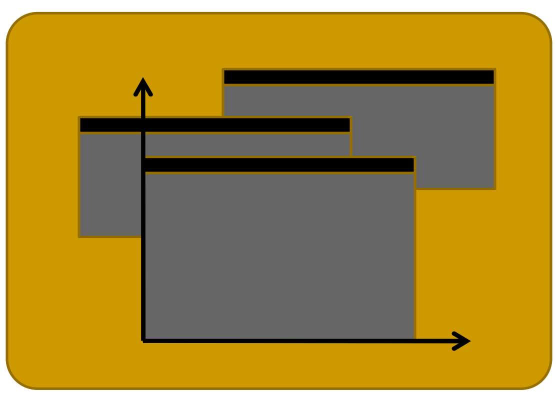
COORDINATE SYSTEMS – THREE POSSIBILITIES (3/3)

3. A window-based system is used.

    Different rectangular windows of different sizes at various locations are supported.

    Initialization involves creating and opening a new window.

    A coordinate system is attached to each window. X increases to the right and y increases upwards



Any window based system using OpenGL

18

Setting Up a Coordinate System

```
void myInit(void)
{
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0, 640.0, 0, 480.0);
}
// sets up coordinate system for window from
   (0,0) to (639, 479)
```


Setting Background Color in GL

- glClearColor (red, green, blue, alpha);
  - Sets background color.
  - Alpha is degree of transparency; use 0.0 for now.
- glClear(GL_COLOR_BUFFER_BIT);
  - clears window to background color

DEVICE INDEPENDENCE PROGRAMMING AND OPENGL

- Device independent graphics programming means a uniform approach is made so that the same program could be compiled and run on a variety of environments with the guarantee to produce the same results,
- OpenGL offers such a tool.
- OpenGL provides an API (application programming interface), i.e. a collection of routines that a programmer can call, shielding the programmer from the hardware and software details of the system.
- OpenGL is most powerful when drawing images of complex three-dimensional (3D) scenes.
- It also works well for two-dimensional (2D) drawings … which we will be looking into for the time being.

WINDOW-BASED PROGRAMMING

- Most window-based programs are event-driven: a program responds to various events such as click of a mouse, pressing of a key on the keyboard or the resizing of the window.
- The systems maintains an event-queue which receives messages stating that certain event has occurred and deals with them on first come first served basis.
- The programs are generally written in terms of call-back functions: the function that is executed when a particular event takes place.

Libraries to Include

- **GL**, for which the commands begin with GL;
- **GLUT**, the GL Utility Toolkit, opens windows, develops menus, and manages events.
- **GLU**, the GL Utility Library, which provides high level routines to handle complex mathematical and drawing operations.
- **GLUI**, the User Interface Library, which is completely integrated with the GLUT library.
  - The GLUT functions must be available for GLUI to operate properly.
  - GLUI provides sophisticated controls and menus to OpenGL applications.

OPENING A WINDOW FOR DRAWING

```
// appropriate #include go here.
void main(int argc, char** argv)
{
  glutInit(&argc, argv);                          // initialize the toolkit
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // set the display mode
  glutInitWindowSize(640, 480);                   // set the window size
  glutInitWindowPosition(100, 150); // set the window position on screen
  // open the screen window
  glutCreateWindow("My first graphics program in opengl");

  // register the callback functions
  glutDisplayFunc(myDisplay);
  glutReshapeFunc(myReshape);
  glutMouseFunc(myMouse);
  glutKeyboardFunc(myKeyboard);

  myInit();     // some additional initialization as required
  glutMainLoop();
}
```

What the Code Does

- glutInit (&argc, argv) initializes Open-GL Toolkit
- glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB) allocates a single display buffer and uses colors to draw
- glutInitWindowSize (640, 480) makes the window 640 pixels wide by 480 pixels high

What the Code Does (2)

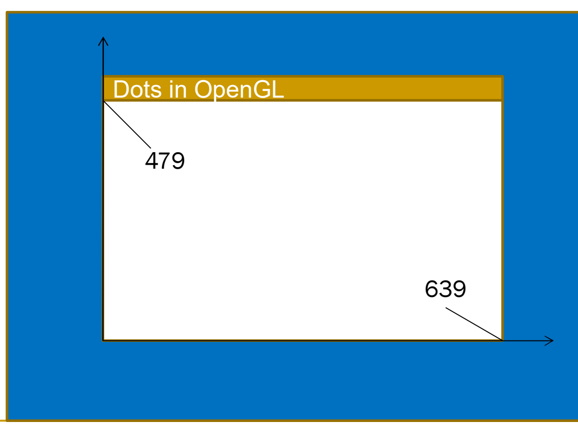
- glutInitWindowPosition (100, 150) puts upper left window corner at position 100 pixels from left edge and 150 pixels down from top edge
- glutCreateWindow (“my first attempt”) opens and displays the window with the title “my first attempt”
- Remaining functions register callbacks

What the Code Does (3)

- The call-back functions you write are registered, and then the program enters an endless loop, waiting for events to occur.
- When an event occurs, GL calls the relevant handler function.

EXPLANATION

The initial coordinate system for drawing



A COMPETE OPENGL PROGRAM TO DRAW THREE DOTS

- In the next few slides, I present a complete OpenGL program to draw some dots on screen.
- This will be followed by an explanation of some of the functions that have been used in this program.

A COMPETE OPENGL PROGRAM TO DRAW THREE DOTS

```
#include <gl/Gl.h>
#include <gl/glut.h>

void myInit()                  // *************** myInit **************
{
  glClearColor(1.0, 1.0, 1.0, 0.0);  // set white background color
  glColor3f(0.0, 0.0, 0.0);                   // set the drawing color
  glPointSize(4.0);                           // a dot will be 4 by 4 pixels
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}

void myDisplay(void)           // *************** myDisplay ****************
{
  glClear(GL_COLOR_BUFFER_BIT);      // clear the screen
  glBegin(GL_POINTS);                // draw three points
    glVertex2i(100, 50);
    glVertex2i(100, 130);
    glVertex2i(150, 130);
  glEnd();
  glFlush();                         // send all output to display
}
```

A COMPETE OPENGL PROGRAM TO DRAW THREE DOTS

```
void main(int argc, char** argv)
{
  glutInit(&argc, argv);                       // initialize the
   toolkit
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);    // set the
   display mode
  glutInitWindowSize(640, 480);                // set the window size
  glutInitWindowPosition(100, 150); // set the window position on
   screen
  // open the screen window
  glutCreateWindow("Dots in opengl");
  glutDisplayFunc(myDisplay); // register the redraw functions
  myInit();    // some additional initialization as required
  glutMainLoop();

}
```


EXPLANATION

- Output primitives line points, lines, polylines and polygons are defined in terms of one or more vertices.
- Such objects are drawn by passing a list of vertices. This list is defined within a pair of OpenGL function calls: glBegin() and glEnd(). We declare the types of the object through a library defined constant in glBegin argument list.

```
glBegin(GL_POINTS);      // draw three points
   glVertex2i(100, 50);
   glVertex2i(100, 130);
   glVertex2i(150, 130);
glEnd();
```

- Here GL_POINTS is a constant built-into OpenGL. Other constants are GL_LINES, GL_POLYGON etc.

OPENGL DATA TYPES

Suffix	Data Type	Typical C or C++ type	OpenGL type name
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	int or long	GLint or GLsizei
f	32-bit floating point	float	GLfloat or GLclampf
d	64-bit floating point	double	Gldouble or Glclampd
ub	8-bit unsigned number	unsigned char	GLubyte or GLboolean
us	16-bit unsigned number	unsigned short	GLushort
ui	32-bit unsigned number	unsigned int or unsigned long	GLuint, GLenum, GLbitfield

4/21/2023 33

33

OPENGL DATA TYPES

- As an example, a function using suffix i “expects” a 32-bit integer, but your system might translate int as a 16-bit integer

```
void drawDot(int x, int y) {
 glBegin(GL_POINTS);      // draws a dot at (x, y)
  glVertex2i(150, 130);
 glEnd();
}
```

- A better option will be to use:

```
void drawDot(GLint x, GLint y) {
 glBegin(GL_POINTS);      // draws a dot at (x, y)
  glVertex2i(150, 130);
 glEnd();
}
```

4/21/2023 34

34

Example of Construction

- glVertex2i (…) takes integer values
- glVertex2d (…) takes floating point values

- OpenGL has its own data types to make graphics device-independent
  - Use these types instead of standard ones

THE OPENGL STATE

- OpenGL keeps track of many "state variables", such as
  - The current size of a point,
  - The current color of a drawing
  - The current background color
- The value of a state variable remains active until a new value is given.
- The size of a point can be set with glPointSize(), which takes one floating point argument. If the argument is 3.0, the point is usually drawn as a square with 3 pixels on a side.
- The color of a drawn can be specified using

```
glColor3f( red, green, blue);
```

where the values of red, green and blue vary between 0.0 and 1.0.

- The background color is set with

```
glClearColor( red, green, blue, alpha);
```

where alpha specifies a degree of transparency.

- To clear the entire window to the background color, use

```
glClear( GL_COLOR_BUFFER_BIT);
```

Setting Drawing Colors in GL

- glColor3f(red, green, blue);
  // set drawing color
  - glColor3f(1.0, 0.0, 0.0);   // red
  - glColor3f(0.0, 1.0, 0.0);   // green
  - glColor3f(0.0, 0.0, 1.0);   // blue
  - glColor3f(0.0, 0.0, 0.0);   // black
  - glColor3f(1.0, 1.0, 1.0);   // bright white
  - glColor3f(1.0, 1.0, 0.0);   // bright yellow
  - glColor3f(1.0, 0.0, 1.0);   // magenta

37

Drawing dot constellations

- A dot constellation is a pattern of dots or points.

Example: The Big Dipper

- The names and coordinates of the eight stars in the Big Dipper, a familiar sight in the night sky are given by the following ordered triplets:
  {Dubhe, 289, 190}, {Merak, 320, 128}, {Phecda, 194, 101}, {Alioth, 129, 83}, {Mizar, 75, 73}, {Alcor, 74, 74}, {Alkaid, 20, 10}
- Since, we have few points here, so they can be hardwired into the source code.
- For larger data sets, it is more convenient to store them in a file and then read from that file to plot these points.
- Things to do:
  - Play with changing the color and size of the points.
  - Change the background color.

38

Drawing dot constellations

Example: Drawing the Sierpinski Gasket

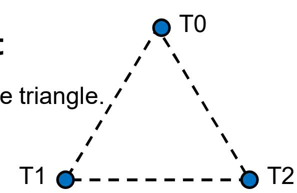
- Sierpinski's gasket is a fractal.
- It can be produced by calling the drawDot() function many times.
- Denoting a kth point as $p_k = (x_k, y_k)$ each point is based on the previous point.
- The procedure goes like this:

Drawing dot constellations

Example: Drawing the Sierpinski Gasket

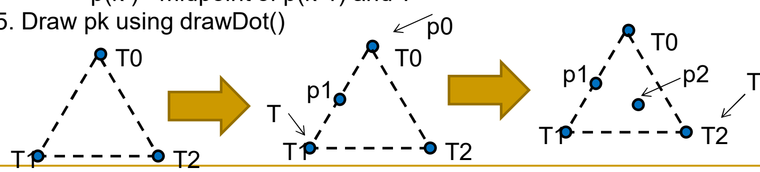
1. Choose three fixed points T0, T1 and T2 to form some triangle.

2. Choose the initial point p0 to be drawn by selecting one of the points T0, T1 and T2 at random.



Now iterate the following steps until the pattern is satisfactorily filled in or the maximum number of iterations has reached.

3. Choose one of the three points T0, T1 and T2 at random, and call it T.
4. Construct the next point p(k) as the mid-point between T and the previous point p(k-1). That is
   p(k)= midpoint of p(k-1) and T
5. Draw pk using drawDot()



Drawing dot constellations

Example: Drawing the Sierpinski Gasket

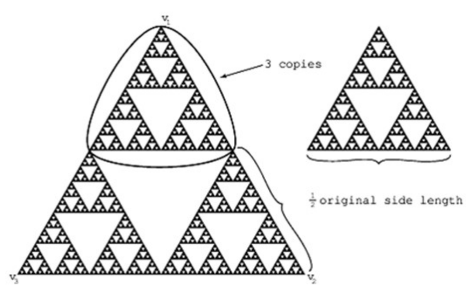
- How to go about it?
  - It is convenient to define a simple class GLintPoint, that describes a point whose coordinates are integers:

```
class GLintPoint {
public:
        GLint x, y;
};
```

  - We then move on to build and initialize an array of three such points T[0], T[1] and T[2].
  - There is no need to store each point, since we simply want to draw it and move on. So we set up a variable point to hold this changing point.
  - We use i=random(3) to choose one of the points T[i] at random. This function is defined as:

```
int random(int m) {
        return rand()%m;
}
```


Sierpinski Gasket



Drawing dot constellations

Example: Drawing the Sierpinski Gasket

```
void Sierpinski(void) {
    GLintPoint T[3] = { {10,10}, {300, 30}, {200, 300} };
    int index = random(3);
    GLintPoint point = T[index];
    drawDot( point.x, point.y);
    for (int i=0; i<1000; i++) {
        index = random(3);
        point.x = ( point.x + T[index].x ) / 2;
        point.y = ( point.y + T[index].y ) / 2;
        drawDot( point.x, point.y);
    }
    glFlush();
}
```


Drawing dot constellations

Example: Simple "Dot Plots"

- This example deals with learning the behaviour of some mathematical function f(x) as x varies.
- Suppose we have

$$f(x) = e^{-x}\cos(2\pi x)$$

  where x varies from x=0 to x=4.
- To plot this function we "sample" it at a collection of equispaced x-values and plot a dot at each coordinate pair (x, f(x)).
- Choosing a suitable increment, say 0.005, between consecutive x-values, the basic process will run as follows:

```
glBegin(GL_POINTS);
        for (GLdouble x=0; x<4.0; x += 0.005)
                glVertex2d(x, f(x));
glEnd();
glFlush
```

Drawing dot constellations

Example: Simple "Dot Plots"

- Problem:
  1. The picture produced is impossibly tiny, because the values of x from 0 to 4 are mapped to only first four pixels at the bottom of the screen window.
  2. The negative values of f(x) lie below the window and are not visible.
- Solution:
  1. Scaling x: The first problem is solved if we scale x and then plot it. Consider a screen of width "screenWidth", the scaled x values can be obtained as

     sx = x * screenWidth / 4.0;

     So for x = 0, sx = 0 and for x = 4.0, sx = screenWidth.

Drawing dot constellations

Example: Simple "Dot Plots"

- Solution:
  2. Scaling and Shifting y: The second problem is solved if we place the plot at the center of the screen window. Consider a screen of height "screenHeight", the scaled and shifted y values can be obtained as

     sy = (y + 1.0)* screenHeight / 2.0;

     So for y = -1.0, sy = 0 and for y = 1.0, sy = screenHeight.
- Note:
  - The conversion from x to sx and from y to sy are of form:

    sx = A x + B

    sy = C y + D

    Affine Transformations
  - For properly chosen values of A, B, C and D.
  - A and C are scaling coefficients and
  - B and D are shifting coefficients.

Drawing dot constellations

Example: Simple Dot Plot – Complete Program (1/2)

```
#include <math.h>
#include <gl/Gl.h>
#include <gl/glut.h>
const int screenWidth = 640;  // width of the screen window in pixels
const int screenHeight =480;  // height of the screen window in pixels
GLdouble A, B, C, D;                    // scaling and shifting coefficients
void myInit(void) {
	glClearColor(1.0, 1.0, 1.0, 0.0);     // background color is set to white
	glColor3f(0.0, 0.0, 0.0); // drawing color is set to black
	glPointSize(2.0);                   // a dot is 2 by 2 pixels
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0.0, (GLdouble)screenWidth, 0.0, (GLdouble)screenWeight);
	A = screenWidth / 4.0;
	B = 0.0
	C = D = screenHeight / 2.0;
}
```


Drawing dot constellations

Example: Simple Dot Plot – Complete Program (2/2)

```
void myDisplay(void) {
	glClear(GL_COLOR_BUFFER_BIT);                 // clear the screen
	glBegin(GL_POINTS);                           // draw the points
	for (GLdouble x=0; x<4.0; x += 0.005) {
		GLdouble func = exp(-x)*cos(2*3.14159265*x);
		glVertex2d( A*x + B, C*func + D );  }
	glEnd(); glFlush();
}
void main(int argc, char **argv) {
	glutInit(&argc, argv);                        // initialize the toolkit
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);  // set display mode
	glutInitWindowSize(screenWidth, screenHeight);  // set window size
	glutInitWindowPosition(100, 150);         // set window position on screen
	glutCreateWindow("Dot Plot of a Function");
	glutDisplayFunc(myDisplay);               // register display function
	myInit();
	glutMainLoop();                           // go for a perpeptual loop
}
```