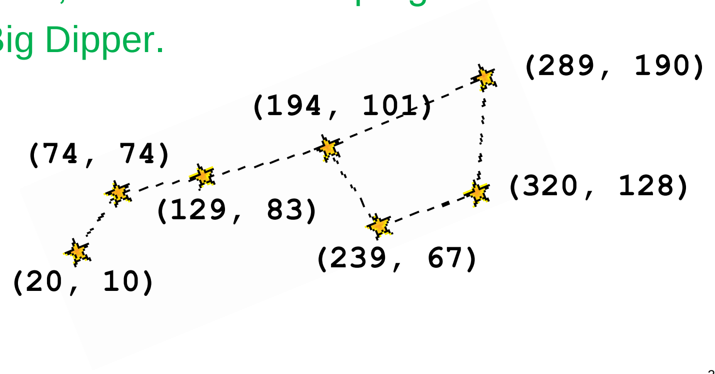
EBU5405
3D Graphics Programming Tools

Drawing in OpenGL

Dr. Marie-Luce Bourguet
(marie-luce.bourguet@qmul.ac.uk)

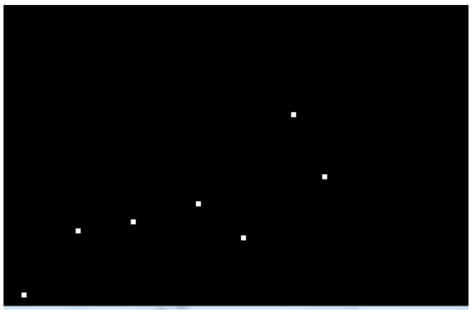
The Big Dipper

Replacing **glBegin(GL_POLYGON);**
with **glBegin(GL_POINTS);** in the previous program, let's write a new program that draws the Big Dipper.



The Big Dipper

```
void mydisplay() {
  GLint vertices[7][2] = {{20, 10}, {74, 74}, {129, 83}, {194, 101}, {239, 67},
                                    {320, 128}, {289, 190}};
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POINTS);
         glVertex2iv(vertices[0]);
         glVertex2iv(vertices[1]);
         glVertex2iv(vertices[2]);
         glVertex2iv(vertices[3]);
         glVertex2iv(vertices[4]);
         glVertex2iv(vertices[5]);
         glVertex2iv(vertices[6]);
  glEnd();
  glFlush();
}
```

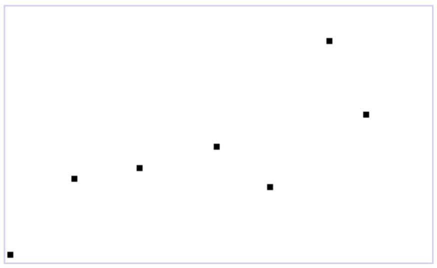


3

3

The Big Dipper (inversed colours)

```
void mydisplay() {
  GLint vertices[7][2] = {{20, 10}, {74, 74}, {129, 83}, {194, 101}, {239, 67},
                                    {320, 128}, {289, 190}};
   glClearColor (1.0, 1.0, 1.0, 1.0);
   glColor3f (0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POINTS);
         glVertex2iv(vertices[0]);
         glVertex2iv(vertices[1]);
         glVertex2iv(vertices[2]);
         glVertex2iv(vertices[3]);
         glVertex2iv(vertices[4]);
         glVertex2iv(vertices[5]);
         glVertex2iv(vertices[6]);
  glEnd();
  glFlush();
}
```



4

4

OpenGL State

- OpenGL operates as a state machine.
- It means that once the value of a property is set, the value persists until a new value is set.
- For example, if we use glColor command to set the current drawing color to black, black will be used to draw ALL objects until we use glColor command again to change the drawing color.
- "Everything shall remain until you explicitly change it!"

OpenGL State

- OpenGL functions are of two types
  - Primitive generating
    - Can cause output if primitive is visible
    - How vertices are processed and the appearance of primitives are controlled by the state
  - State changing
    - Transformation functions
    - Attribute functions

OpenGL Functions

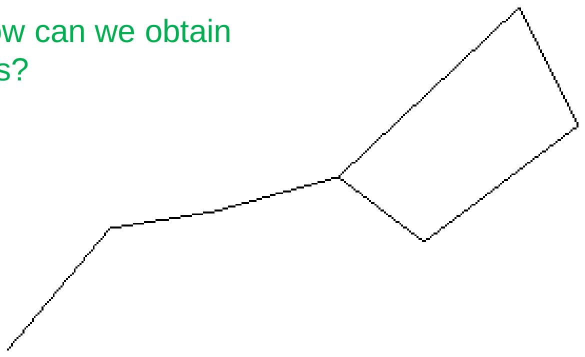
- Primitives
  - Points
  - Line Segments
  - Polygons …
- Attributes
- Transformations
  - Modelling
  - Viewing
- Control (GLUT)
- Input (GLUT)
- Query

7

7

Drawing lines

How can we obtain this?



8

8

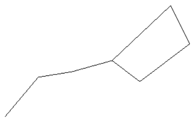
The Big Dipper (2)

Let’s change the primitive from GL_POINTS to GL_LINES

```
glBegin(GL_LINES);
        glVertex2iv(vertices[0]);
        glVertex2iv(vertices[1]);
        glVertex2iv(vertices[2]);
        glVertex2iv(vertices[3]);
        glVertex2iv(vertices[4]);
        glVertex2iv(vertices[5]);
        glVertex2iv(vertices[6]);
 glEnd();
 glFlush();
```

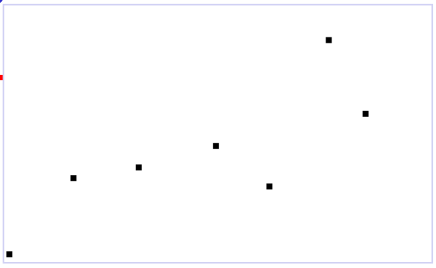
9

9



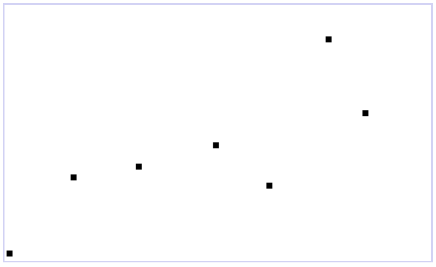
Drawing lines

GL_LINES



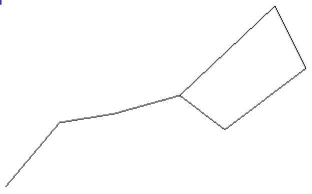
GL_LINE_STRIP

GL_LINE_LOOP



10

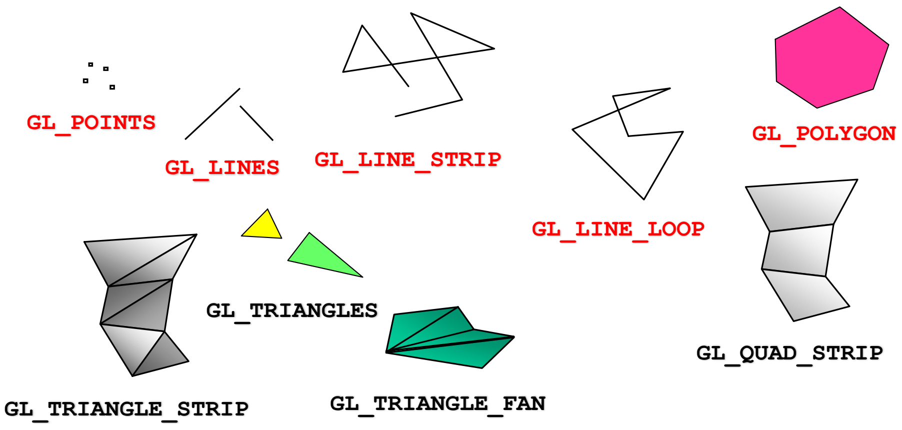
The Big Dipper (2)



11

11

OpenGL Primitives



12

12

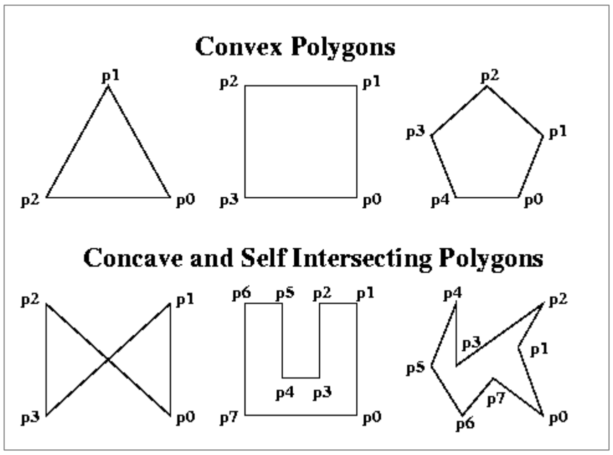
Polygon Issues

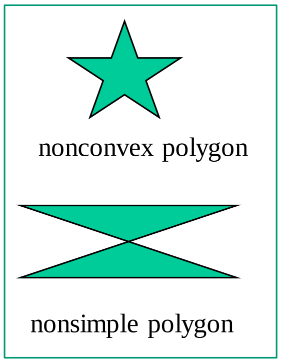
- OpenGL will only display polygons correctly that are
  - Simple: edges cannot cross
  - Convex: All points on line segment between two points in a polygon are also in the polygon
  - Flat: all vertices are in the same plane
- User program can check if above true
  - OpenGL will produce output if these conditions are violated but it may not be what is desired
- Triangles satisfy all conditions

13

13

Polygon Issues





14

14

GLU object functions

- gluDisk — draws a disk
- gluSphere — draws a sphere
- gluCylinder — draws a cylinder

- Not primitives, but useful objects

15

15

Attributes

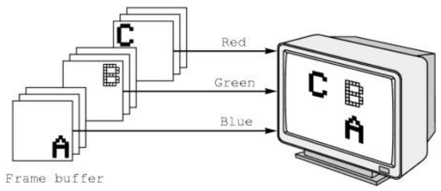
- Attributes are part of the OpenGL state and determine the appearance of objects
  - Color (points, lines, polygons)
  - Size and width (points, lines)
  - Stipple pattern (lines, polygons)
  - Polygon mode
    - Display as filled: solid color or stipple pattern
    - Display edges
    - Display vertices

16

16

RGB color

- Each color component is stored separately in the frame buffer
- Usually 8 bits per component in buffer
- Note in **glColor3f** the color values range from 0.0 (none) to 1.0 (all), whereas in **glColor3ub** the values range from 0 to 255



17

17

Color and State

- The color as set by **glColor** becomes part of the state and will be used until changed
  - Colors and other attributes are not part of the object but are assigned when the object is rendered
- We can create conceptual *vertex colors* by writing code such as

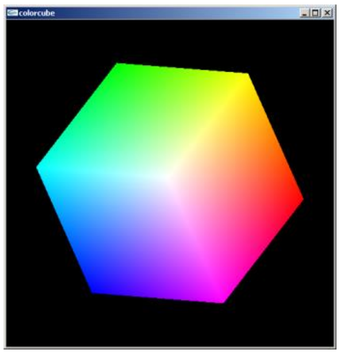
```
glColor
 glVertex
 glColor
 glVertex
```

18

18

Smooth Color

- Default is *smooth* shading
  - OpenGL interpolates vertex colors across visible polygons
- Alternative is *flat shading*
  - Color of first vertex determines fill color
- **`glShadeModel`**
  `(GL_SMOOTH)`
  or **`GL_FLAT`**



Using dot plots

How do you expect the graph of this function to look like?

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

Using dot plots

- Plot a dot at each coordinate pair (x_i, $f(x_i)$)
- Choose some suitable increment between consecutive x-values
- Scale and position the values to be plotted so that they cover the screen window area

21

21

Using dot plots

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

For example:

```
Gldouble A, B, C, D, x;
A = screenWidth / 4.0;
B = 0.0;
C = D = screenHeight / 2.0;
glBegin(GL_POINTS);
for (Gldouble x = 0; x < 4.0; x += 0.005)
        glVertex2d (A * x + B, C * f(x) + D);
glEnd();
glFlush();
```

22

22

Using dot plots

What do you expect to see with this program ?

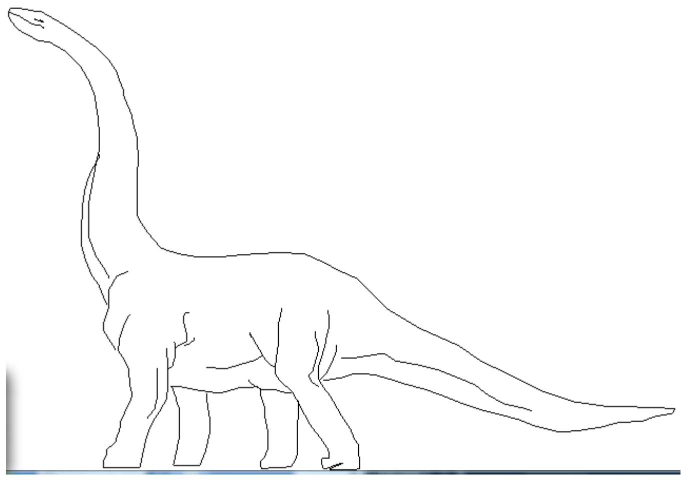
```
Gldouble A, B, C, D, x;
A = screenWidth / 4.0;
B = 0.0;
C = D = screenHeight / 2.0;
glBegin(GL_POINTS);
for (Gldouble x = 0; x < 4.0; x += 0.05)
        glVertex2d (A * x + B, C * f(x) + D);
glEnd();
glFlush();
```

Using dot plots

How about this program ?

```
Gldouble A, B, C, D, x;
A = screenWidth / 4.0;
B = D = 0.0;
C = screenHeight / 2.0;
glBegin(GL_POINTS);
for (Gldouble x = 0; x < 4.0; x += 0.005)
        glVertex2d (A * x + B, C * f(x) + D);
glEnd();
glFlush();
```

Drawing from a file



25

25

Drawing from a file

```
void drawPolyLineFile(char *filename) {
  FILE *ifp;
  int i, j;
  GLint numpolys, numLines, x, y;
  ifp = fopen (filename, "r");
  fscanf (ifp, "%d", &numpolys);
  for (j=0; j < numpolys; j++) {
       fscanf (ifp, "%d", &numLines);
       glBegin (GL_LINE_STRIP);
       for (i = 0; i < numLines; i++) {
              fscanf (ifp, "%d %d", &x, &y);
              glVertex2i (x, y);
       }
       glEnd();
  }
  glFlush();
  fclose (ifp);
}
```

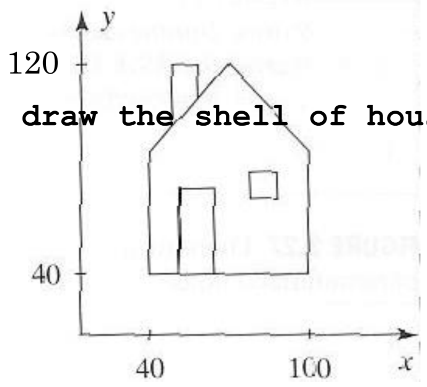
```
21
29
32 435
10 439
4 438
2 433
4 428
6 425
10 420
15 416
21 413
Etc.
```

26

26

Drawing parameterised figures

Here is a house consisting of a few polylines. How would you complete the code for drawing the chimney, the door and the window?

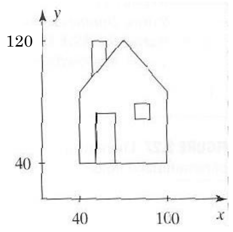


```
glBegin(GL_LINE_LOOP);// draw the shell of house
  glVertex2i(40, 40);
  glVertex2i(40, 90);
  glVertex2i(70, 120);
  glVertex2i(100, 90);
  glVertex2i(100, 40);
glEnd();
```

27

27

Drawing parameterised figures



Here is the same house "parameterised" …

```
int peak[2] = {70, 120};
int width = 60;
int height = 80;
glBegin(GL_LINE_LOOP);
  glVertex2i(peak[0], peak[1]);
  glVertex2i(peak[0]+width/2,peak[1]-3*height/8);
  glVertex2i(peak[0]+width/2,peak[1]-height);
  glVertex2i(peak[0]-width/2,peak[1]-height);
  glVertex2i(peak[0]-width/2,peak[1]-3*height/8);
glEnd();
```

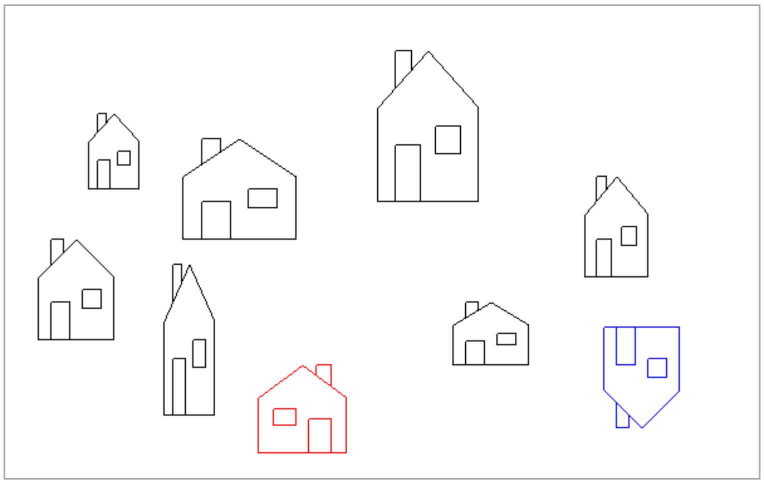
Complete the code for drawing the chimney, the door and the window …

28

28

Drawing parameterised figures

How would you achieve this?



29

29