Drawing Geometric Objects

Drawing Primitives

- OpenGL sets three types of drawing primitives
  - Points
  - Lines
  - Polygons, e.g, triangles

- All primitives are represented in terms of vertices
  - that define the positions of the points themselves or the ends of line segments or the corners of polygons

Points

- Object of zero dimension (infinitely small)
- Specified by a set of floating-point numbers (coordinates) called a **vertex**
- Displayed as a single pixel on screen
- void **glPointSize**(Glfloat *size*);
  - ➢ Sets the size of a rendered point in pixels

Specifying Vertices

- void **glVertex**{234}{sifd}[v](TYPE *coords*);
  - Specifies a vertex for use in describing a geometric object

    glVertex2s(2,4);

    glVertex4f(2.3, 1.0, -2.2, 1.0);

    GLdouble dvect[3] = {5.0, 9.0, 4.0);

    glVertex3dv(dvect);

- **OpenGL works in homogeneous coordinates**

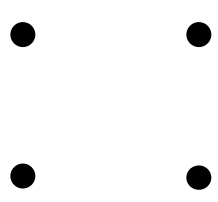
  vertex:: (*x*, *y*, *z*, *w*)

  ***w* = 1 for default**

Displaying Vertices

- Bracket a set of vertices between a call to **glBegin**() and a call to **glEnd**() pair
  - The argument GL_POINTS passed to glBegin() means drawing vertices in the form of the points

```
glBegin(GL_POINTS);
    glVertex2f(0.0, 0.0);
    glVertex2f(4.0, 0.0);
    glVertex2f(4.0, 4.0);
    glVertex2f(0.0, 4.0);
glEnd();
```



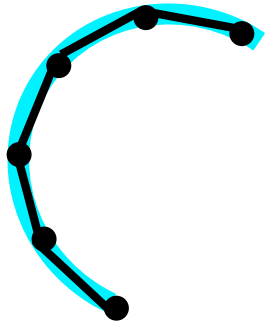
  - Other drawing options for vertex-data list

Lines (GL_LINES)

Polygon (GL_POLYGON)

Lines

- The term *line* refers to a *line segment*
- Specified by the vertices at their endpoints
- Displayed solid and one pixel wide
- Smooth curves from line segments

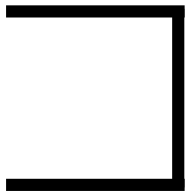


Drawing Lines

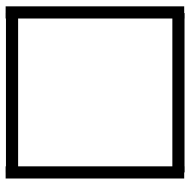
- To draw a vertex-data list as lines

```
glBegin(GL_LINES);
    glVertex2f(0.0, 0.0);
    glVertex2f(4.0, 0.0);
    glVertex2f(4.0, 4.0);
    glVertex2f(0.0, 4.0);
glEnd();
```

- GL_LINE_STRIP
  - A series of connected lines



- GL_LINE_LOOP
  - A closed loop



Wide and Stippled Lines

- void **glLineWidth**(GLfloat *width*);
  - Sets the width in pixels for rendered lines

- void **glLineStipple**(GLint *factor*, GLshort *pattern*);
  - Sets the current stippling pattern (dashed or dotted) for lines
  - *Pattern* is a 16-bit series of 0s and 1s

    1 means one pixel drawing, and 0 not drawing
  - *Factor* stretches the pattern multiplying each bit

  - Trun on and off stippling

    glEnable(GL_LINE_STIPPLE)

    glDisable(GL_LINE_STIPPLE)

Example of Stippled Lines

- **glLineStipple**(1, 0x3F07);

  Pattern **0x3F07** translates to **0011111100000111**

  Line is drawn with 3 pixels on, 5 off, 6 on, and 2 off

- **glLineStipple**(2, 0x3F07);

  Factor is 2

  Line is drawn with 6 pixels on, 10 off, 12 on, and 4 off

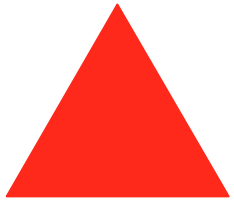
Polygon

- Areas enclosed by single closed loops of line segments
- Specified by vertices at the corners
- Displayed as solid with the pixels in the interior filled in

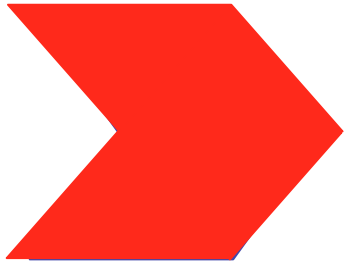
  Examples: Triangle and Pentagon

Polygon Tessellation

- Simple and convex polygon
  - Triangle
  - Any three points always lie on a plane



- Polygon tessellation
  - Nonsimple or nonconvex polygons can be represented in the form of triangles



- Curved surfaces can be approximated by polygons

Drawing Polygon

- Draw a vertex-data list as a polygon

```
glBegin(GL_POLYGON);
    glVertex2f(0.0, 0.0);
    glVertex2f(4.0, 0.0);
    glVertex2f(4.0, 4.0);
    glVertex2f(0.0, 4.0);
glEnd();
```

- GL_TRIANGLES

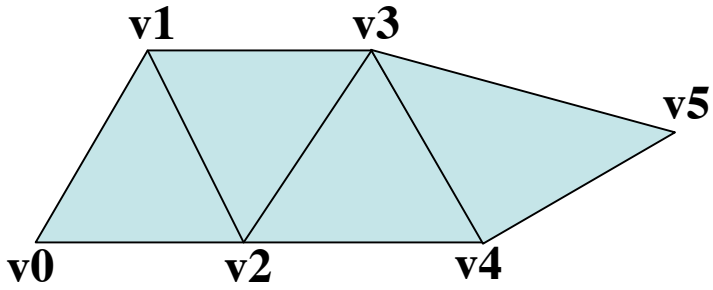
  Draws first three vertices as a triangle

- GL_QUADS

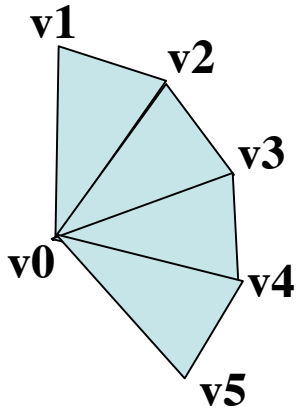
  Quadilateral is a four-sided polygon

Drawing Polygons

- GL_TRIANGLE_STRIP
  - Draws a series of triangles using vertices in the order
    v0,v1,v2; v2,v1,v3
    v2,v3,v4; v4,v3,v5
  - All triangles are drawn with the same orientation (clockwise order)



- GL_TRIANGLE_FAN
  - One vertex is in common to all triangles
  - Clockwise orientation



- GL_QUAD_STRIP
  - Draws a series of quadrilaterals

Polygons as Points and Outlines

- void **glPolygonMode**(GLenum *face*, Glenum *mode*);
  - Controls the drawing mode for a polygon's front and back faces

```
glPolygonMode(GL_FRONT, GL_FILL);
glPolygonMode(GL_BACK, GL_LINE);
glPolygonMode(GL_FRONT_AND_BACK,
    GL_POINT);
```

- By convention, polygons whose vertices appear in counterclockwise order are front-facing

```
GL_CCW
```

Deciding Front- or Back Facing

- Decision based the sign of the polygon's area, ***a*** computed in window coordinates

$$a = \frac{1}{2}\sum_{i=0}^{n-1}\left[x_i y_{i\oplus 1} - x_{i\oplus 1} y_i\right]$$

- For GL_CCW, if $a > 0$ means the polygon be front-facing, then $a < 0$ means the back-facing

- For GL_CW, if $a < 0$ for front-facing, then $a > 0$ for back-facing

Reversing and Culling Polygons

- void **glFrontFace**(GLenum *mode*);
  - Controls how front-facing polygons are determined
  - Default mode is GL_CCW (vertices in counterclockwise order)
  - Needs to be enabled

- void **glCullFace**(GLenum *mode*);
  - Indicates which polygons (back-facing or front-facing) should be discarded (culled)
  - Needs to be enabled

Stippling Polygons

- Void **glPolygonStipple**(const GLbyte **mask*);
  - Defines the current stipple pattern for the filled polygons
  - The argument is a pointer to a 32x32 bitmap (a mask of 0s and 1s)

- Needs to be enabled and disabled

```
glEnable(GL_POLYGON_STIPPLE);
glDisable(GL_POLYGON_STIPPLE);
```

Normal Vectors

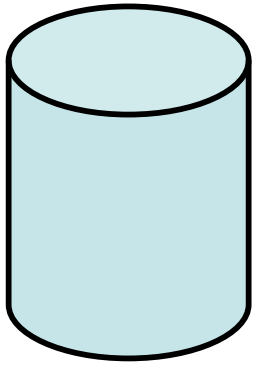
- Points in a direction that is perpendicular to a surface
  - The normal vectors are used in lighting calculations

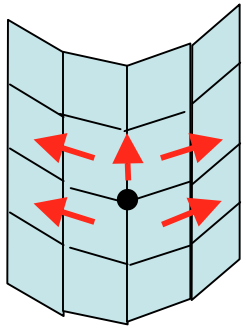
- void **glNormal3**(bsidf)(*TYPE nx, TYPE ny, TYPE nz*);
  - Sets the current normal vector as specified by the arguments

- void **glNormal3**(bsidf)**v**(const *TYPE* **v*);
  - Vector version supplying a single array ***v*** of three element

Finding Normal Vector

- Surfaces described with polygonal data
  - Calculate normal vector for each polygonal facet
  - Average these normals for neighboring facets
  - Use the averaged normal for the vertex that the neighboring facets have in common





- Using normal vectors in lighting model to make surface appear smooth rather than facet

Finding Normal Vector

- Make two vectors from any three vertices v1, v2 and v3

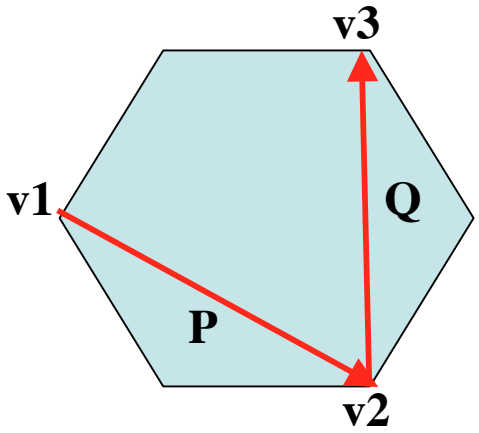
  $P = v1 - v2;\ \ Q = v2 - v3$

- Cross product of these vectors is perpendicular to polygonal surface

  $N = P \times Q = [Px\ Py\ Pz] \times [Qx\ Qy\ Qz]$

  $= [PyQz-QyPz)\ (QxPz-PxQz)\ (PxQy-QxPy]$

  $= [Nx\ Ny\ Nz]$



- Normalize the vector

  $n = [nx\ ny\ nz] = [Nx/L\ Ny/L\ Nz/L]$

  where L is length of the vector $[Nx\ Ny\ Nz]$

Vertex Arrays

- OpenGL has vertex array routines to specify a lot of vertex-related data with a few arrays
  - To reduce the number of function calls
  - To avoid processing of shared vertices

- Three steps in using vertex arrays
  - Activate up to eight arrays
  - Put data into the arrays
  - Render geometry with the data

Step1: Enabling Arrays

- void **glEnableClientState**(GLenum *array*);
  - Specifies the array to enable
  - Parameter *array* defines the type (up to eight types)

    GL_VERTEX_ARRAY

    GL_COLOR_ARRAY

    GL_NORMAL_ARRAY

    glEnableClientState(GL_NORMAL_ARRAY);

- void **glDisableCleintState**(GLenum *array*);
  - Specifies the array to disable

    glDisableClientState(GL_NORMAL_ARRAY);

Step2: Specifying Data for the Arrays

- void **glVertexPointer**(GLint *size*, GLenum *type*, GLsizei *stride*, const GLvoid **pointer*);
  - Specifies where vertex (spatial coordinate) data can be accessed
  - *Pointer* is the memory address of the first coordinate of the first vertex in the array

    Static GLint vertices[] = (2.0, 4.0, 1.5, ….)

    glVertexPointer(3, GL_FLOAT, 0, vertices);

- void **glColorPointer**(GLint *size*, GLenum *type*, GLsizei *stride*, const GLvoid **pointer*);

- void **glNormalPointer**(GLenum *type*, GLsizei *stride*, const GLvoid **pointer*);

Step 3: Dereferencing and Rendering

- void **glArrayElement**(GLint i^{th});
  - Obtains the data of one (the i^{th}) vertex for all enabled arrays
  - Called between **glBegin**() and **glEnd**()

- void **glDrawElements**(GLenum *mode*, GLsizei *count*, GLenum *type*, void **indices*);
  - Defines a sequence of geometric primitives (*mode*) using *count* number of elements with indices in the array *indices*

- void **glDrawArrays**(GLenum *mode*, GLint *first*, GLsizei *count*);
  - Constructs a sequence of geometric primitives (*mode*) using array elements starting at *first* and ending at *first+count-1*

Building Polygonal Models of Surfaces

- You can approximate smooth surfaces by polygons

- Important points
  - Polygon orientation consistency (all clockwise or all anticlockwise)
  - Caution at non-triangular polygons
  - Trade-off between display speed and image quality

Examples

- Building an icosahedron

- Polygonal approximation to a sphere