OpenGL Programming-1

EE – 7000

Sep,19,2011

Topics in OpenGL

- **OpenGL basics**
- **Drawing geometric objects**

- Viewing
- Color

- Lighting
- Some special topics (texture mapping)

What is OpenGL?

A Standard, hardware-independent interface to Graphics hardware

- Introduced in 1992
- Most widely used 3D graphics API
- Portable across a wide array of platforms

Current version: OpenGL 4.2

- Older versions: 1.'s, 2.'s and 3.'s

No commands for windows 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

Foundations for high-performance graphics

- Many APIs built on the top of OpenGL

What Is OpenGL?

GL routine has a prefix gl

glColor()

Head file for GL-library calls

#include <GL/gl.h>

Software information and download

http://www.opengl.org

OpenGL Command Syntax

OpenGL functions

Prefix gl and initial capital letters for each word making up the function name

glVertex() glClearColor()

OpenGL defined constants

Begin with GL_, use all capital letters, and use underscore to separate words

GL_COLOR_BUFFER_BIY GL_TRIANGLES

OpenGL Command Syntax

Suffixes in functions

void glVertex{234}{sifd}[v](TYPE coords)

2 or 3 or 4 means the # of arguments to be given
s or i or f or d means date 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 Data Types

b	8	GLbyte	signed char
s	16	GLshort	short
i	32	GLint, GLsizei	int / long
f	32	GLfloat, GLclampf	float
d	64	GLdouble, GLclampd	double
ub	8	GLubyte, Glboolean	unsigned char
us	16	GLushort	unsigned short
ui	32	GLuint, GLenum, GLbitfield	unsigned int

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

Surfaces Rendering

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 glutInitDisplayMode(unsigned int mode);

  Specifies a display mode(color mode or buffer)

  A double-buffered and RGBA color mode window:

  glutInitDisplayMode(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, keyboard input
- 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* restablishes 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(zero) 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 returns
- 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 exaples

  torus, icosahedron, octahedron, cone, teapot

Important OpenGL Operations

Clearing the window

- Clear the color buffer filled by 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 framework

OpenGL Setup

1). Check: http://www.ece.lsu.edu/xinli/OpenGL/GLUTSetup.htm to download the precompiled libraries you need.

2). Download the "HelloWorld" program from:

http://www.ece.lsu.edu/xinli/OpenGL/program1.cpp

3). Create a Win32 console project, include this "program1.cpp", then compile and run it.

4). If you get linker errors or run-time errors, your system environment might not be compatible with the precompiled libraries. You might need to go back to 1) and download the source codes, compile them in your system. Then use the libraries (glut.h, glut32.lib, glut32.dll) newly generated.

Examples 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;
}
```

Simplified 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

- Create 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 (wind)

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

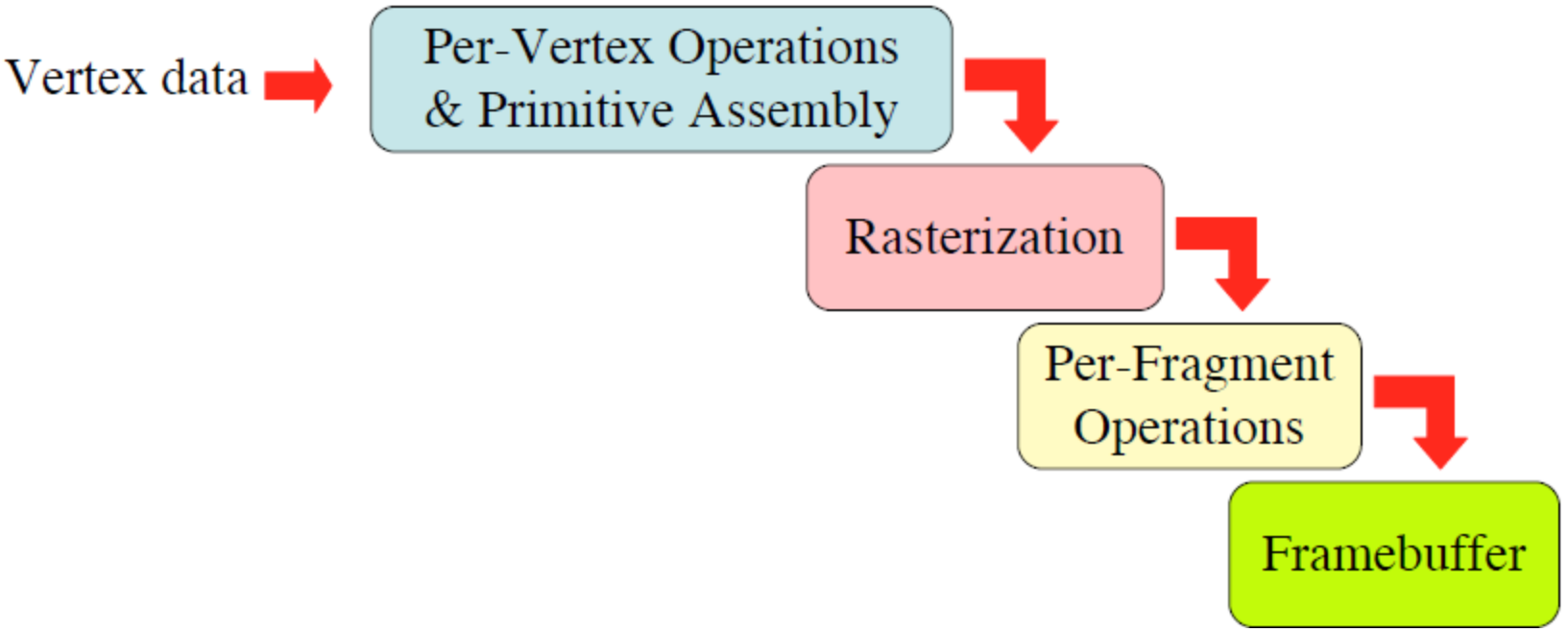
Many states can be turned on and off with

```
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.
- **Pre-fragment operations:** Hidden surface removal using the depth buffer (z buffer) or alpha blending for transparent materials
- **Framebuffer:** A collocation 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, ect.

OpenGL Basics: Summary

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

Resources:

There are many online resources about OpenGL:

1. The OpenGL official website http://www.opengl.org/ find coding resources, documentation, tutorials…

2. Nate Robins OpenGL website:
   http://www.xmission.com/~nate/index.html

3. OpenGL Tutorials at NeHe http://nehe.gamedev.net/

4. And so on…

Drawing Geometric Objects

Drawing Primitives

- OpenGL sets three types of drawing primitives

  Points

  Lines

  Polygons, e.g, triangles

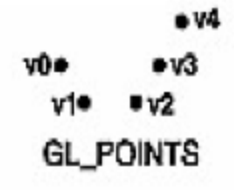
- All primitives are represented in terms of vertices

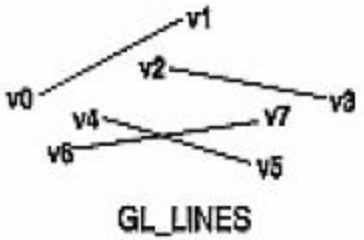
  that define the positions of the points themselves or the ends of line segments or the corners of polygons

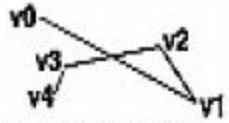
OpenGL Primitives

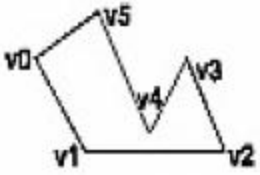
- Geometric object is described by a set of vertices (glVertex*) and the type of the primitive to be drawn
  - GL_POINTS
  - GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP
  - GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN
  - GL_QUADS, GL_QUAD_STRIP
  - GL_POLYGON

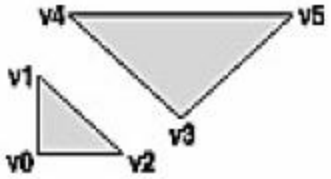
OpenGL Primitives

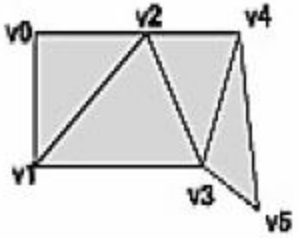


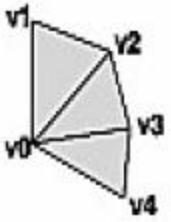


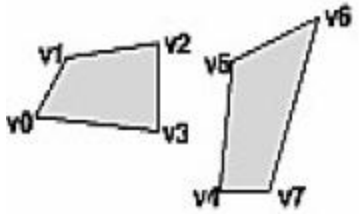


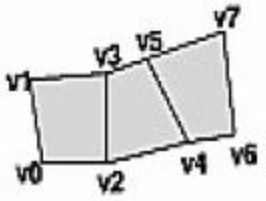


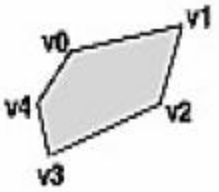












Points

- Objects of zeros dimension (infinitely small)
- Specified by a set of floating-point numbers (coordinates) called a vertex

- Displayed as a singled pixel on screen
- void glPointSize (Glfloat size);

  Sets the size of a rendered point in pixels

Specifying Vertices

- Void glVertex{234}{sidf}[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

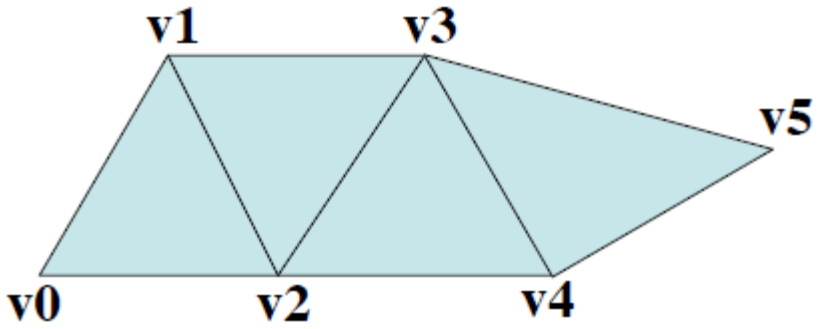
- GL_TRIANGLE_STRIP

  Draws a series of triangles using vertices in the order

  v0,v1,v2; v2,v1,v3

  v2,v3,v4; v4,v3,v5

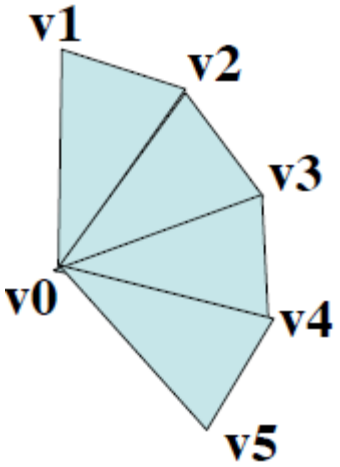
  All triangles are drawn with the same orientation (clockwise order)



- 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

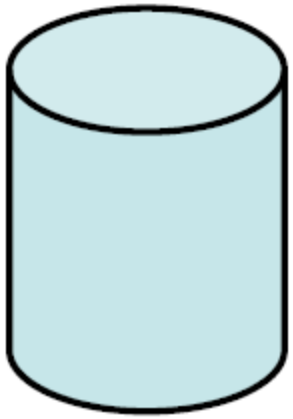
Finding Normal Vector

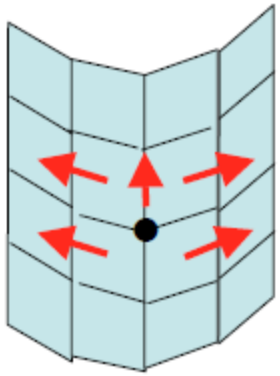
- Surfaces described with polygonal data

  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

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

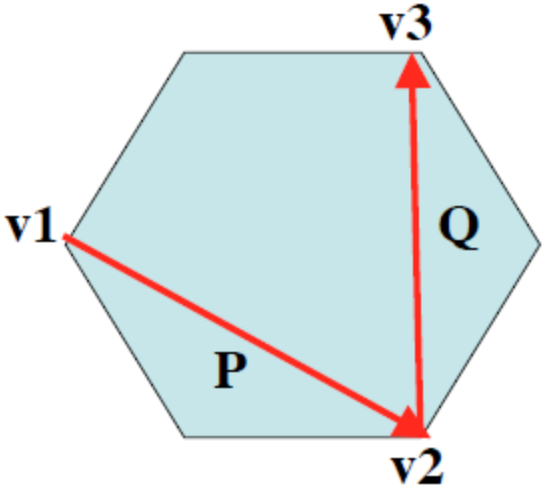
  $= [PyQz-QyPz)\ (QxPz-PxQz)\ (PxQy-QxPy]$

  $= [Nx\ Ny\ Nz]$

- 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*);

Step3: Dereferencing and Rendering

- void **glArrayElement**(GLint *ith*);

  Obtains the data of one (the *ith*) 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

Examples

- Polygonal approximation to a sphere