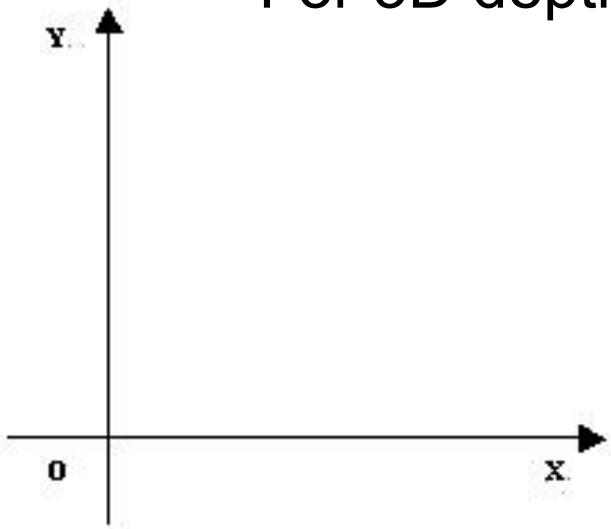
Graphics Output Primitives

Coordinate Reference Frames

– Screen coordinates
  - Locations on a video monitor in integer pixel locations
  - Scan line number (y=0 to y_{max})
  - Column number (x=0 to x_{max})
  - For 3D depth value normally stored in z



2D World-coordinate Reference Frame

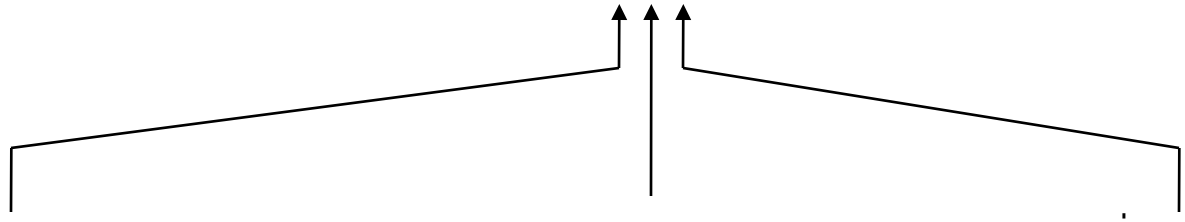
– Specify dimension and position for 2D window
  - Set projection matrix to identity
  - Orthogonal projection – maps vertices to screen if (x_v, y_v) are between (x_{min}, x_{max}) and (y_{min}, y_{max})

$$vertex(x_v, y_v, z_v) \Rightarrow screen(x_s, y_s),\ x_{\min} < x_v < x_{\max}, y_{\min} < y_v < y_{\max}$$

```
glMatrixMode( GL_PROJECTION );
glLoadIdentity( );
gluOrtho2D (xmin, xmax, ymin, ymax);
```

OpenGL Command Formats

```
glVertex3f(1.1, 2.0, 5.9)
glVertex3fv(v)
```



Number of Components

```
2 – (x,y)
3 – (x,y,z)
4 – (x,y,z,w)
```

Data Type

```
b  – byte
ub – unsigned byte
s  – short
us – unsigned short
i  – int
ui – unsigned int
f  – float
d  – double
```

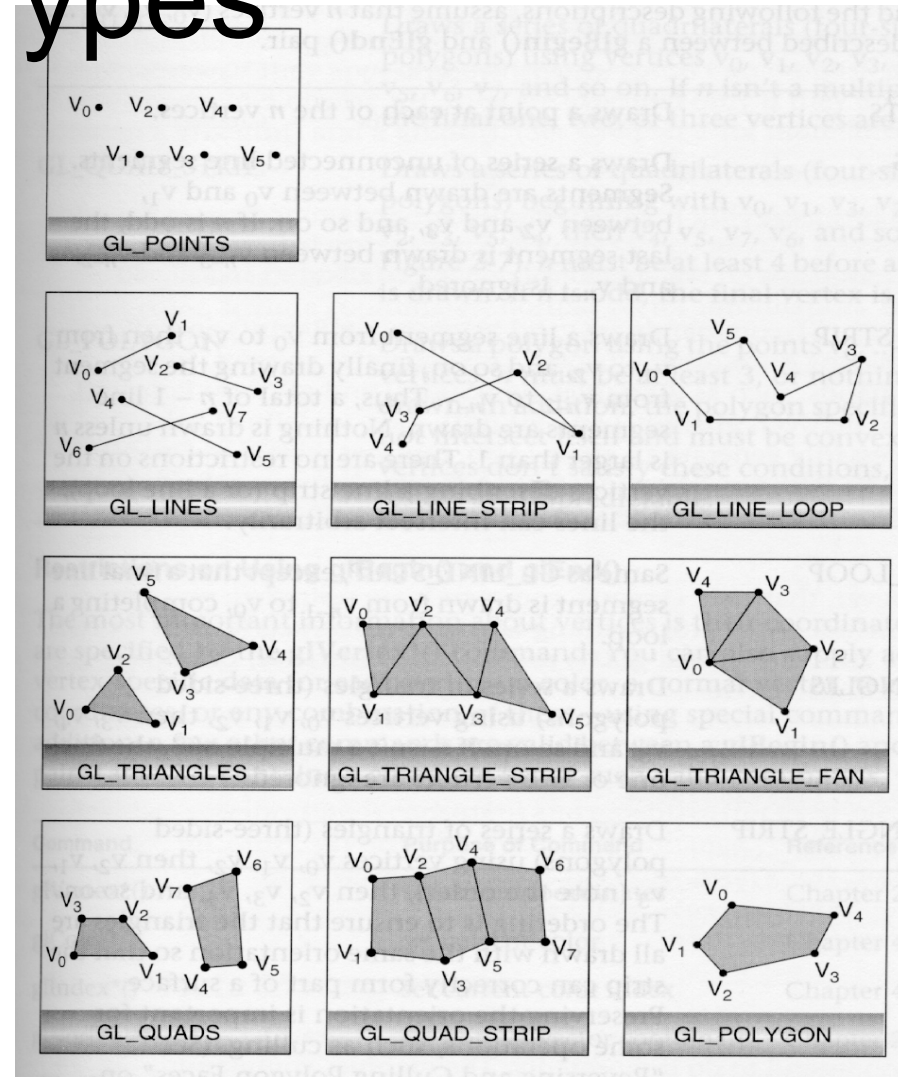
v is for Vector

```
float vertex[3];
glVertex3fv(vertex);

 Omit “v” for scaler:
  glVertex2f( x, y );
```

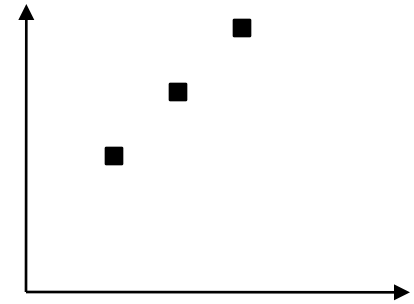
Primitive Types

- GL_POINTS
- GL_LINE
  - {S | _STRIP | _LOOP}
- GL_TRIANGLE
  - {S | _STRIP | _FAN}
- GL_QUAD
  - {S | _STRIP}
- GL_POLYGON



Point or Vertex

– Example: 3 points in a straight line
– All within projection (0..200)
– Window size (in pixels) not important



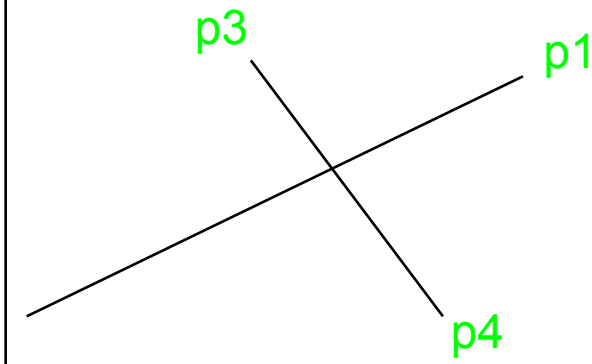
```
gluOrtho2D(0.0,200.0,0.0,200.0);
glBegin( GL_POINTS );
   glVertex2f( 50.0, 100.0 );
   glVertex2f( 75.0, 150.0 );
   glVertex2f( 100.0, 200.0);
glEnd();
```

Line

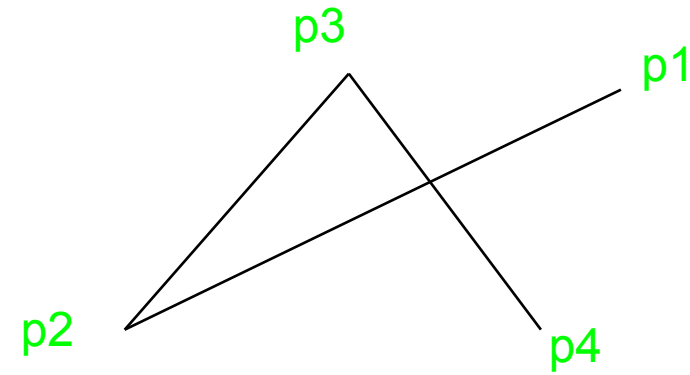
– Example: 2 lines

```
float p1[3],p2[3],p3[3],p4[3];

glBegin( GL_LINES );
   glVertex2fv( p1 );
   glVertex2fv( p2 );
   glVertex2fv( p3 );
   glVertex2fv( p4 );
glEnd();
```




```
glBegin( GL_LINE_STRIP );
   glVertex2fv( p1 );
   glVertex2fv( p2 );
   glVertex2fv( p3 );
   glVertex2fv( p4 );
glEnd();
```

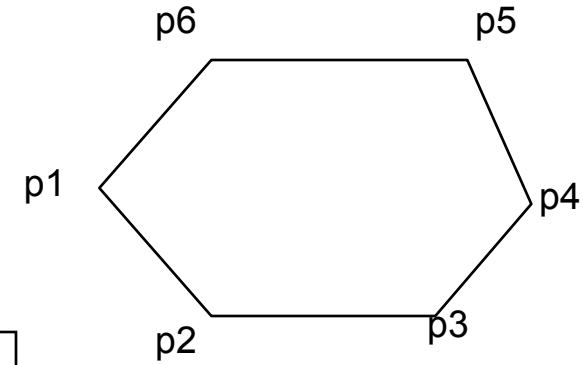


Fill area primitive

- Area filled with color or pattern
- Usually polygons
- Surface tessellation: approximating a curved shaped with a set of polygons (polygon mesh)

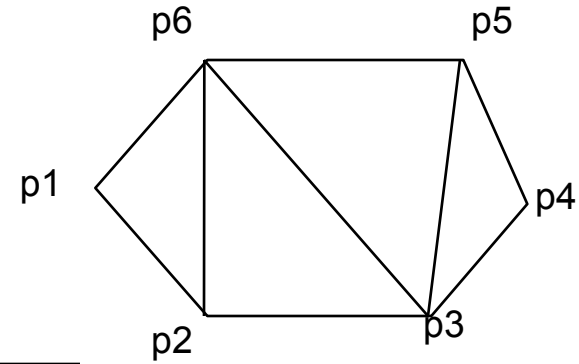
OpenGL Polygon Fill-Area

– Polygon example



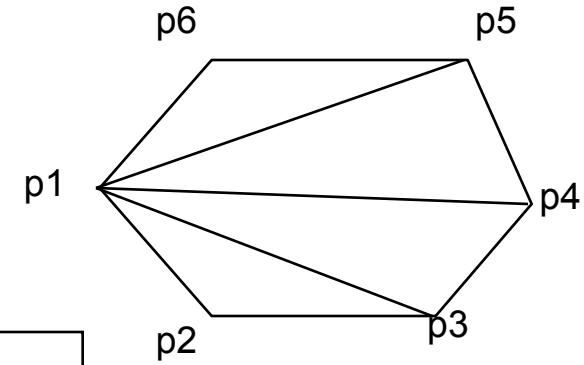

```
glBegin( GL_POLYGON );
   glVertex2fv( p1 );
   glVertex2fv( p2 );
   glVertex2fv( p3 );
   glVertex2fv( p4 );
   glVertex2fv( p5 );
   glVertex2fv( p6 );
glEnd();
```

– Triangle strips
  - Each successive triangles share an edge with previous defined triangle
  - First 3 in CCW order
  - N vertices => N-2 triangles



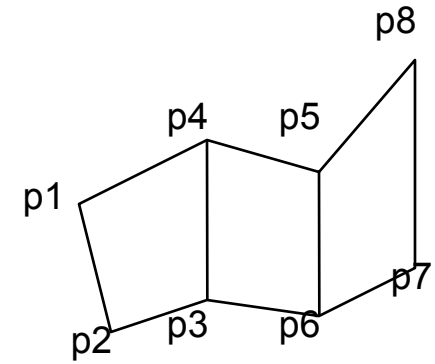

```
glBegin( GL_TRIANGLE_STRIP );
   glVertex2fv( p1 );
   glVertex2fv( p2 );
   glVertex2fv( p6 );
   glVertex2fv( p3 );
   glVertex2fv( p5 );
   glVertex2fv( p4 );
glEnd();
```

– Triangle fan
  - First vertex is the center of fan




```
glBegin( GL_TRIANGLE_FAN );
   glVertex2fv( p1 );
   glVertex2fv( p2 );
   glVertex2fv( p3 );
   glVertex2fv( p4 );
   glVertex2fv( p5 );
   glVertex2fv( p6 );
glEnd();
```

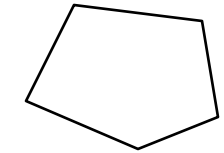
- Quad strip
- N vertices = N/2-1 quads



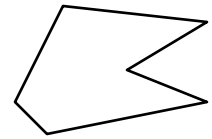

```
glBegin( GL_QUAD_STRIP );
   glVertex2fv( p1 );
   glVertex2fv( p2 );
   glVertex2fv( p4 );
   glVertex2fv( p3 );
   glVertex2fv( p5 );
   glVertex2fv( p6 );
   glVertex2fv( p8 );
   glVertex2fv( p7 );
glEnd();
```

Polygons

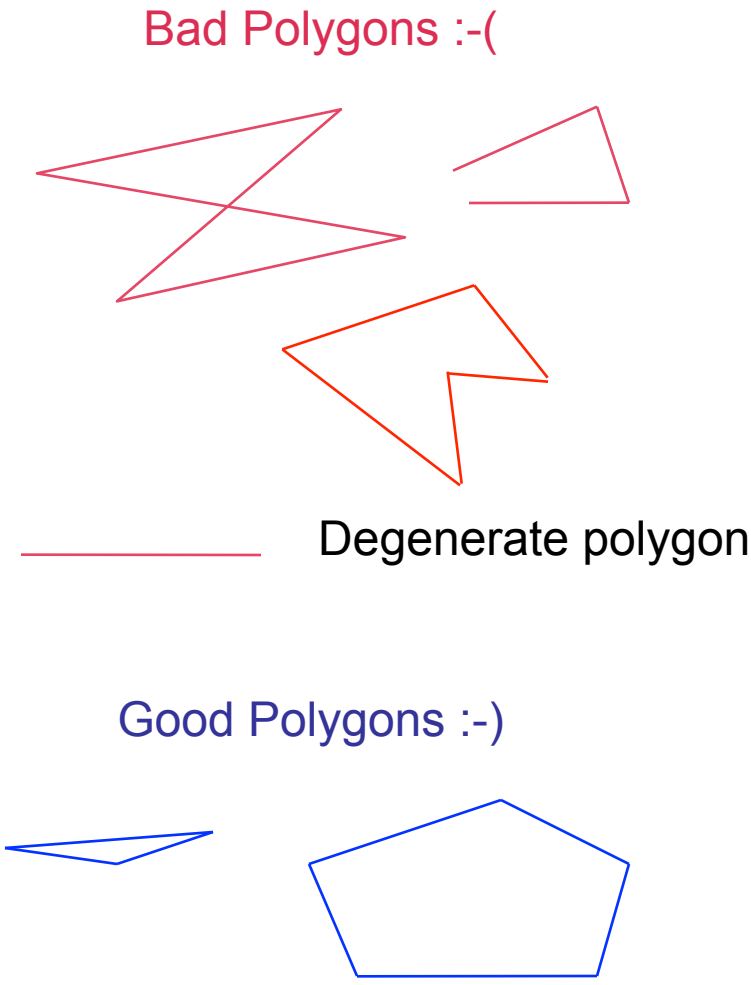
- Defined by a set of vertices and edges
  - Vertices given in either clockwise or counterclockwise order
  - Order defines front and back faces of objects – be consistent!



- Planarity: Automatic for triangles
- Four ways to test if a polygon is convex:
  - All interior angles of the polygon ≤ 180°
  - Polygon is on one side of any of the edges
  - Line segment connecting any two vertices is inside polygon
  - Cross product of any adjacent edges all point in same direction
- Otherwise, concave



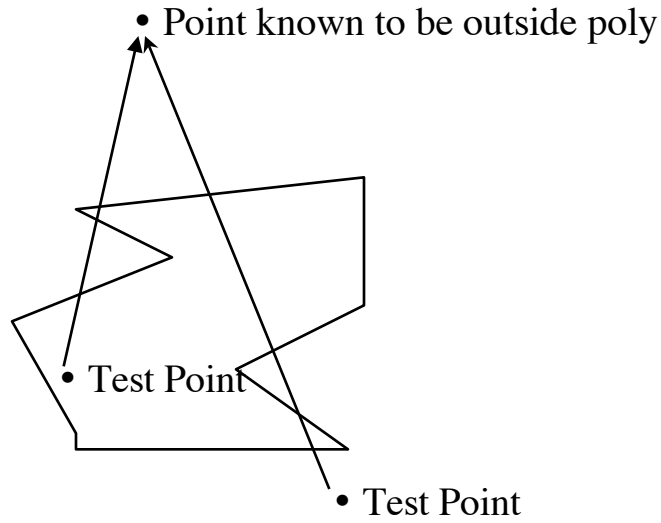
– Well behaved polygons:
  - No crossing edges
  - Planar
  - Closed
  - No edges repeated
  - Convex



Test if point Inside or Outside Poly

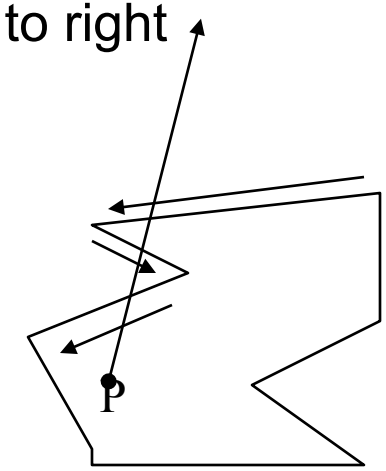
– Odd-even rule (also called odd-parity rule)

- Draw a line from test point to known outside point
  Crosses odd number of segments: inside
  Crosses even number of (or 0) segments: outside



– Nonzero winding-number rule

- Define an infinite ray from ‘P’ in any direction
- Count the number of times the boundary of an object “winds” around a point ‘P’ in the counterclockwise direction
- Add 1 whenever segment crosses the ray right to left
- Subtract 1 whenever segment crosses the ray left to right
- If winding-number non-zero, interior point
- In example on right winding-number = 1

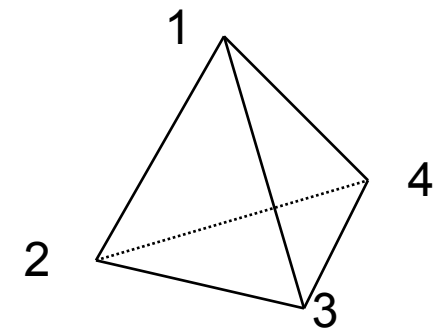


Polygonal geometry representation

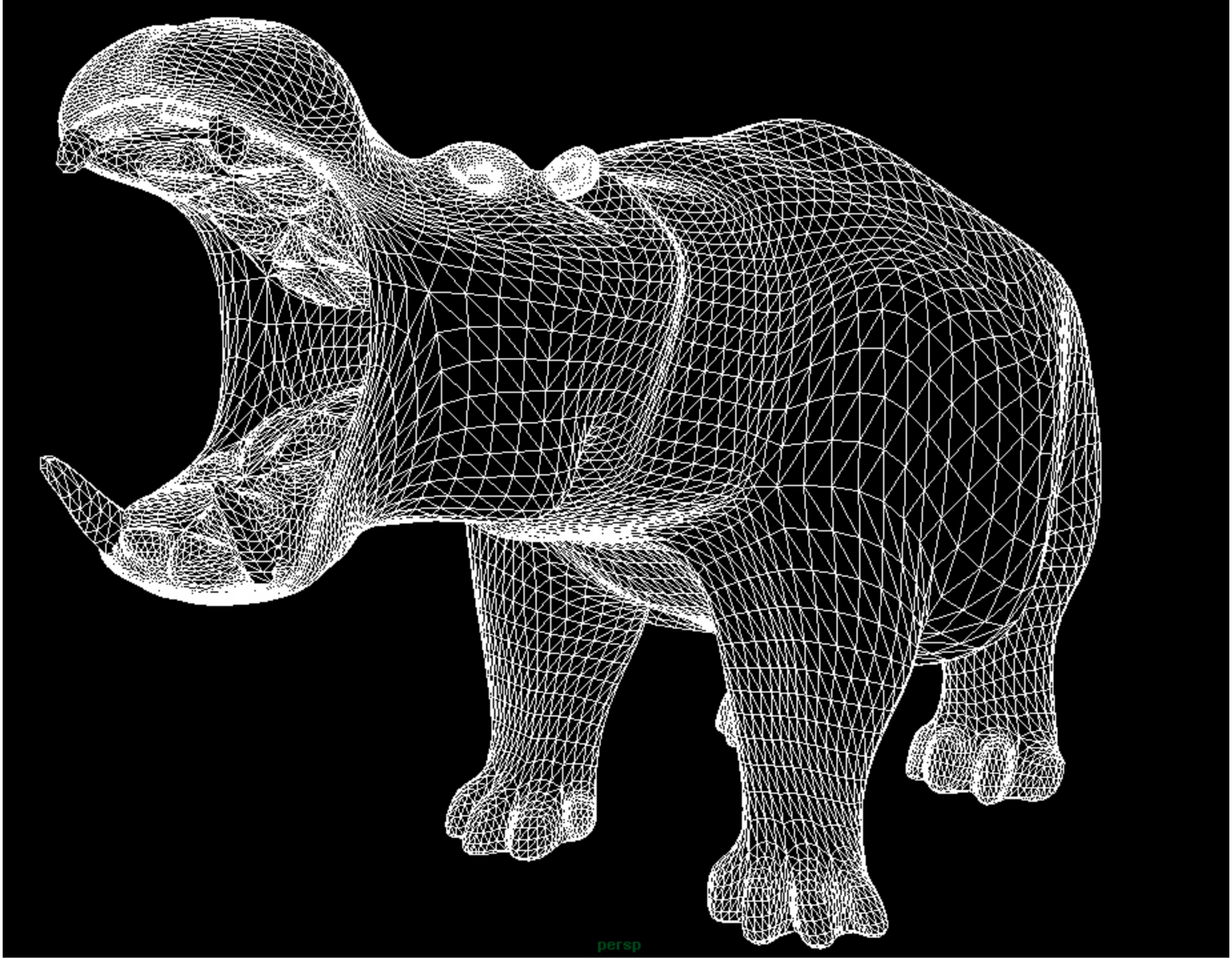
- Polygons represented by vertices given in clockwise or counterclockwise order
- Consist of vertices and edges
- Store the coordinates for each point in a table

Vertex Table

Vertex Number	x	y	z
1	0.0	0.0	0.0
2	-1.0	-2.0	1.0
3	1.0	-2.0	1.0
4	0.0	-2.0	-1.0



Tetrahedron – 4 triangles


persp

Explicit Model Representation

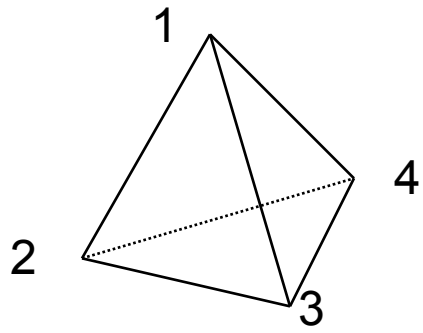
- $P_i = ((X_1,Y_1,Z_1),(X_2,Y_2,Z_2),\ldots(X_n,Y_n,Z_n))$
  - All in 1 table
- Shared vertices duplicated
- No representation of shared edges and vertices
- Must x-form each vertex and clip each edge of each polygon even though the vertices and edges are shared among polygons
- Draw each edge twice

Triangle Number	Vert pos 1	Vert pos 2	Vert pos 3
1	0.0, 0.0, 0.0	1.0, -2.0, 1.0	0.0, -2.0, -1.0
2	0.0, 0.0, 0.0	-1.0, -2.0, 1.0	1.0, -2.0, 1.0

Vertex and Polygon

- $V_i = ((X_1,Y_1,Z_1),\ldots(X_n,Y_n,Z_n))$
  - Vertex table
- $P_i = (V_1,V_2,\ldots,V_n)$
  - Polygon table
- Hard to find shared edges
- Edges drawn twice

Vertex #	x	y	z
1	0.0	0.0	0.0
2	-1.0	-2.0	1.0
3	1.0	-2.0	1.0
4	0.0	-2.0	-1.0



Triangle	Vertex	Vertex	Vertex
A	1	2	3
B	1	3	4
C	1	4	2
D	3	2	4

- This is the file format used for our models: http://www.icg.seas.gwu.edu/cs266/resources/model_osu/model_osu.html

Vertex, Edge, and Polygon

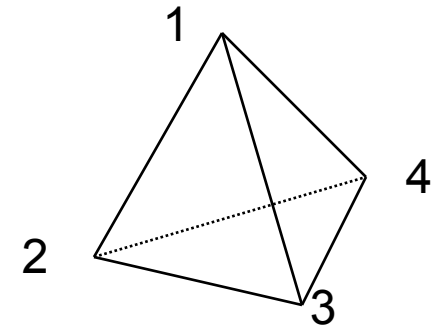
- $V_i = ((X_1,Y_1,Z_1),\ldots(X_n,Y_n,Z_n))$
  - Vertex table
- $E_i = (V_i,V_j)$
  - Edge table
  - Edges may have a list of polygons that use the edge
- $P_i = (E_1,E_2,\ldots, E_m)$
  - Polygon table
- Allows more error checking
  - Every vertex endpoint for two edges
  - Every edge in at least one polygon
  - Every polygon is closed
  - Each polygon has at lest one shared edge

Three table design

Vertex Number	x	y	z
1	0.0	0.0	0.0
2	-1.0	-2.0	1.0
3	1.0	-2.0	1.0
4	0.0	-2.0	-1.0

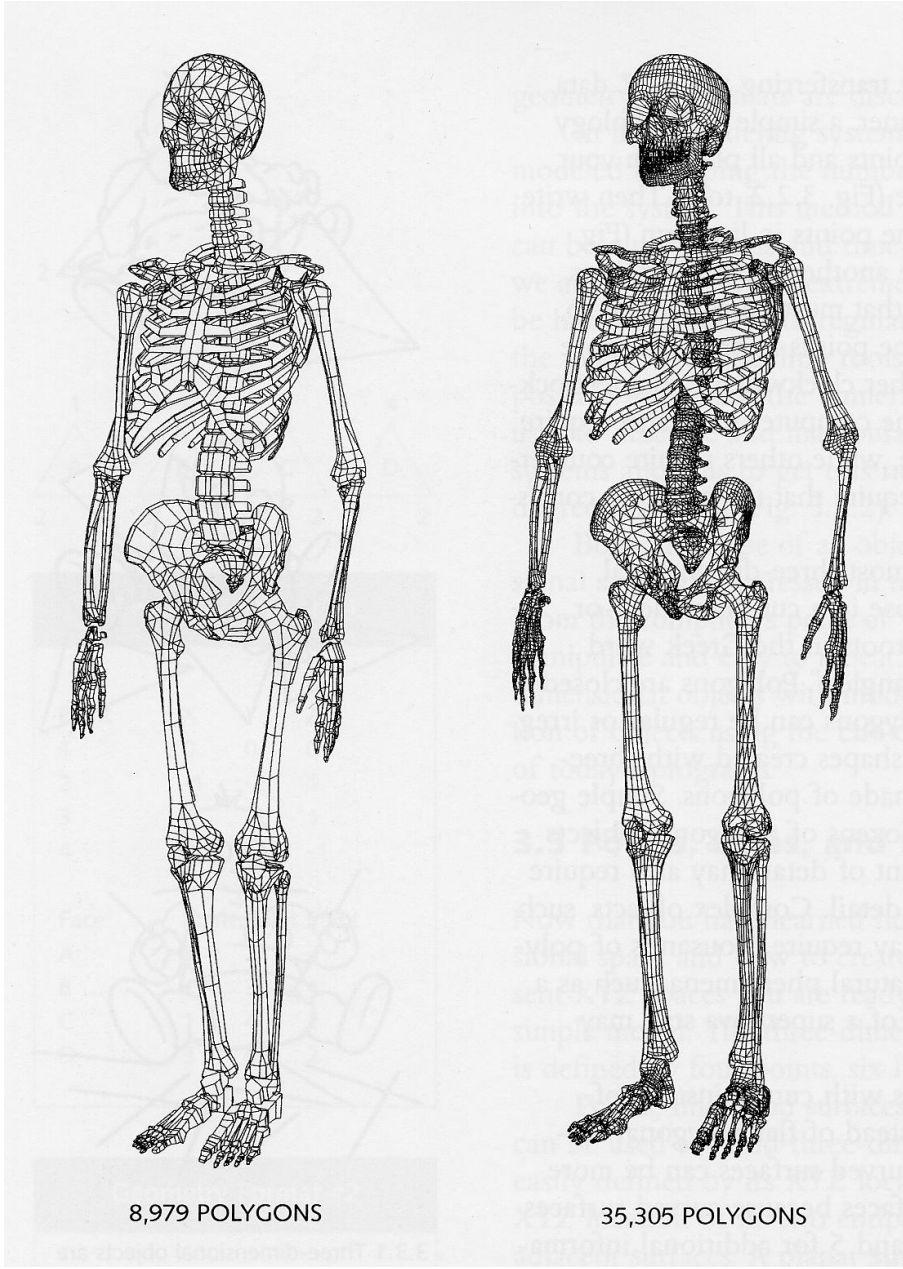
Edge #	Vert 1	Vert 2
1	1	4
2	1	3
3	3	4
4	3	2
5	1	2

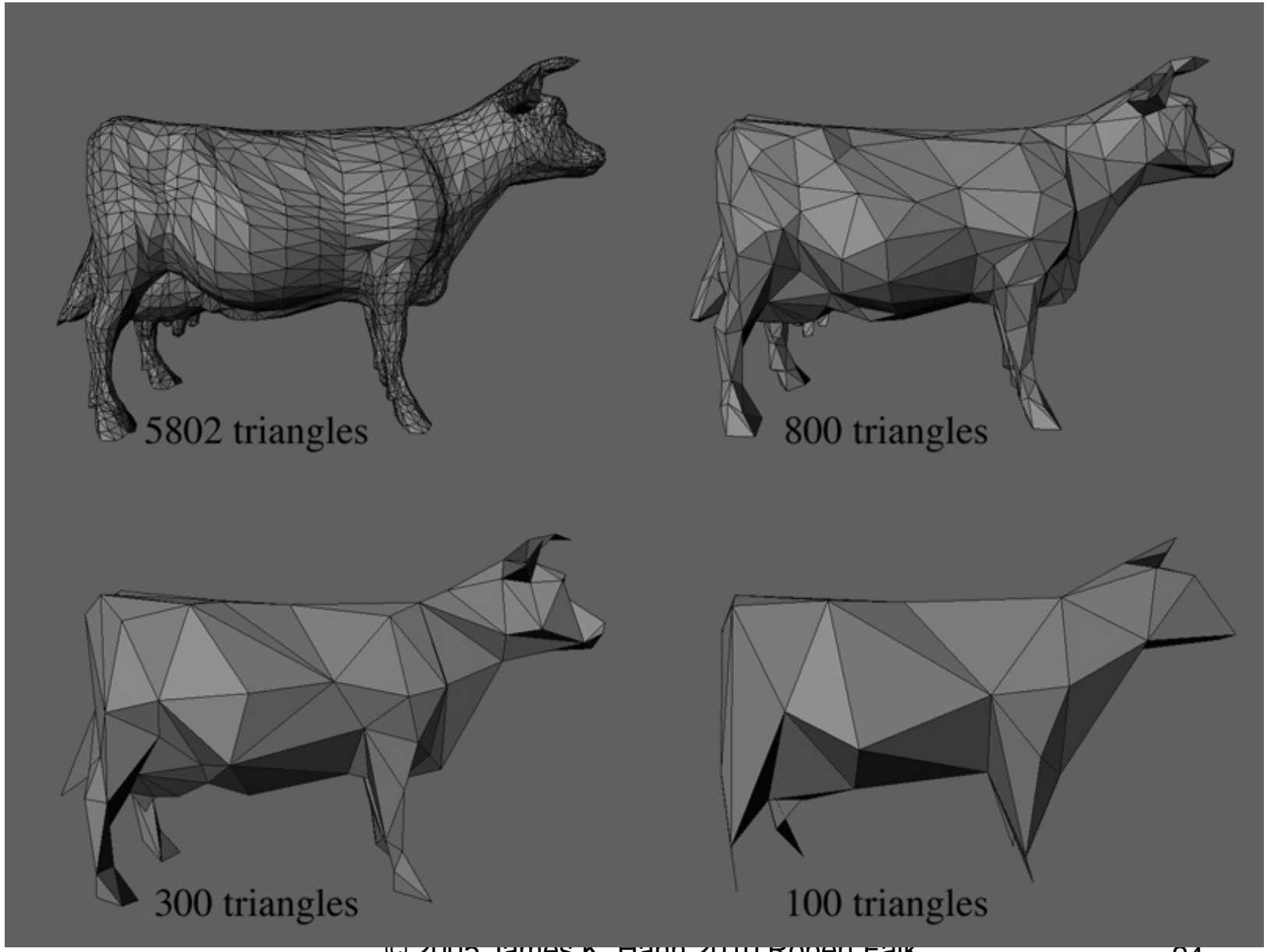
Face	Edge 1	Edge 2	Edge 3
1	1	2	3
2	2	5	4



Level of detail

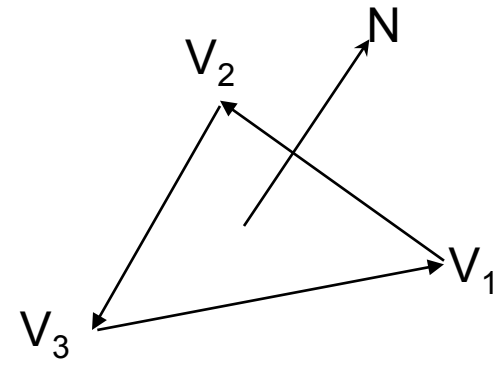
- For polygonal approximation
  - More polygons more accurate, but slower to render
- Necessary so that objects can be rendered at any distance
- May need to switch from one LOD to another (e.g. flight simulators, games)



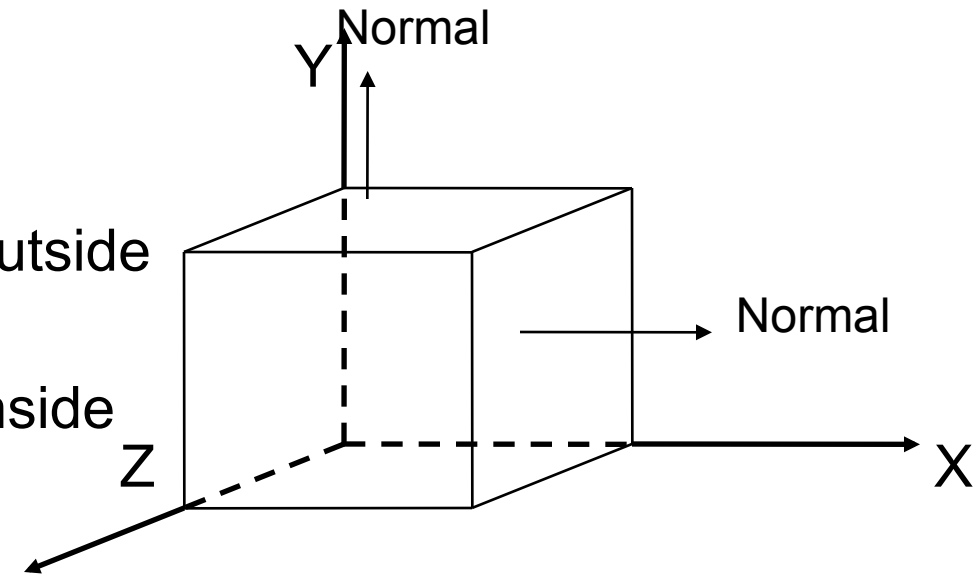


Plane equation

- $Ax + By + Cz + D = 0$
- (A, B, C) is the surface normal N of the plane
  - $N = (V_2 - V_1) \times (V_3 - V_2)$ which is then normalized
- D is gotten by substituting any point (x, y, z) on the plane
- Alternative equivalent formulation in the book (4-2)
  - Three plane equation from three successive vertices
  - Solve set of equations for A, B, C, D



Front and Back Polygon Face

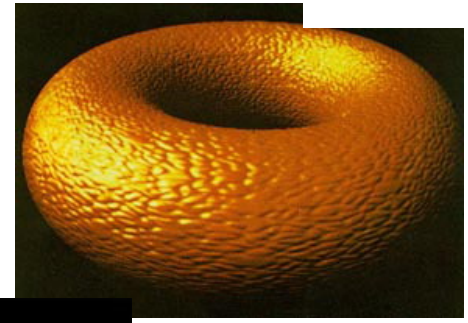


– A polygon has two sides
  - The side facing toward the outside of the object is “front facing”
  - The side facing toward the inside of the object is “back facing”
  - Not important for 2D
  - Important for 3D to eliminate back facing polygons
  - For closed polyhedra, back face culling eliminates unnecessary processing of faces that cannot be seen

– If we consider the “normal” of the polygon to point “outward” then
  - $Ax + By + Cz + D < 0$ then (x, y, z) is behind the plane
  - $Ax + By + Cz + D > 0$ then (x, y, z) is in front of the plane
  - In vector notation, n•p+d < 0 or n•p+d > 0

Icons of Computer Graphics (almost graphics primitives)

Pixar



University of Utah

Viewpoint Data Labs

Next: Attributes of Graphics Primitives