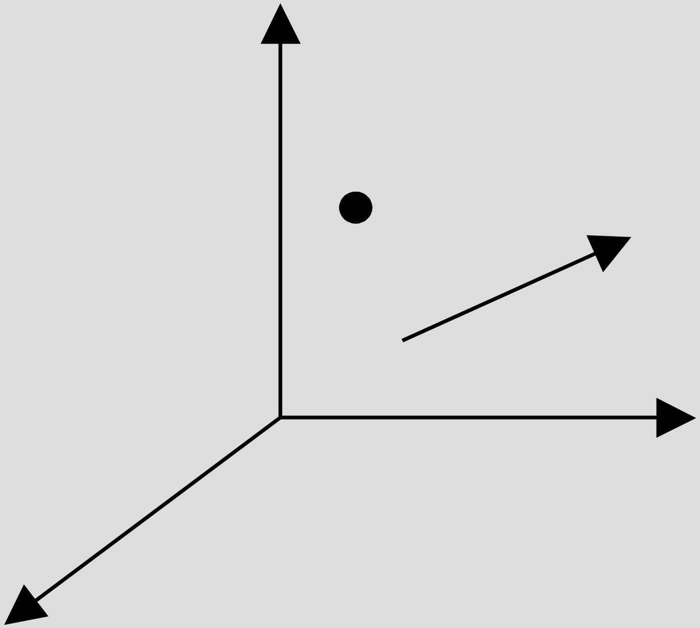
# Java 3D Geometry

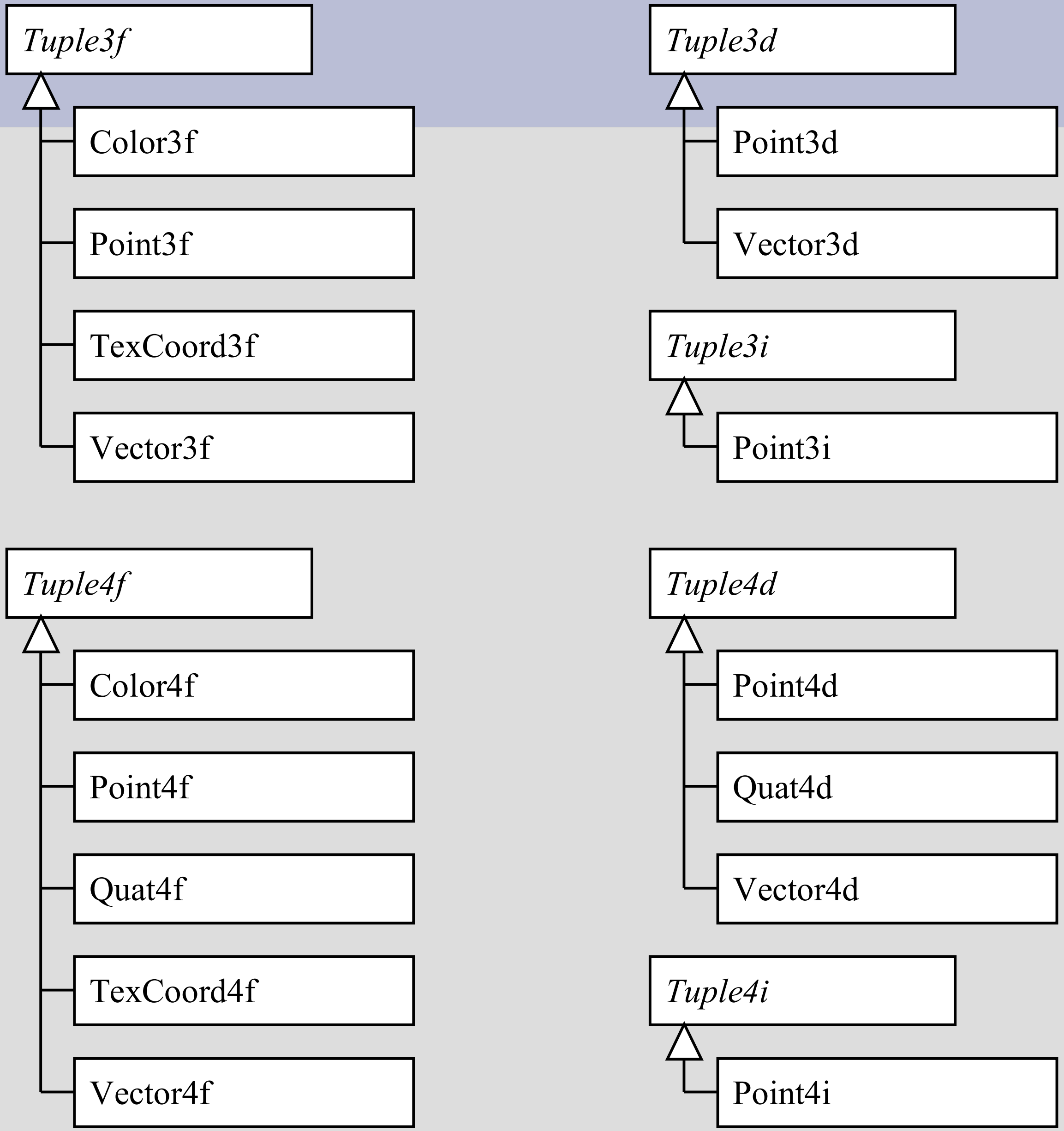
# Points and Vectors

$$\left(x_1, x_2, \ldots, x_n\right)$$



Homogeneous coordinates $\left(x, y, z, w\right)$

# Points and Vectors



# Java's javax.vecmath package

- javax.vecmath package
  - Classes related to vectors and matrices
- Java 3D makes extensive use of these classes
- Naming convention of classes
  - Class names end with:  [34][fdib]
    - The 3 or 4 indicates how many components
    - The fdib indicates types used
      - The f,d,i,b are for float, double, int, and byte
  - Tuple* are abstract base classes
  - Color* are for colors
  - Point* and Vector* are geometric points and vectors
  - TexCoord* are for texture-mapping coordinates
  - Quat* are for quaternions

# The vector classes' methods

- Methods for standard operations
- The “Tuple” base classes
  - Methods: add and sub for adding and subtracting tuples
  - Method scale for scaling a tuple
  - Method negate negates the tuple's components
- The “Point” classes
  - Methods for finding distance to other points
- The “Vector” classes
  - Methods dot and angle computes dot product and angle with another vector
  - Method cross computes cross product of 2 vectors
  - Method length computes length of vector

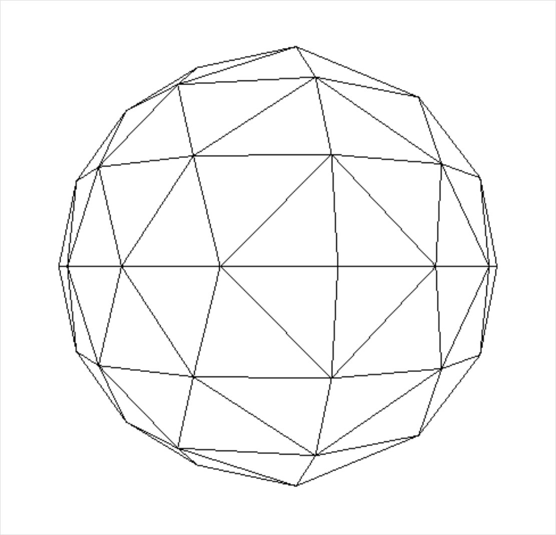
# Surface Equations

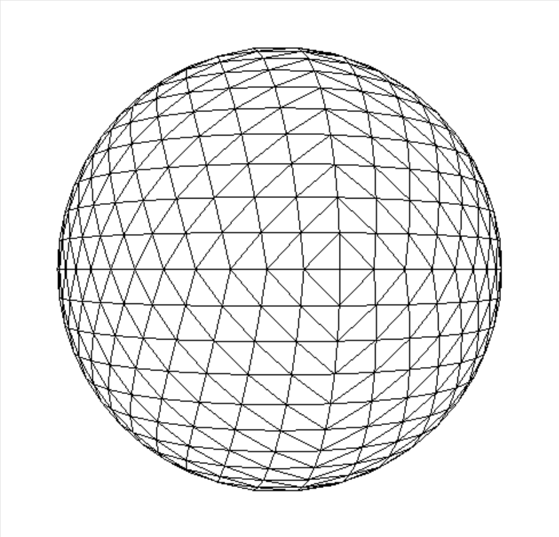
Implicit equation $F(x, y, z) = 0$

Parametric equation

$$x = f(u, v)$$
$$y = g(u, v)$$
$$z = h(u, v)$$

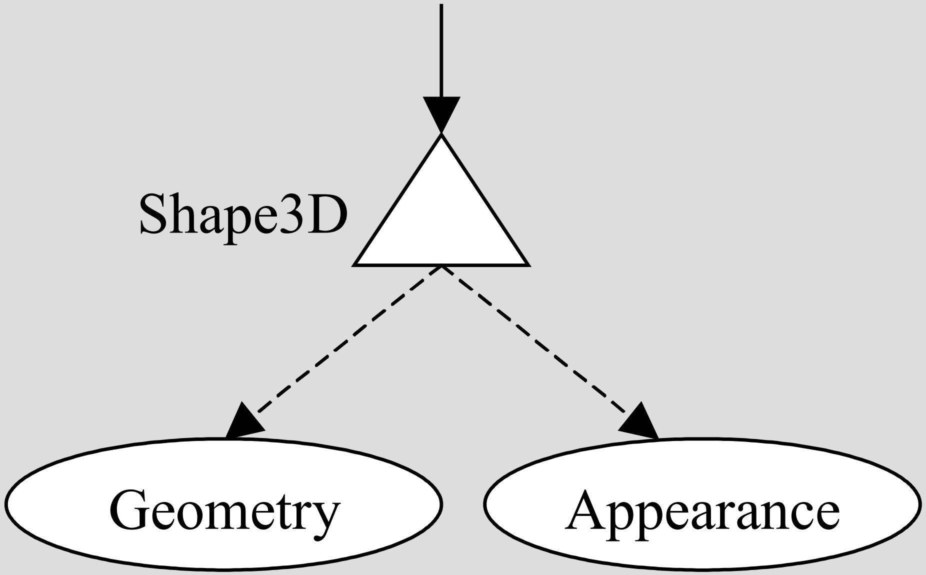
# Surface Represented with Polygons





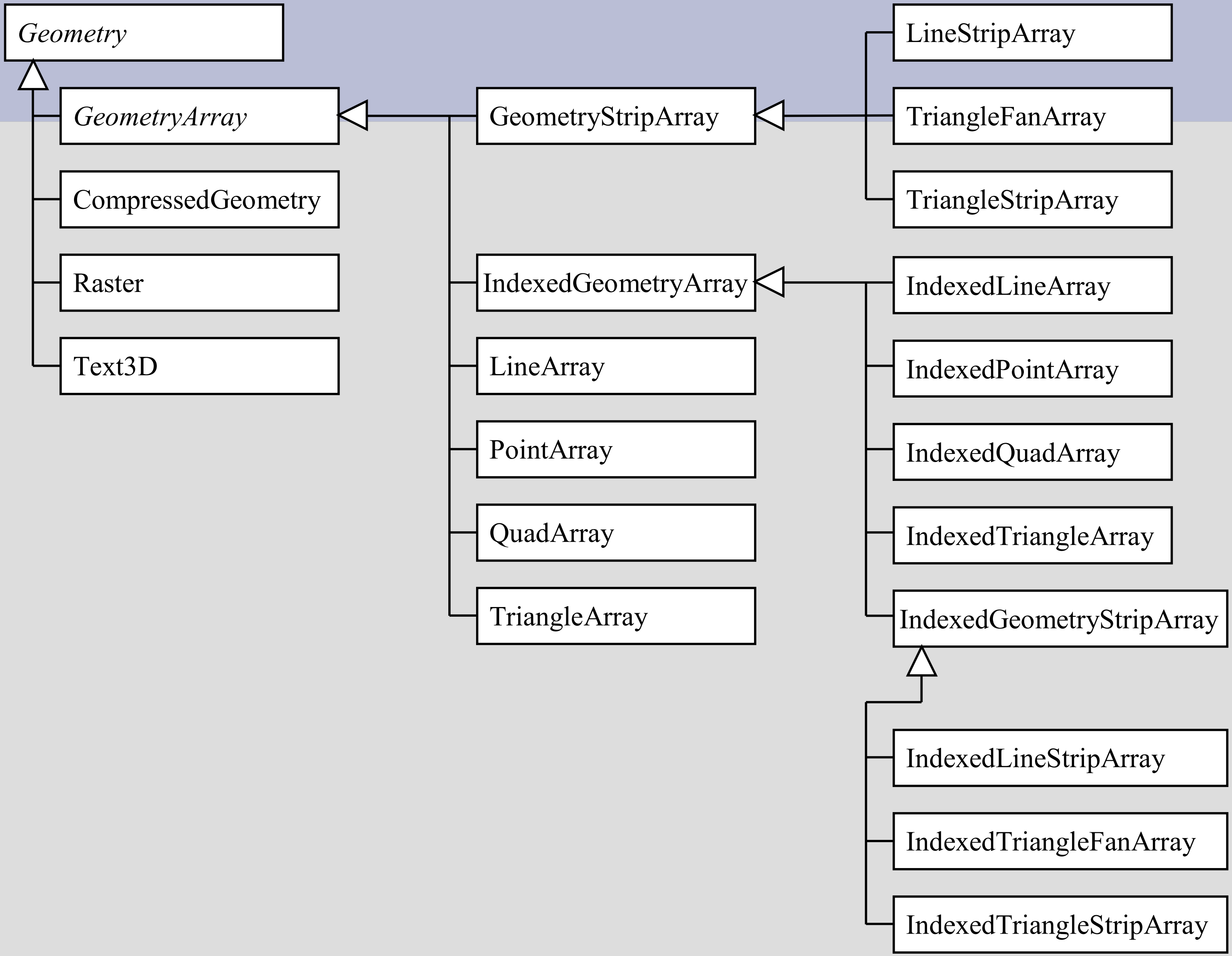
Complex surfaces approximated with a mesh of polygons
- e.g., a mesh of triangles or quadrilaterals

# Shape3D Node

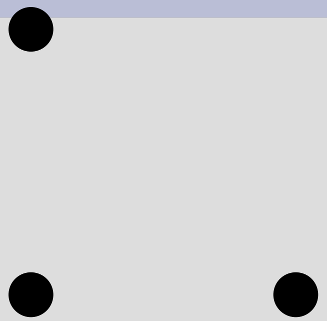


A Shape3D leaf node usually references Geometry and Appearance objects

# Geometry Classes



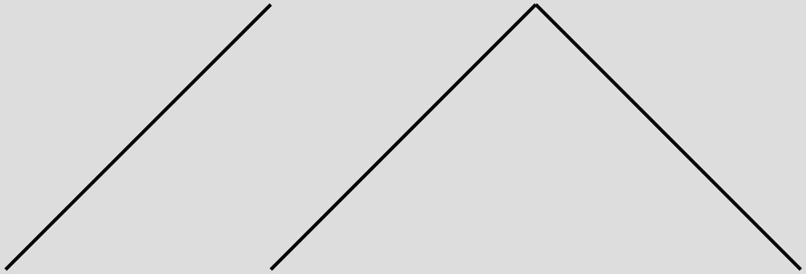
# PointArray



```
PointArray pa = new PointArray(3, GeometryArray.COORDINATES);
pa.setCoordinate(0, new Point3f(0f, 0f, 0f));
pa.setCoordinate(1, new Point3f(1f, 0f, 0f));
pa.setCoordinate(2, new Point3f(0f, 1f, 0f));
```

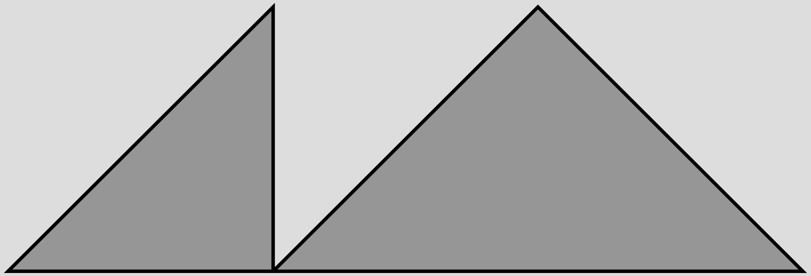
- A bit mask indicating type of vertex data
- Can also include: NORMALS, COLOR_3, COLOR_4, and some texture related properties

# LineArray



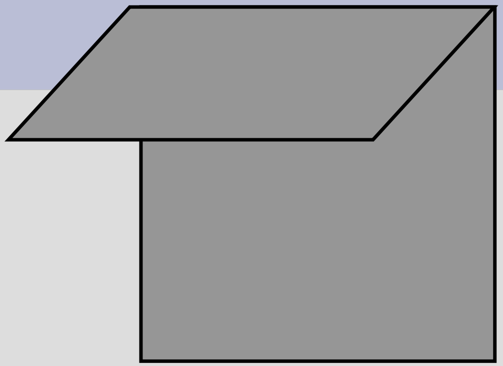
```
LineArray la = new LineArray(6, GeometryArray.COORDINATES);
Point3f[] coords = new Point3f[6];
coords[0] = new Point3f(0f, 0f, 0f);
coords[1] = new Point3f(1f, 1f, 0f);
coords[2] = new Point3f(1f, 0f, 0f);
coords[3] = new Point3f(2f, 1f, 0f);
coords[4] = new Point3f(2f, 1f, 0f);
coords[5] = new Point3f(3f, 0f, 0f);
la.setCoordinates(0, coords);
```

# TriangleArray



```
TriangleArray ta = new TriangleArray(6,
GeometryArray.COORDINATES);
Point3f[] coords = new Point3f[6];
coords[0] = new Point3f(0f, 0f, 0f);
coords[1] = new Point3f(1f, 1f, 0f);
coords[2] = new Point3f(1f, 0f, 0f);
coords[3] = new Point3f(1f, 0f, 0f);
coords[4] = new Point3f(2f, 1f, 0f);
coords[5] = new Point3f(3f, 0f, 0f);
ta.setCoordinates(0, coords);
```

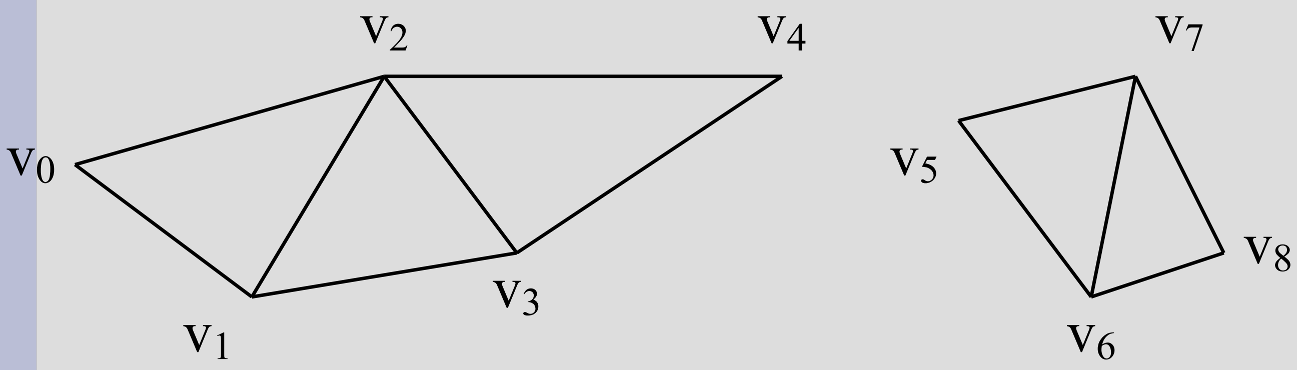
# QuadArray



```
QuadArray qa = new QuadArray(8, GeometryArray.COORDINATES);
Point3f[] coords = new Point3f[8];
coords[0] = new Point3f(0f, 0f, 0f);
coords[1] = new Point3f(1f, 0f, 0f);
coords[2] = new Point3f(1f, 1f, 0f);
coords[3] = new Point3f(0f, 1f, 0f);
coords[4] = new Point3f(1f, 1f, 0f);
coords[5] = new Point3f(0f, 1f, 0f);
coords[6] = new Point3f(0f, 1f, 1f);
coords[7] = new Point3f(1f, 1f, 1f);
qa.setCoordinates(0, coords);
```

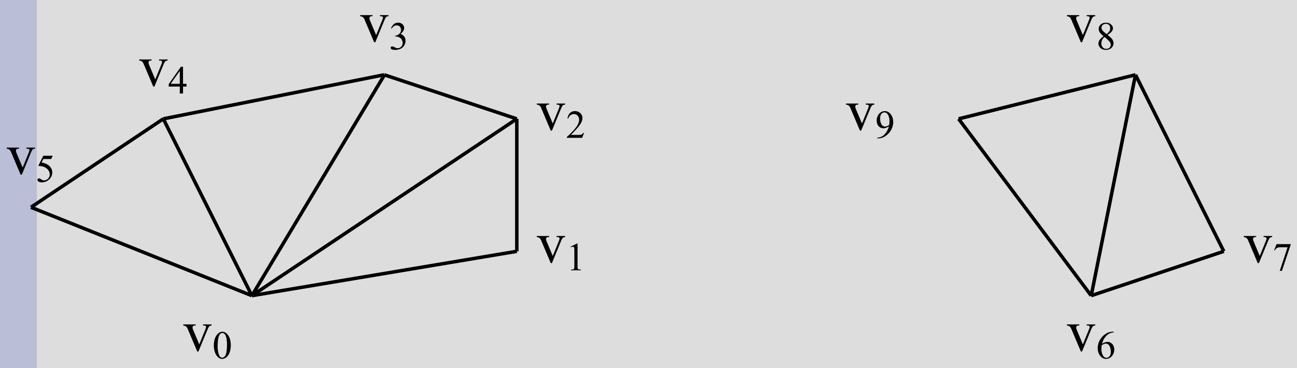
# StripArray

TriangleStripArray



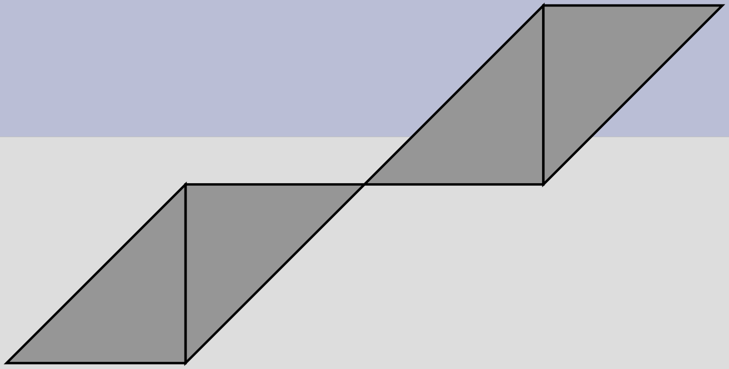
strip vertex counts: 5, 4

TriangleFanArray



strip vertex counts: 6, 4

# IndexedArray



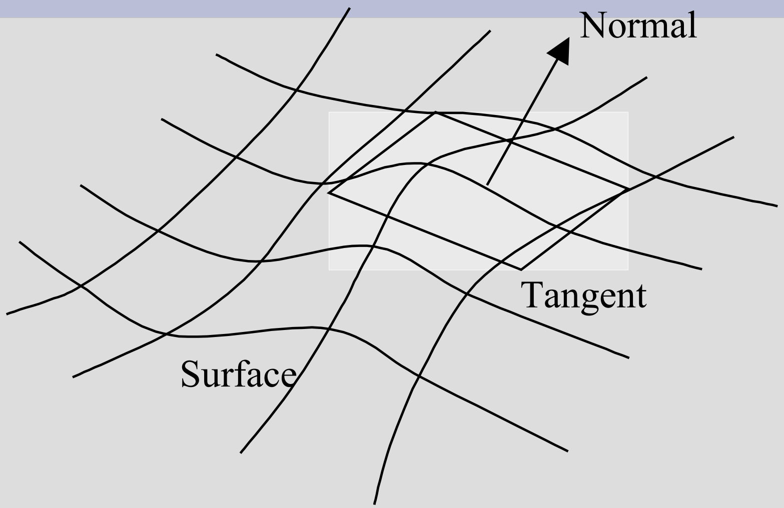
```
int[] stripIndexCounts = {4, 4};
IndexedTriangleStripArray itsa = new IndexedTriangleStripArray(7,
  GeometryArray.COORDINATES, 8, stripIndexCounts);
Point3f[] coords = new Point3f[7];
coords[0] = new Point3f(0f, 0f, 0f);
coords[1] = new Point3f(0f, 1f, 0f);
coords[2] = new Point3f(1f, 1f, 0f);
coords[3] = new Point3f(2f, 1f, 0f);
coords[4] = new Point3f(-1f, 0f, 0f);
coords[5] = new Point3f(-1f, -1f, 0f);
coords[6] = new Point3f(-2f, -1f, 0f);
itsa.setCoordinates(0, coords);
int[] indices = {0, 1, 2, 3, 0, 4, 5, 6};
itsa.setCoordinateIndices(0, indices);
```

# The Tetrahedron

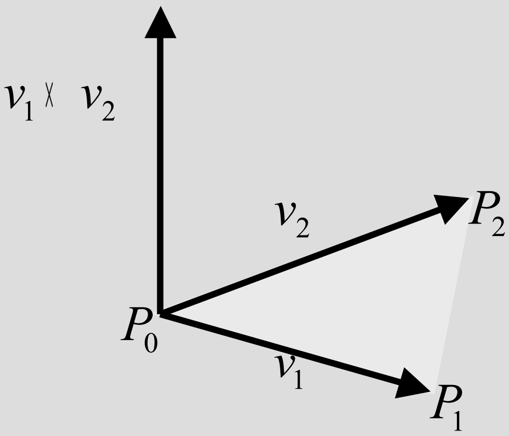
- One of five regular polyhedra
- Vertices:
  - (1, 1, 1), (1, -1, -1), (-1, 1, -1), (-1, -1, 1)
- Indices:
  - 0,1,2,   0,3,1,   1,3,2,   2,3,0
- Normals:
  - (1, 1, -1), (1, -1, 1), (-1, -1, -1), (-1, 1, 1)

# Surface Normals

The surface normal at a point is perpendicular to the tangent plane



The cross product is useful for calculating normals



# Normal Calculation for a Smooth Surface

Parametric equation

$$x = f(u,v)$$
$$y = g(u,v)$$
$$z = h(u,v)$$

Derivatives

$$(dx/du, dy/du, dz/du) = (f_u, g_u, h_u)$$
$$(dx/dv, dy/dv, dz/dv) = (f_v, g_v, h_v)$$

Normal

$$n = (f_u, g_u, h_u) \times (f_v, g_v, h_v)$$

# Normal Calculation for a Geometric Object with Planar Surfaces

- Given 3 distinct points on the plane, $P_0$, $P_1$, and $P_2$.
- Can define 2 vectors in the plane with:
  - $V_1 = P_{1 -} P_0$.
  - $V_2 = P_{2 -} P_0$.
- The normal for the plane is then:
  - $V_1 \times V_2$
- Assume p0, p1, and p2 are Point3f

```
p1.sub(p0);
p2.sub(p0);
Vector3f v1 = new Vector3f(p1);
Vector3f v2 = new Vector3f(p2);
Vector3f normal = new
   Vector3f();
normal.cross(v1,v2);
normal.normalize();
```

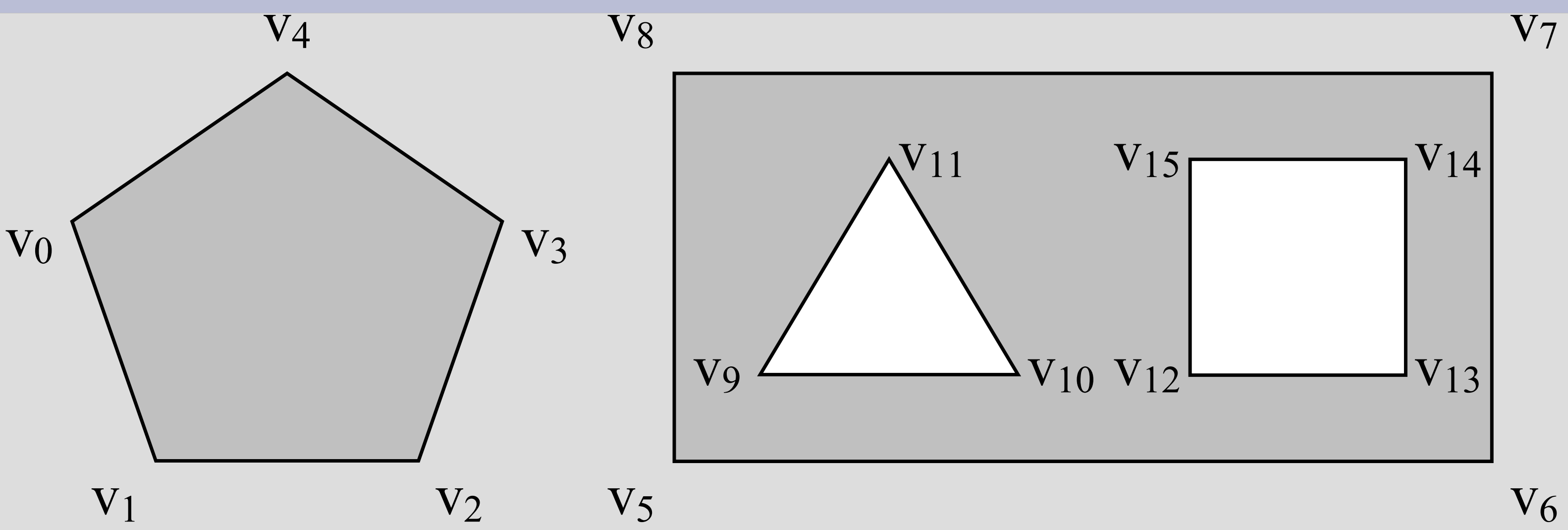
# GeometryInfo

- Last time we saw one way to specify 3D geometry
  - Extend one of the classes in the Geometry hierarchy
    - e.g., IndexedTriangleArray
  - Pass an object of your class to the constructor of Shape3D
- Another more general way of defining 3D geometry is to use the GeometryInfo class

# GeometryInfo

- Why use GeometryInfo?
  - In addition to Triangle and Quadrilateral arrays, can also specify geometry with Polygon arrays
  - Can use a NormalGenerator to automate the generation of the normal vectors
  - Can use a Stripifier to turn the geometry into a polygon strip array.
- Process:
  - Extend Shape3D
    - Construct a GeometryInfo object
      - Specify the geometry
    - Use the NormalGenerator and the Stripifier
    - Call the setGeometry method of Shape3D
  - Construct an object of your shape class

# GeometryInfo Class



strip counts: 5, 4, 3, 4 countour counts: 1, 3

Utility classes

NormalGenerator
Stripifier

# Polygon Mesh

We can define a 3D surface with a parametric equation in 2 independent variables

x = f(u,v)
y = g(u,v)
z = h(u,v)

with a <= u <= b and c <= v <= d

Vertices

$$u_i = a + i(b - a)/m, \quad i = 0,1,2,\ldots,m$$

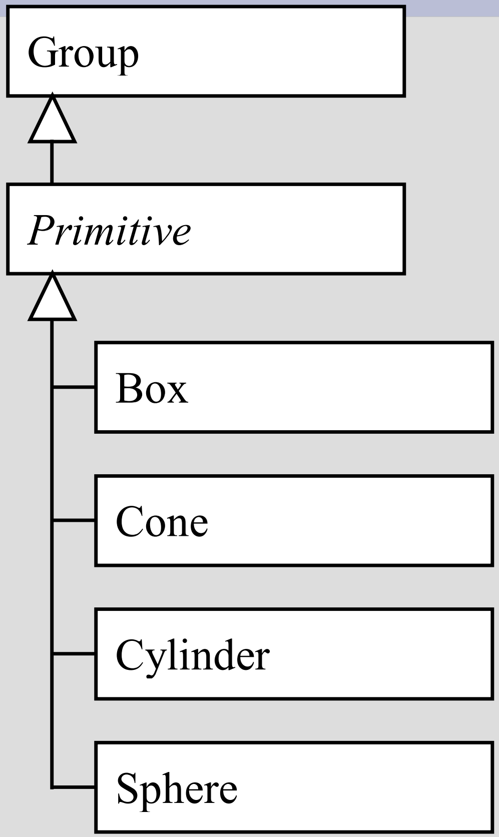
$$v_j = c + j(d - c)/n, \quad j = 0,1,2,\ldots,n$$

Quadrilateral path

$$(u_i, v_j),(u_{i+1}, v_j),(u_{i+1}, v_{j+1}),(u_i, v_{j+1})$$

Each quadrilateral patch can be further divided into 2 triangles

# Primitives



# Font and Text

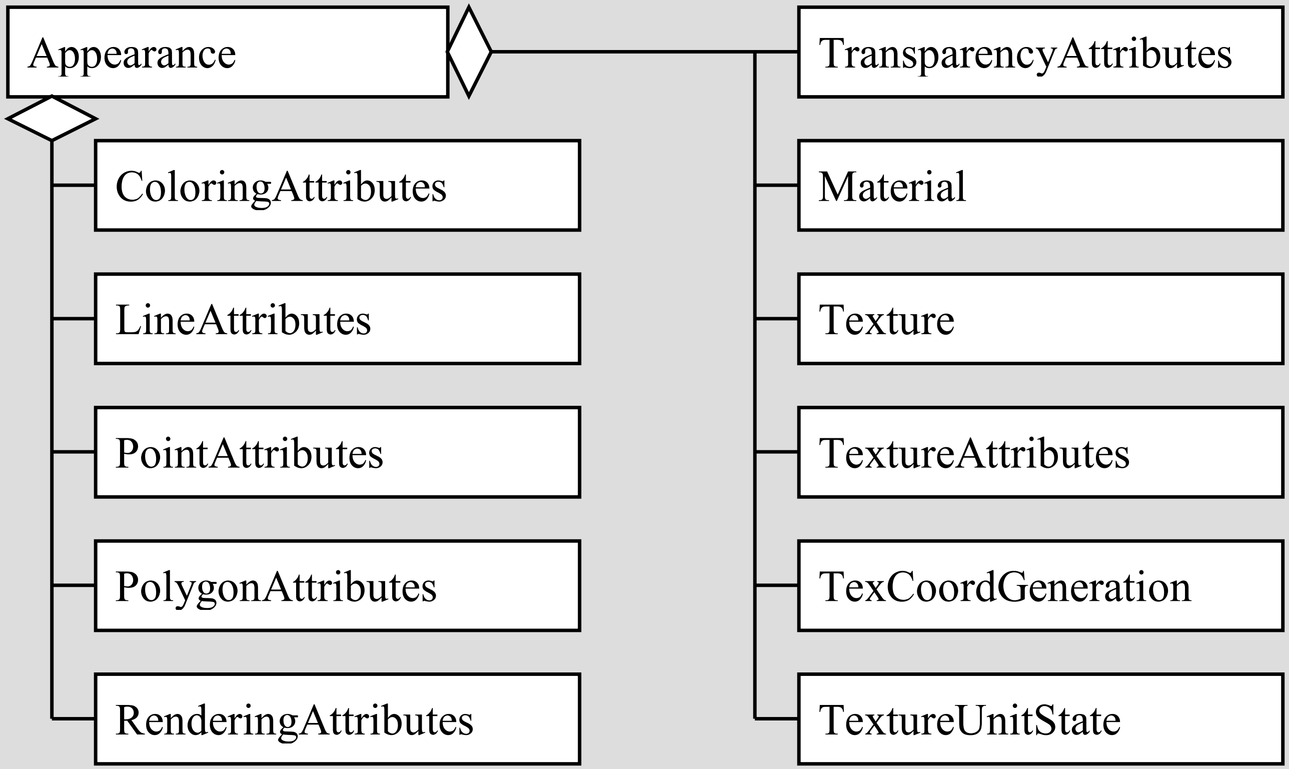
Create a Text3D object

```
Font font = new Font(“Serif”, Font.BOLD, 1);
FontExtrusion extrusion = new FontExtrusion();
Font3D font3d = new Font3D(font, extrusion);
Text3D text = new Text3D(font3d, “Hello”);
```

Create a Text2D object

```
Text2D text = new Text2D(“Hello”, Color.blue,
“Serif”, 16, Font.Italic);
```

# Appearance Classes



Appearance

ColoringAttributes

LineAttributes

PointAttributes

PolygonAttributes

RenderingAttributes

TransparencyAttributes

Material

Texture

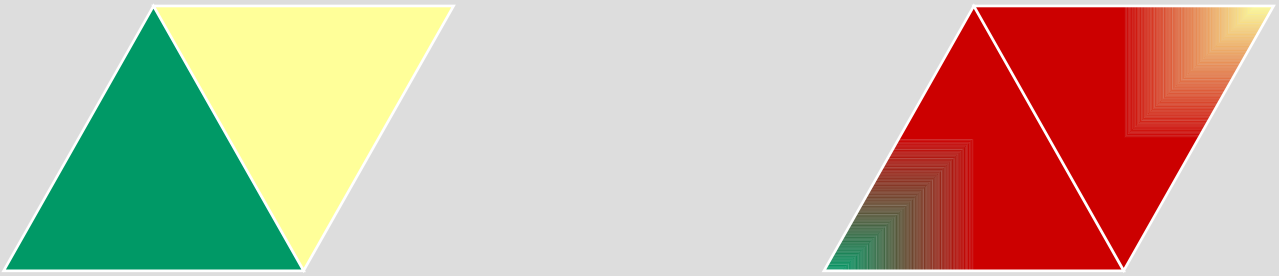
TextureAttributes

TexCoordGeneration

TextureUnitState

# Shading Model

- Flat shading: a fixed color for a face
- Gouraud shading: interpolating vertex colors



# Coloring

- The lighting model is applied if the Appearance references a valid Material object and the Material object enables lighting.
- If vertex colors are present and not ignored, they are used to render the polygons. The enabling of the vertex colors is controlled by a RenderingAttributes object. When vertex colors are used, the shading mode of the polygons is determined by the ColoringAttributes object. A flat shading assigns a single color to a polygon and a Gouraud shading interpolates the vertex colors in the interior of a polygon.
- If lighting is not enabled and the vertex colors of the geometry are not present or ignored, the color specified by the ColoringAttributes object will be used for coloring the geometry.