Java3D: Geometry

- The basic node for shape data is ***Shape3D***.
- A ***Shape3D*** node is a container for a ***Geometry*** node and an ***Appearance*** node.
- A ***Shape3D*** object has the following constructors and mutators

```
Shape3D ();
Shape3D (Geometry geo);
Shape3D (Geometry geo,
            Appearance app);
void setGeometry (Geometry geo);
void setAppearance (Appearance app);
Geometry getGeometry ();
Appearance getAppearance ();
```

Geometry (contd.)

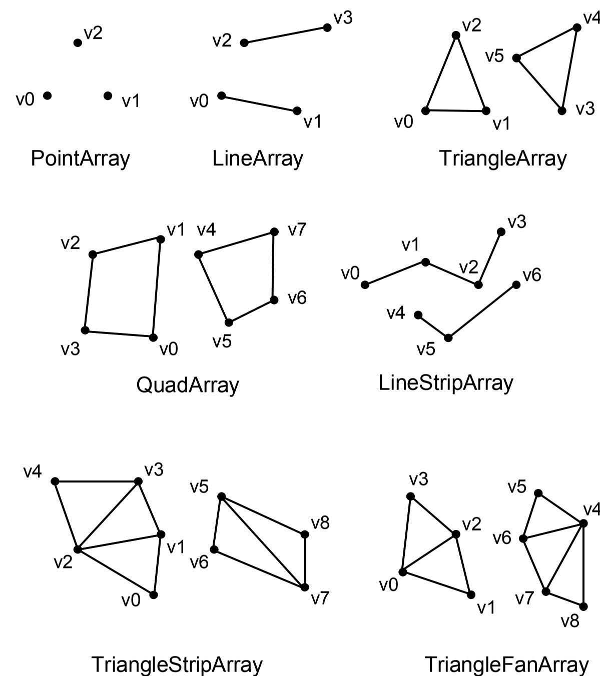
- A ***Shape3D*** can have more than one ***Geometry*** associated with it.
- The ***Geometry*** objects for a ***Shape3D*** are stores as a list.
  - Methods that affect a single geometry, affect the geometry at index 0.
- Methods to manage the list of geometries are:

```
void addGeometry (Geometry geo);
void insertGeometry (Geometry geo, int index);
void removeGeometry (int index);
void setGeometry (Geometry geo, int index)
java.util.Enumeration getAllGeometries ();
Geometry getGeometry (int index);
int numGeometries ()
```

GeometryArrays

- ***GeometryArray*** node components are used to create points, lines and polygons.
- These shape primitives are specified by the location of each *vertex*.
- ***GeometryArray*** is subclassed as follows.
  - GeometryArray
    - PointArray
    - LineArray
    - TriangleArray
    - QuadArray
    - GeometryStripArray
      - LineStripArray
      - TriangleFanArray
    - IndexedGeometryArray
      - IndexedPointArray
      - IndexedLineArray
      - IndexedTriangleArray
      - IndexedQuadArray
      - IndexedGeometryStripArray
        - » IndexedLineStripArray
        - » IndexedTriangleFanArray
        - » IndexedTriangleStripArray

Basic Geometry Types



Vertices

- All the primitive geometry types are built from *vertices*.
  - Every vertex has a coordinate that specifies its location in 3D space.
  - A vertex may contain other optional coordinates.
    - Normals
    - Colours
    - Texture
- For 2D geometries the order of the vertices is important. Normals define the orientation of the surface. The normal defines the “front” surface and the vertices are ordered in counter-clockwise direction around the normal.
- Java3D uses the **javax.vecmath** package to help specify vector data. The name of the classes in this package identify the data type, dimension and precision.
  - Point3d specifies a 3D point object defined using double data.
  - Color4b specifies a colour using 4 bytes.
  - Vector3f specifies a 3D vector object defined using float data (Vector and TexCoord classes are only allowed to use float values).

Vertices (contd.)

- A point can be defined by either:

```
Point3f point =
      new Point3f (1.0f, 1.0f, 1.0f);
```

or

```
Point3f point = new Point3f ();
point.set (1.0f, 1.0f, 1.0f);
```

or

```
Point3f point = new Point3f ();
point.x = 1.0;
point.y = 1.0;
point..z = 1.0;
```

GeometryArray classes

- Contructors:
  PointArray (int vertexCount, int vertexFormat);
  LineArray (int vertexCount, int vertexFormat);
  TriangleArray (int vertexCount, int vertexFormat);

- The format value indicates the type of data associated with each vertex.
  - The format value is the ORing of static integer fields.
    - COORDINATES
    - NORMAL
    - COLOR_3
    - COLOR_4
    - TEXTURE_COORDINATE_2
    - TEXTURE_COORDINATE_3

  GeometryArray.COORDINATE |
  GeometryArray.NORMAL |
  GeometryArray.TEXTURE_COORDINATE_2

- The data for the vertex coordinates can be set by:
  void setCoordinates (int length, Point3f coord);
  void setCoordinates (int length, Point3f [ ] coords);
  void setCoordinates (int length, Point3f [ ] coords,
  int start, int length);

- There are similar methods for setting colour and texture.

Example:PointArray

```
PointArray pa =
    new PointArray (3, PointArray.COORDINATES);

Point3f [ ] pts = new Point3f [3];
pts[0] = new Point3f (0.0f, 0.0f, 0.0f);
pts[1] = new Point3f (1.0f, 0.0f, 0.0f);
pts[2] = new Point3f (0.0f, 1.0f, 0.0f);

pa.setCoordinates (0, pts);
```

Example:LineArray

```
LineArray la =
    new LineArray (3,
                    LineArray.COORDINATES |
                    LineArray.COLOR_3);

Point3f [ ] pts = new Point3f [4];
pts[0] = new Point3f (0.0f, 0.0f, 0.0f);
pts[1] = new Point3f (1.0f, 0.0f, 0.0f);
pts[2] = new Point3f (0.0f, 1.0f, 0.0f);
pts[3] = new Point3f (1.0f, 1.0f, 0.0f);

Color3f clrs = new Color3f [4];
clrs[0] = new Color3f (0.0f, 0.0f, 0.0f); // black
clrs[1] = new Color3f (1.0f, 1.0f, 1.0f); // white
clrs[2] = new Color3f (1.0f, 0.0f, 0.0f); // red
clrs[3] = new Color3f (0.0f, 1.0f, 0.0f); // green

la.setCoordinates (0, pts);
La.setColors (0, clrs);
```

Example:TriangleArray

```
TriangleArray ta = new TriangleArray (3,
                TriangleArray.COORDINATES |
                TriangleArray.COLOR_3 |
                TriangleArray.NORMALS);

Point3f [ ] pts = new Point3f [3];
pts[0] = new Point3f (0.0f, 0.0f, 0.0f);
pts[1] = new Point3f (1.0f, 0.0f, 0.0f);
pts[2] = new Point3f (0.0f, 1.0f, 0.0f);

Color3f clrs = new Color3f [3];
clrs[0] = new Color3f (1.0f, 0.0f, 0.0f);      // red
clrs[1] = new Color3f (0.0f, 1.0f, 0.0f);      // green
clrs[2] = new Color3f (0.0f, 0.0f, 1.0f);      // blue

Vector3f [ ] norms = new Vector3f [3];
Vector3f triNormal = new Vector3f (0.0f, 0.0f, 1.0f);
norms[0] = triNormal;
norms[1] = triNormal;
norms[2] = triNormal;

ta.setCoordinates (0, pts);
ta.setColors (0, clrs);
ta.setNormals (0, norms);
```

Geometry Strips

- Constructors

```
LineStripArray (int vertexFormat,
                int vertexCount,
                int [ ] stripVetexCounts);
TriangleStripArray (int vertexFormat,
                    int vertexCount,
                    int [ ] stripVetexCounts);
TriangleFanArray (int vertexFormat,
                  int vertexCount,
                  int [ ] stripVetexCounts);
```

- The *stripVertexCounts* array defines the number of strips and the length of each strip.

```
int stripLength = new int [2];
stripLength [0] = 3;
stripLength [1] = 2;
LineStripArray lsa =
    new LineStripArray (5, LineArray.COORDINATES,
                        stripLength);

Point3f [ ] pts = new Point3f [5];
pts[0] = new Point3f (0.0f, 0.0f, 0.0f);
pts[1] = new Point3f (1.0f, 0.0f, 0.0f);
pts[2] = new Point3f (0.0f, 1.0f, 0.0f);
pts[3] = new Point3f (1.0f, 1.0f, 0.0f);
pts[4] = new Point3f (0.0f, 0.0f, 0.0f);

lsa.setCoordinates (0, pts);
```

Indexed Geometry

- Indexed geometry is useful when vertices are shared. A classic example is a cube.

```
Point3f [ ] pts = new Point3f [8];
pts[0] = new Point3f (-1.0f, -1.0f, -1.0f);
pts[1] = new Point3f (1.0f, -1.0f, -1.0f);
pts[2] = new Point3f (-1.0f, 1.0f, -1.0f);
pts[3] = new Point3f (-1.0f, -1.0f, 1.0f);
pts[4] = new Point3f (-1.0f, 1.0f, 1.0f);
pts[5] = new Point3f (1.0f, -1.0f, 1.0f);
pts[6] = new Point3f (1.0f, 1.0f, -1.0f);
pts[7] = new Point3f (1.0f, 1.0f, 1.0f);

int [ ] indices = { 0, 3, 4, 2,      // left
                    0, 1, 5, 3,      // bottom
                    0, 2, 6, 1,      // back
                    7, 5, 1, 6,      // right
                    7, 6, 2, 4,      // top
                    7, 4, 3, 5       // front
                  };

IndexedQuadArray iqa = new IndexedQuadArray (8,
    GeometryInfo.COORDINATES);
iqa.setCoordinates (0, pts);
iqa.setCoordinateIndices (0, indices);
```

Geometry Utilities

- Instead of using ***GeometryArray*** to specify a geometry we can use ***GeometryInfo***. ***GeometryInfo*** contains several tool classes that generate a ***GeometryArray*** for complex geometries that we can use in the scene graph.
- The tool classes are:
  - ***Triangulator*** which breaks up complex polygons into triangles.
  - ***NormalGenerator*** which calculates vertex normal for a set of polygons.
  - ***Stripifier*** which turns a mesh of triangles into triangle strips to improve performance.

GeometryInfo

- Constructor:

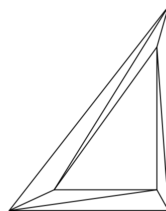
  GeometryInfo (int primType)

  where the primitive type is one of:

  POLYGON_ARRAY
  QUAD_ARRAY
  TRIANGLE_ARRAY
  TRIANGLE_FAN_ARRAY
  TRIANGLE_STRIP_ARRAY
- Quad and Triangle arrays are defined and used similarly to ***GeometryArray*** except:
  - The number of vertices is set implicitly when you define the geometry.
  - The entire array must be set together; it is not possible to set a range of coordinates starting at an index.
- The POLYGON_ARRAY is used to specify polygons with more than 4 vertices or contain holes.
  - Each polygon is made from one or more contours. The first contour specifies the outer boundary of the polygon. Each subsequent contour specifies a hile in the interior of the polygon.

Triangulator

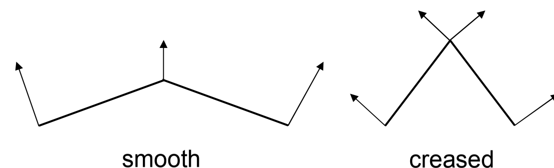
- The ***Triangulator*** breaks complex polygons into triangles.



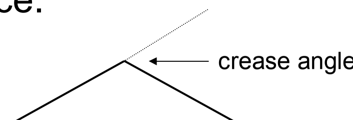
```
int stripCounts = new int [2];
stripCounts [0] = 3;
stripCounts [1] = 3;
int contourCounts = new int [1];
contourCounts [0] = 2;

Point3f [ ] pts = new Point3f [6];
pts[0] = new Point3f (-1.0f, -1.0f, 0.0f);
pts[1] = new Point3f (1.0f, -1.0f, 0.0f);
pts[2] = new Point3f (1.0f, 1.0f, 0.0f);
pts[3] = new Point3f (-0.6f, -0.8f, 0.0f);
pts[4] = new Point3f (0.8f, 0.6f, 0.0f);
pts[5] = new Point3f (0.8f, -0.8f, 0.0f);

GeometryInfo gi =
        new GeometryInfo
                (GeometryInfo.POLYGON_ARRAY);
gi.setCoordinates (pts);
gi.setStripCounts (stripCounts);
gi.setContourCounts (contourCounts);

Triangulator tr = new Triangulator ();
tr.triangulate (gi);
GeometryArray triWithHole = gi.getGeometryArray ();
```

NormalGenerator

- Normals are essential for shading. The ***NormalGenerator*** analyses the surface of connected polygon to determine if they meet smoothly or in a crease.



- It does this by calculating if the joint angle is greater than the crease angle that was defined for the surface.



NormalGenerator Example

```
GeometryInfo gi =
        new GeometryInfo (GeometryInfo.QUAD_ARRAY);

Point3f [ ] pts = new Point3f [8];
pts[0] = new Point3f (-1.0f, -1.0f, -1.0f);
pts[1] = new Point3f (1.0f, -1.0f, -1.0f);
pts[2] = new Point3f (-1.0f, 1.0f, -1.0f);
pts[3] = new Point3f (-1.0f, -1.0f, 1.0f);
pts[4] = new Point3f (-1.0f, 1.0f, 1.0f);
pts[5] = new Point3f (1.0f, -1.0f, 1.0f);
pts[6] = new Point3f (1.0f, 1.0f, -1.0f);
pts[7] = new Point3f (1.0f, 1.0f, 1.0f);

int [ ] indices = { 0, 3, 4, 2,         // left
                    0, 1, 5, 3,         // bottom
                    0, 2, 6, 1,         // back
                    7, 5, 1, 6,         // right
                    7, 6, 2, 4,         // top
                    7, 4, 3, 5          // front
                  };

gi.setCoordinates (pts);
gi.setCoordinateIndices (indices);
NormalGenerator ng = new NormalGenerator ();
ng.setCreaseAngle ((float) Math.toRadians (45));
ng.generateNormals (gi);
GeometryArray cube = gi.getGeometryArray ();
```

High level Shape Primitives

- Java3D provide several basic shape classes, ***Primitives***. A ***Primitive*** is subclass of ***Group*** and may have several ***Shape3D*** nodes. For example, a ***Cone*** object contains 2 ***Shape*** nodes, one for the base and one for the body.
- The most general form of these primitives is:

  Cone (float radius, float height, int flags, int xdiv, int ydiv, Appearance app);
- The values for the *flags* field are:
  - ENABLE_APPEARANCE_MODIFY
  - ENABLE_GEOMETRY_PICKING
  - GENERATE_NORMALS
  - GENERATE_NORMALS_INWARDS
  - GENERATE_TEXTURE_COORDS
  - GEOMETRY_NO_SHARE
- The *xdiv* and *ydiv* control the resolution of the curved surfaces. The greater the resolution the smoother the curved surface but the rendering time is longer.

Loaders

- A large scene may contain thousands of basic 3D shapes. This is not something you would like to code “by hand”. 3D modeler tools allow you to build large scenes relatively easily and store the scene in some format.
- A ***Loader*** is a class that imports data from a file into Java3D scene graph. Java3D comes with loaders for Wavefront (.obj files) and Lightwave (.lwo files). A list of third party loaders can be found at:

  http://java3d.j3d.org/utilities/loaders.html
- ***Loader*** is an interface. The object that implements the interface for .obj files is ***ObjectFile***.

  ObjectFile ();

  ObjectFile (int flags);

  ObjectFile (int flags, float creaseAngle);

- Valid flag values that specify how the data is to be created are:
  - RESIZE
  - REVERSE
  - STRIPIFY
  - TRIANGULATE

Loader Example

```
java.net.URL imageURL =
    “http://www.dcu.ie/image.obj;

int flags = ObjectFile.RESIZE;
ObjectFile f = new ObjectFile (flags);
Scene s = null;

try
{
    s = f.load (imageURL)
}
catch (Exception e)
{
    System.err.println (“Can’t load file, exception: ” +
    e);
}

BranchGroup sceneGroup = s.getSceneGroup ();
```