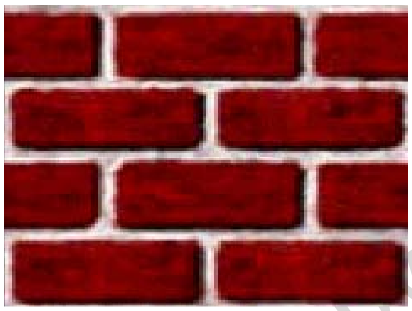
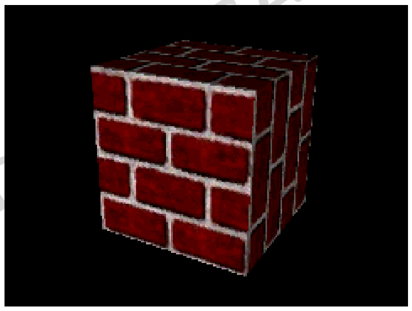
INTRODUCTION TO TEXTURE MAPPING

1. A `texture` is an image pasted onto a shape to create the illusion of detail.

   - Realism can be improved by modeling every detail of every 3D shape in a scene
     - This requires an enormous amount of modeling effort
     - More shapes ⇒ more to draw and worse interactivity
   - Realism can be improved with the illusion of detail:
     - Take a photograph of the "real thing"
     - Paste that photo onto simple 3D geometry
   - Texture mapping increases realism without increasing the amount of geometry to draw.

2. Bricks: an example

3. Texture mapping is controlled by node components in a shape's `Appearance`.

   - `Appearance` is a container for multiple visual attributes for a shape
     - Color and transparency control

- `Material`
- `ColoringAttributes`
- `TransparencyAttributes`

◦ Rendering control

- `PointAttributes`
- `LineAttributes`
- `PolygonAttributes`
- `RenderingAttributes`

◦ Texture control

- Texture control attributes are divided among a few node components
  - `Texture` selects a texture image and control basic mapping attributes

  - `TextureAttributes` controls advanced mapping attributes
  - `TexCoordGeneration` automatically generates texture co-ordinates if you do not provide your own.

4. `Texture` is the base class for two node components that select the image to use

- `Texture2D`: a standard 2D image
- `Texture3D`: a 3D volume of images

Class Hierarchy

```
java.lang.Object
└  javax.media.j3d.SceneGraphObject
   └  javax.media.j3d.NodeComponent
      └  javax.media.j3d.Texture
         ├  javax.media.j3d.Texture2D
         └  javax.media.j3d.Texture3D
```

5. Simple texture recipe

(1) Prepaer texture images

(2) Load the texture

(3) Set the texture in Appearance bundle

(4) Specfigy TexturCoordinates of Geometry

6. Image format

- JPEG/JPG (*Joint Photographic Experts Group*)
  - ◦ JPG is a lossy compression technique designed to compress color and grayscale continuous-tone images.
    - The information discarded in the compression is information that the human eye cannot detect.
    - JPG images support 16 million colors and are best

suited for photographs and complex graphics.

- The user typically has to compromise on either the quality of the image or the size of the file. JPG does not work well on line drawings, lettering or simple graphics because there is not a lot of the image that can be thrown out in the lossy process, so the image loses clarity and sharpness.

- GIF (*Graphics Interchange Format*)
  - ◦ Unlike JPG, the GIF format is a lossless compression technique and it supports only 256 colors.
    - GIF is better than JPG for images with only a few distinct colors, such as line drawings, B/W images and small text that is only a few pixels high.

- PNG (*Portable Network Graphics*)
  - PNG was developed as a patent-free answer to the GIF format but is also an improvement on the GIF technique.
  - An image in a lossless PNG file can be 5%-25% more compressed than a GIF file of the same image.

7. Preparing for texture mapping

- Ensure the images are of acceptable dimensions
  - For rendering efficiency, Java 3D requires the size of the texture image to be a mathematical power of two (1, 2, 4, 8, 16, ...) in each dimension.
  - Failing to meet this restriction will result in a runtime exception.

- Ensure the images are saved in a file format which can be read.

8. Loading a texture requires:

- A `TextureLoader` to load that file

```
TextureLoader myLoader = new TextureLoader("brick.jpg");
```

- An `ImageComponent` uses a standard `BufferedImage` to hold the loaded image
  - The extended `ImageComponent2D` holds a 2D image.

```
ImageComponent2D myImage = myLoader.getImage( );
```

9. The remaining steps

- Create a Texture2D using the image, and turn it on

```
Texture2D myTex = new Texture2D( );
```

```
myTex.setImage( 0, myImage );
myTex.setEnable( true );
```

- Create an Appearance and set the texture in it

```
Appearance myAppear = new Appearance( );
myAppear.setTexture( myTex );
```

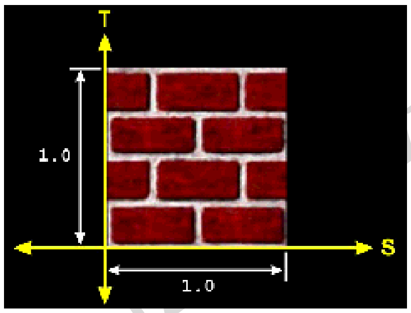
- Assemble the shape

```
Shape3D myShape = new Shape3D( myGeom, myAppear );
```

USING TEXTURE COORDINATES

1. *Texture coordinates* describe a 2D shape that maps from parts of a texture to parts of a shape.
   - Define a "texture cookie cutter" to cut out a texture piece
   - Translate, rotate, and scale the cookie cutter before cutting out the piece
   - Map the cut out texture "cookie" onto your shape
2. Texture images have a *true size* and a *logical size*
   - True size is the width and height of the image in pixels
     - Must be powers of 2
     - Width and height need not be the same

- Logical size is a generic treatment of image dimensions
  - *Always* a width of 1.0
  - *Always* a height of 1.0



3. An example of texture coordinates

- Create lists of 3D coordinates, lighting normals, and texture coordinates for the vertices

```
Point3f[] myCoords = {
    new Point3f( 0.0f, 0.0f, 0.0f ),
     .  .  .
}
Vector3f[] myNormals = {
     new Vector3f( 0.0f, 1.0f, 0.0f ),
     .  .  .
}
Point2f[] myTexCoords = {
    new Point2f( 0.0f, 0.0f ),
     .  .  .
}
```

- Create a QuadArray and set the vertex coordinates, lighting normals, and texture coordinates

```
QuadArray myQuads = new QuadArray(
    myCoords.length,
    GeometryArray.COORDINATES |
    GeometryArray.NORMALS |
    GeometryArray.TEXTURE_COORDINATE_2 );
myQuads.setCoordinates( 0, myCoords );
myQuads.setNormals( 0, myNormals );
myQuads.setTextureCoordinates( 0, myTexCoords );
```

- Assemble the shape

```
Shape3D myShape = new Shape3D( myQuads, myAppear );
```

4. The "texture cookie cutter" can be transformed before cutting out a piece of texture.

   - `TextureAttributes` control how a texture is mapped, including use of a texture coordinates transform.

Class Hierarchy

java.lang.Object
└ **javax.media.j3d.SceneGraphObject**
  └ **javax.media.j3d.NodeComponent**
    └ **javax.media.j3d.TextureAttributes**

5. An example of texture rotation

   - Create TextureAttributes

```
TextureAttributes myTA = new TextureAttributes( );
```

   - Create a rotation transform

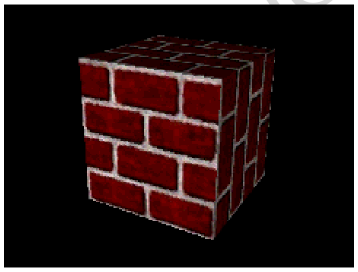
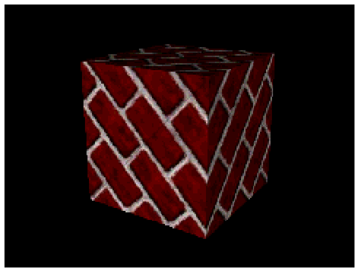
```
Transform3D myTrans = new Transform3D( );
myTrans.rotZ( Math.PI/4.0 ); // 45 degrees
myTA.setTextureTransform( myTrans );
```

- Set the texture attributes on an Appearance

```
Appearance myAppear = new Appearance( );
myAppear.setTextureAttributes( myTA );
```

- Assemble the shape

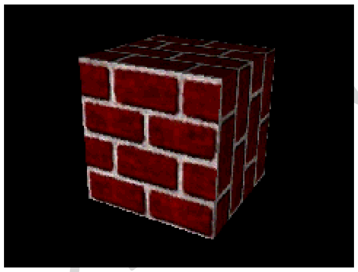
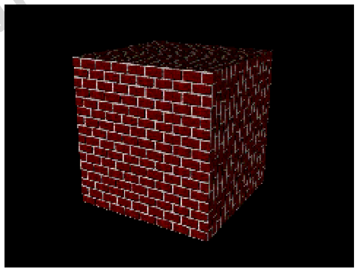
```
Shape3D myShape = new Shape3D( myText, myAppear );
```

6. An example of texture scaling

   - Create a scaling transform

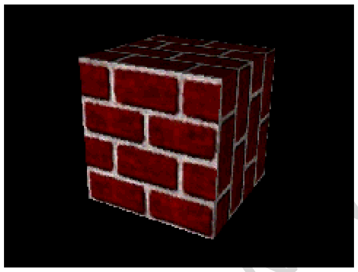
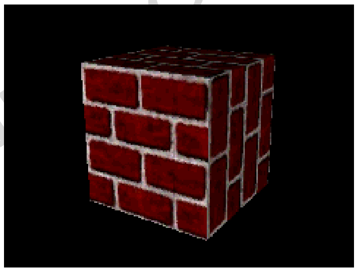
```
Transform3D myTrans = new Transform3D( );
myTrans.set( 4.0 );
myTA.setTextureTransform( myTrans );
```

7. An example of texture translation

   - Create a translation transform

```
Transform3D myTrans = new Transform3D( );
myTrans.set( new Vector3f( 0.25f, 0.0f, 0.0f ) );
myTA.setTextureTransform( myTrans );
```

8. Using texture boundary modes when texture coordinates extend past the edge of the image they can.
   - *Wrap* to create a repeating pattern
   - *Clamp* to prevent repeatition

9. An example of texture boundary mode
   - Load a texture image

```
TextureLoader myLoader = new TextureLoader("brick.jpg");
ImageComponent2D myImage = myLoader.getImage( );
```

   - Create a Texture2D using the image, and turn it on

```
Texture2D myTex = new Texture2D( );
myTex.setImage( 0, myImage );
myTex.setEnable( true );
```

   - Set the boundary modes and color

```
myTex.setBoundaryModeS( Texture.WRAP );
myTex.setBoundaryModeT( Texture.WRAP );
// WRAP is the default in both S and T
```

   - Create an Appearance and set the texture in it

```
Appearance myAppear = new Appearance( );
myAppear.setTexture( myTex );
```

- Assemble the shape

```
Shape3D myShape = new Shape3D( myText, myAppear );
```