Computer Game Technology

Textures

Introduction

- Use of 2D textures
- Texture mapping
- MIPmaps
- Texture mapping in Java 3D

Using 2D Textures

- Surface features of real-world objects are often complex
  - Bricks, wood, grass, trees, clothing, carpet
- Fine detail could be represented by explicit geometry
  - E.g. all branches, leaves, bark on a tree
- But this can easily become too computationally expensive
- Good visual appeal may still be achieved by using a 2D image as a surface texture on a simple geometry
  - E.g. planet Earth in first 3D practical

Example Textures

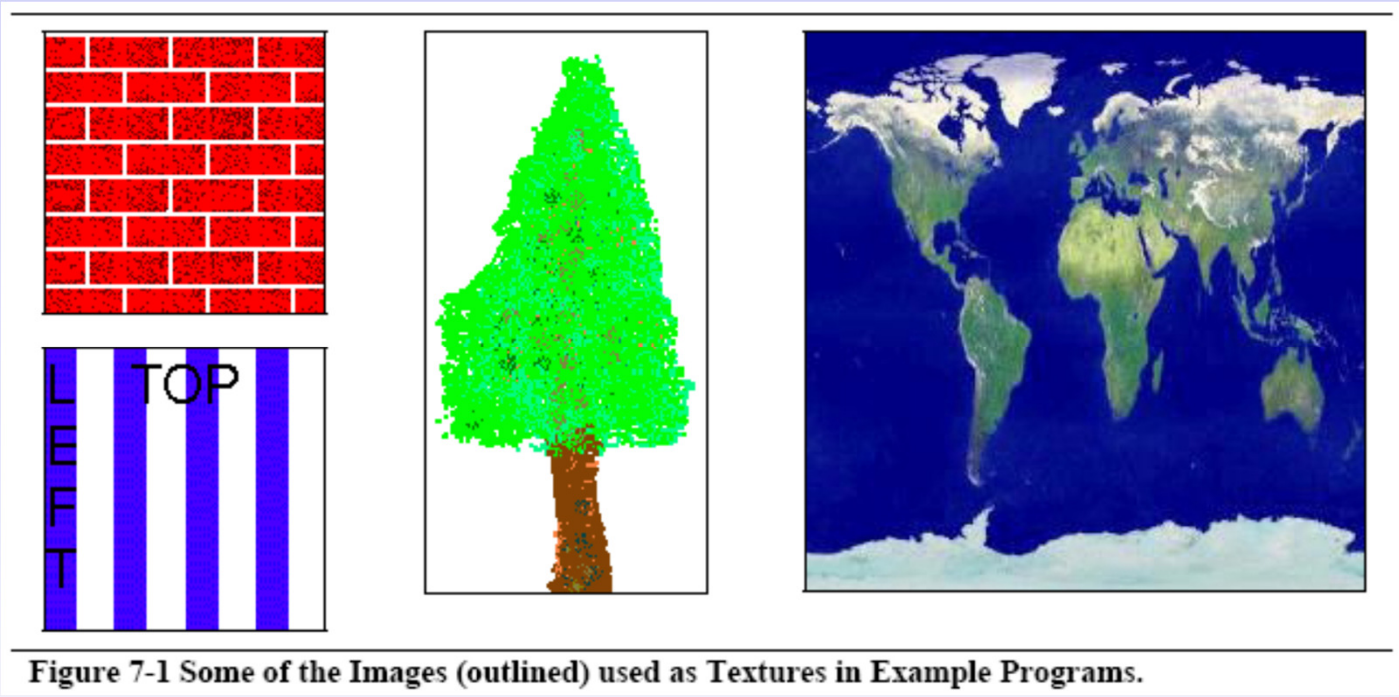
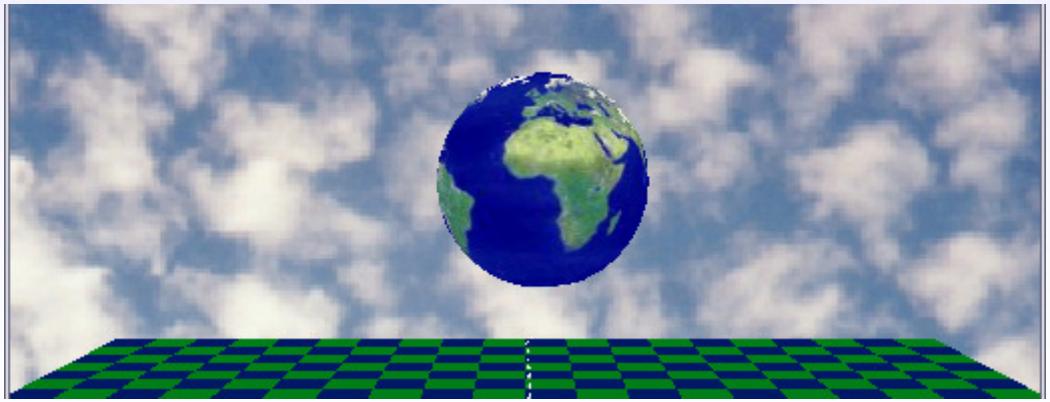


Figure 7-1 Some of the Images (outlined) used as Textures in Example Programs.



Texture Mapping

- Instead of displaying a plain colour on a polygon, we can fit a texture onto it instead.
- This can make it look a lot more realistic.
- This technique is called texturing or 'Texture Mapping'.
  - You may have used textures to fill an area in a 2D drawing.
  - Backgrounds on web pages

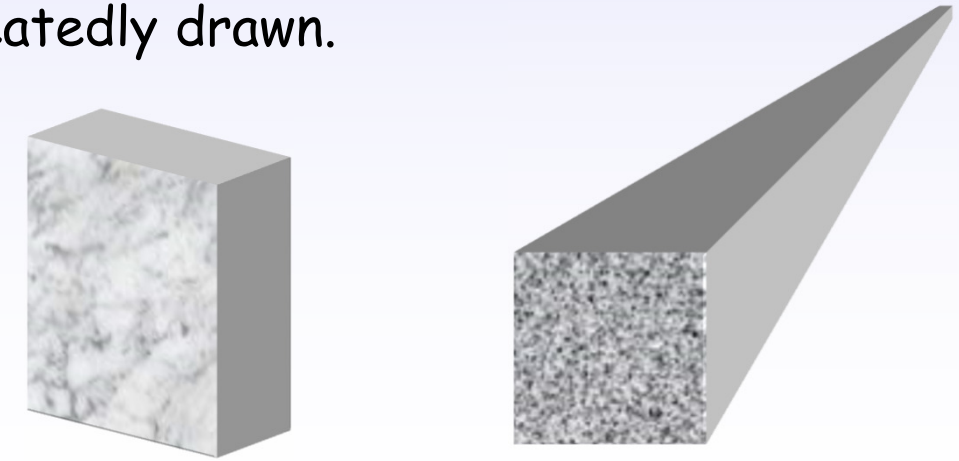
Example:

Texture Mapping (2)

- The basic problem is how to map the 2D image to the surface of the polygon
  - Images and polygons may be different sizes and shapes
- The textures used to fill a polygon are often smaller than the area they are filling and need to be repeated
  - Choosing a suitable isomorphic pattern can make this less obvious.
  - It is the same principle as background patterns on PC screens or web pages.

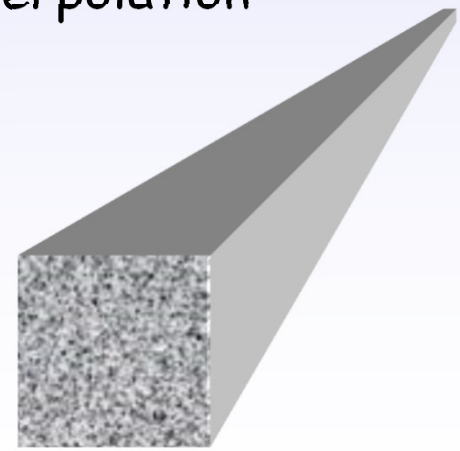
Texture Mapping (3)

- Each pixel in a texture is called a texel
  - texture element
- We might consider mapping each texel in our texture to a pixel on the screen
- This can inefficient however
  - If a polygon such as a rectangle is seen obliquely then some distortion occurs.
  - Texels furthest from the camera overlap and would be repeatedly drawn.



Texture Mapping (4)

- It is more common to map pixels to texels
  - For each pixel on the screen, find out which texel it should be.
- You can probably work out that this will be enough since many texels could end up mapping to just one pixel
  - In this case we need to use averaging techniques such as bi-linear or tri-linear interpolation



MIP Maps

- Extra efficiency can be gained by using a number of textures for a given geometry
- Texture images of smaller size are applied as the visual object moves further away from the viewer
  - Known as MIP mapping (*multum in parvo* – many things in a small place)
  - Same texture at different resolutions
  - Aim to use texture image closest in size to the screen size of the object to simplify pixel-to-texel mapping
- Similar to LOD, in which the geometry is changed (simplified) with distance

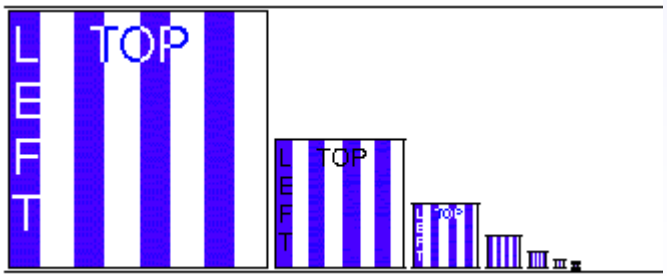


Figure 7-19 Multiple Levels of a Texture (outlined).

Textures in Java 3D

The basic recipe for using textures in Java 3D is:

1. Prepare image files: each dimension must be a power of 2 for efficiency in texture mapping e.g. 256 x 128
2. Load image into a Texture object
3. Add texture to an Appearance bundle
4. Specify texture coordinates for object geometry

- Hardest work is specifying texture coordinates, which determine the basic mapping of the texture to the surface of the visual object
  - Primitive objects, such as Sphere, can do this automatically (see code in first 3D practical)

Creating the Appearance Bundle

Code fragment for loading an image and adding the texture to an appearance bundle:

```
TextureLoader loader = new TextureLoader("stripe.jpg", this);
Appearance appear = new Appearance();
appear.setTexture(loader.getTexture());
```

More explicit code that could allow setting texture properties:

```
TextureLoader loader = new TextureLoader("stripe.jpg", this);
ImageComponent2D image = loader.getImage();
Texture2D texture = new Texture2D();
texture.setImage(0, image);
Appearance appear = new Appearance();
appear.setTexture(texture);
```

Setting Texture Coordinates

- Need to relate the texture image to the surface of the object to be textured
- Set up mapping between texture coordinates and the vertices of the object geometry
  - Texture coordinates are (s,t)
  - Vertex coordinates are (x, y, z)
- e.g. map a square image onto a Quad array (square)

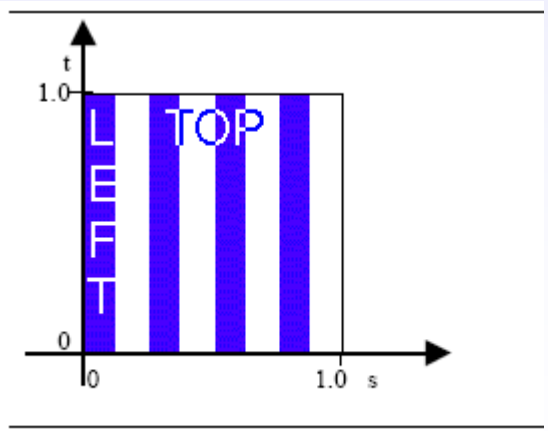
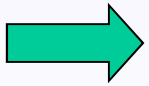


Figure 7-3 Texture Mapping Coordinates



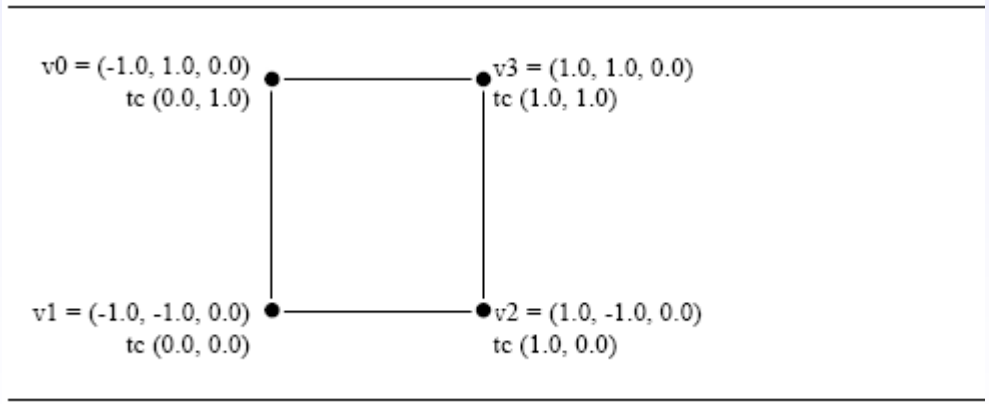
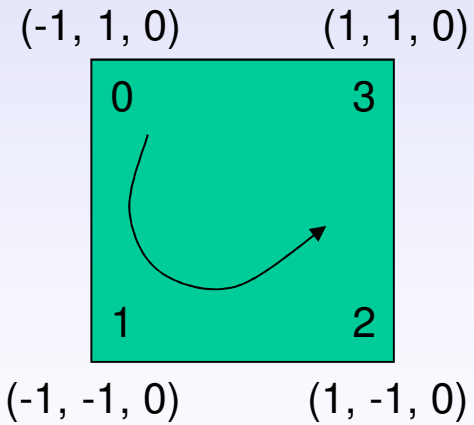


Figure 7-4 The Orientation of Texture Coordinates in Texture Image Space.

Setting Texture Coordinates (2)

Code fragment for setting texture coordinates for a Quad array - creating the Quad array geometry (square in x-y plane):

```
QuadArray plane = new QuadArray(4, GeometryArray.COORDINATES
    | GeometryArray.TEXTURE_COORDINATE_2);
Point3f p = new Point3f();
p.set(-1.0f, 1.0f, 0.0f);
plane.setCoordinate(0, p);
p.set(-1.0f, -1.0f, 0.0f);
plane.setCoordinate(1, p);
p.set( 1.0f, -1.0f, 0.0f);
plane.setCoordinate(2, p);
p.set( 1.0f, 1.0f, 0.0f);
plane.setCoordinate(3, p);
```



Setting Texture Coordinates (3)

Code fragment continued – setting the texture coordinates:

```
TexCoord2f q = new TexCoord2f();
q.set(0.0f, 1.0f);
plane.setTextureCoordinate(0, 0, q);
q.set(0.0f, 0.0f);
plane.setTextureCoordinate(0, 1, q);
q.set(1.0f, 0.0f);
plane.setTextureCoordinate(0, 2, q);
q.set(1.0f, 1.0f);
plane.setTextureCoordinate(0, 3, q);
```

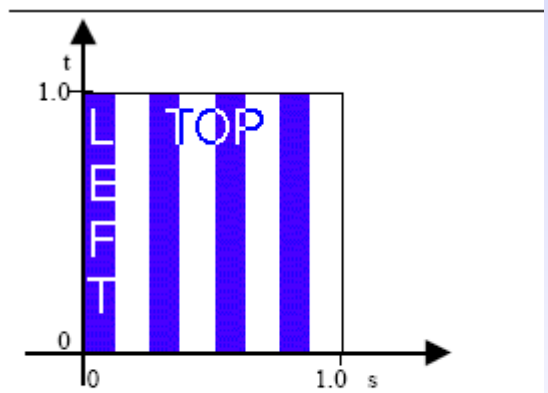


Figure 7-3 Texture Mapping Coordinates

Different Mappings

How we map the texture coordinates to the vertices will determine the orientation of the texture on the plane:

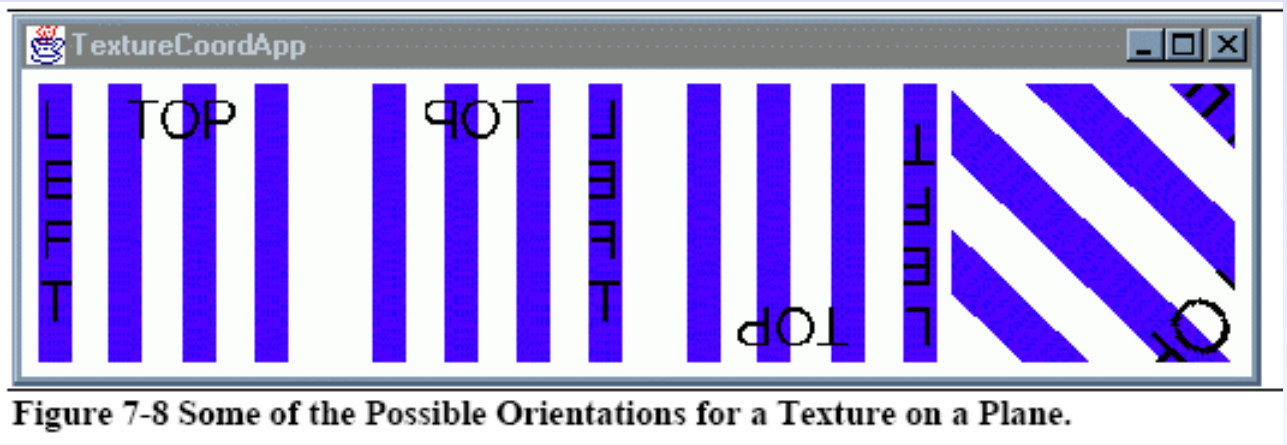


Figure 7-8 Some of the Possible Orientations for a Texture on a Plane.

Mapping Non-vertex Coordinates

- Rendering the textured visual object involves choosing a texel for each pixel corresponding to the object
- Texels for vertices are defined by the texture coordinates
- Texels for non-vertex pixels determined by trilinear interpolation

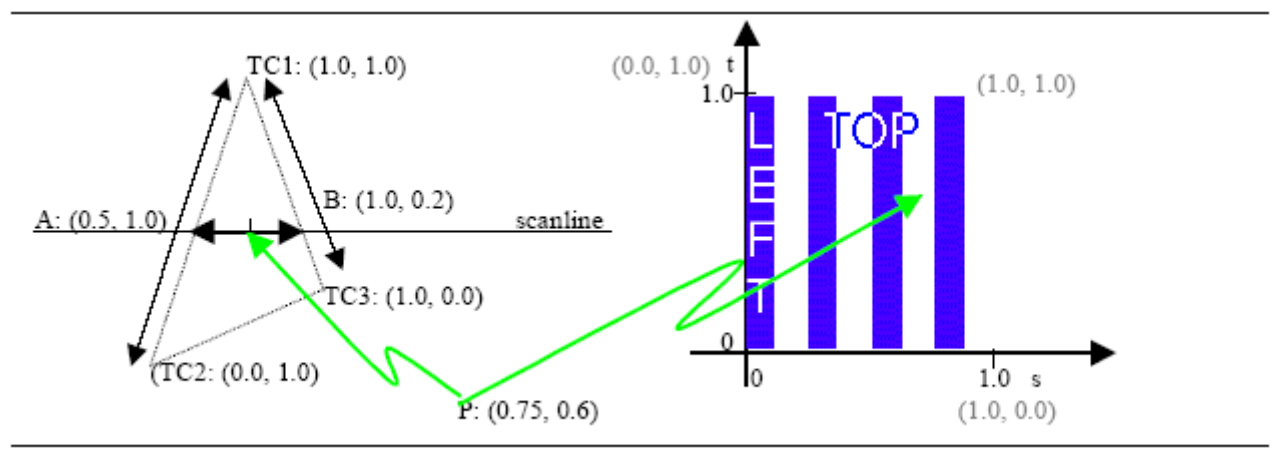


Figure 7-7 Picture of Texture Mapping

Texture Rendering Choices

- Texture mapping may reference texels outside the boundary of the texture image
  - Image may be tiled (repeated) to provide a texel
  - Colour at the edge of the image may be used
- Final pixel colour depends not only on texel colour but also on geometry colours and lighting
- Choices for how these colours are combined
  - Various blending modes can be set through a TextureAttributes object

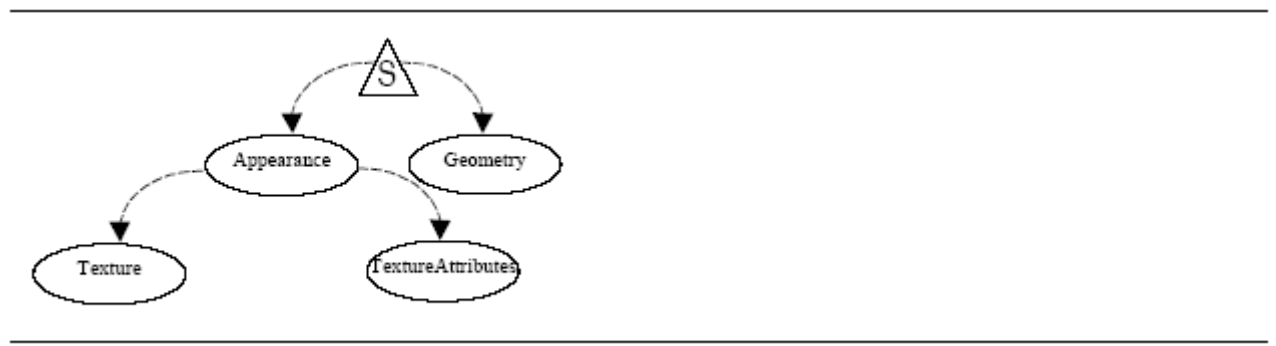


Figure 7-10 An Appearance Bundle with Texture and TextureAttributes Components.

The End