Xi'an Jiaotong-Liverpool University
西交利物浦大學

CPT205 Computer Graphics

Texture Mapping

Lecture 10
2025-26

Yong Yue and Nan Xiang

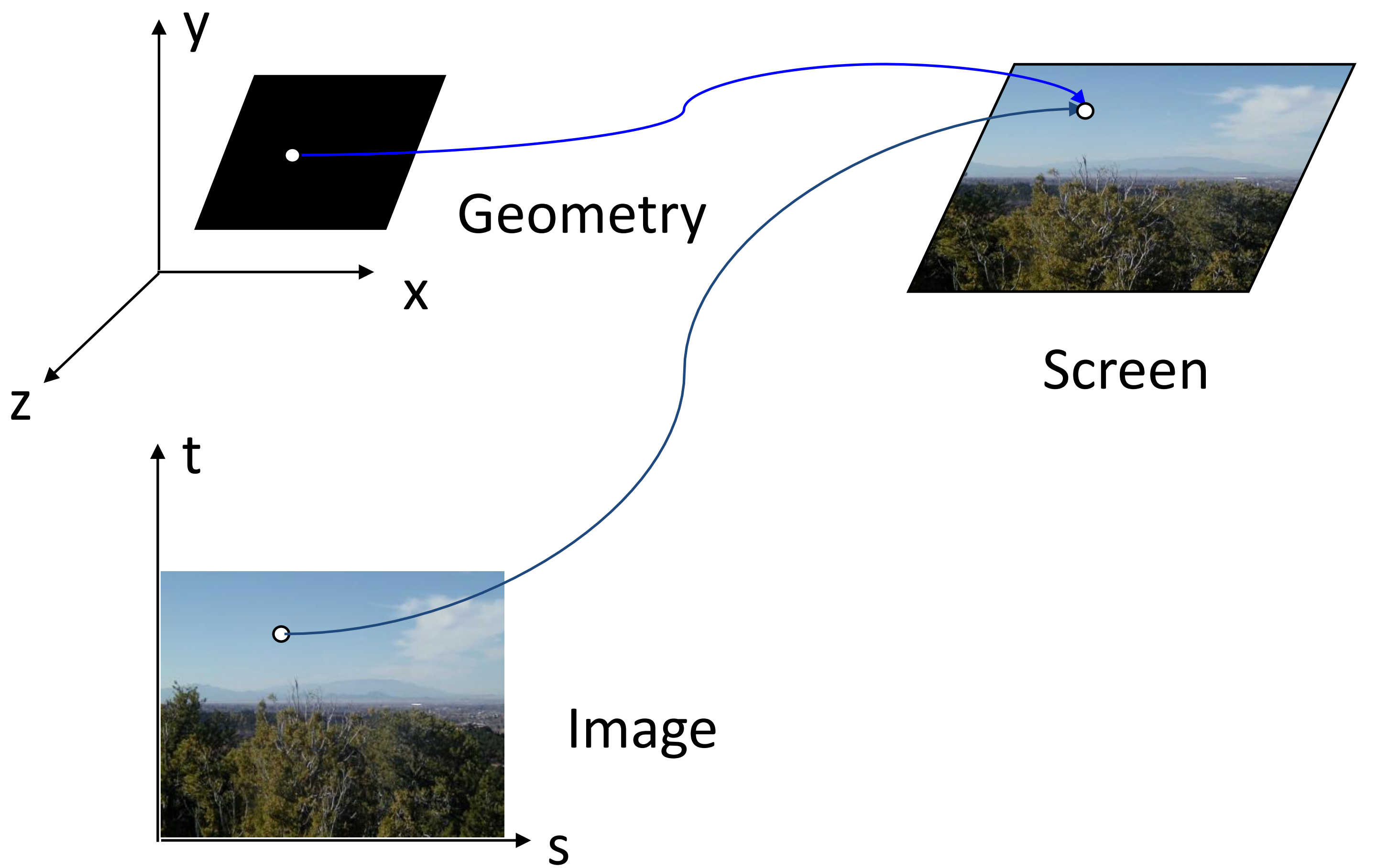
Topics for today

- Concepts
- Types of texture mapping
- Specifying the texture
- Magnification and minification
- Two-step mapping
- Steps in texture mapping and OpenGL functions
- Filtering, wrapping and modes
- Multiple levels of detail – Mipmapping

Concepts of texture mapping (1)

- Although graphics cards can render over 10 million polygons per second, this may be insufficient or there could be alternative way to process phenomena, e.g. clouds, grass, terrain and skin.
- A common method for adding detail to an object is to map patterns onto the geometric description of the object.
  The method for incorporating object detail into a scene is called texture mapping or pattern mapping.
- What makes texture mapping tricky is that a rectangular texture can be mapped to nonrectangular regions, and this must be done in a reasonable way.
- When a texture is mapped, its display on the screen might be distorted due to various transformations applied – rotations, translations, scaling and projections.

Concepts of texture mapping (2)



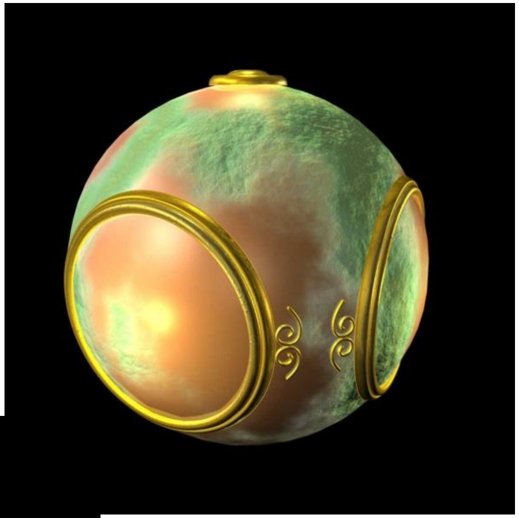
Types of texture mapping (1)

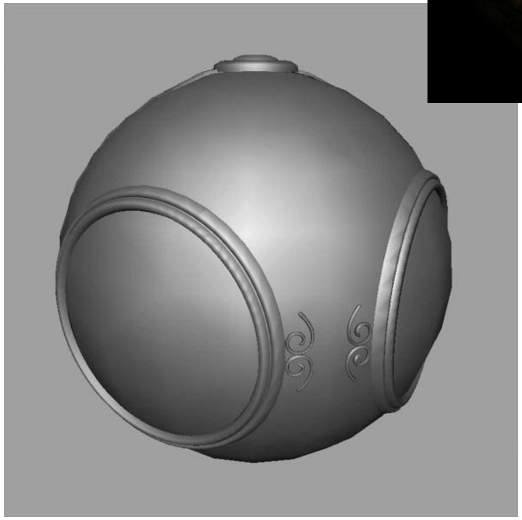
- Texture Mapping
  - Uses images to fill inside of polygons
- Environment (reflection) mapping
  - Uses a picture of the environment for texture maps
  - Allows simulation of highly specular surfaces
- Bump mapping
  - Emulates altering normal vectors during the rendering process

Types of texture mapping (2)

Texture mapped

Bump mapped





Geometric model

Environment mapped

Specifying the texture (1)

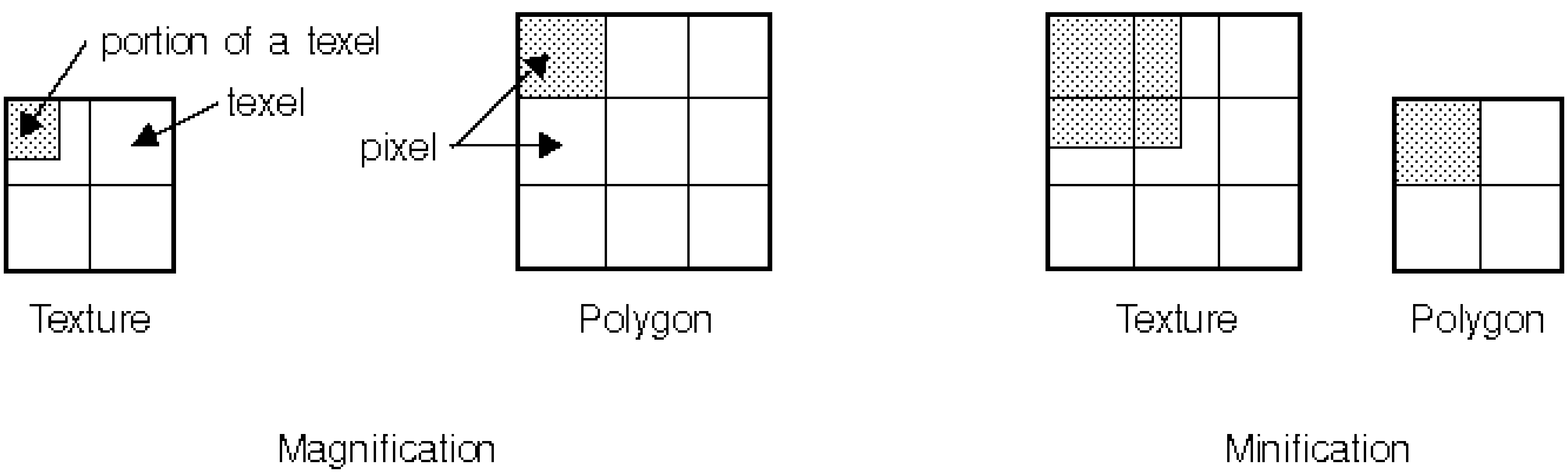
- There are two common types of texture:
  - Image: a 2D image is used as the texture.
  - Procedural: a procedure or program generates the texture.
- The texture can be defined as one-dimensional, two-dimensional or three-dimensional patterns.

Specifying the texture (2)

- Texture functions in a graphics package often allow the number of colour components for each position in a pattern to be specified as an option.
- For example, each colour definition in a texture pattern could consist of four RGBA components, three RGB components, a single intensity value for a shade of blue, an index into a colour table, or a single luminance value (a weighted average of the RGB components of a colour).
- The individual values in a texture array are often called *texels* (texture elements).

Magnification and minification (1)

- Depending on the texture size, the distortion and the size of the screen image, a texel may be mapped to more than one pixel (called *magnification*), and a pixel may be covered by multiple texels (called *minification*).

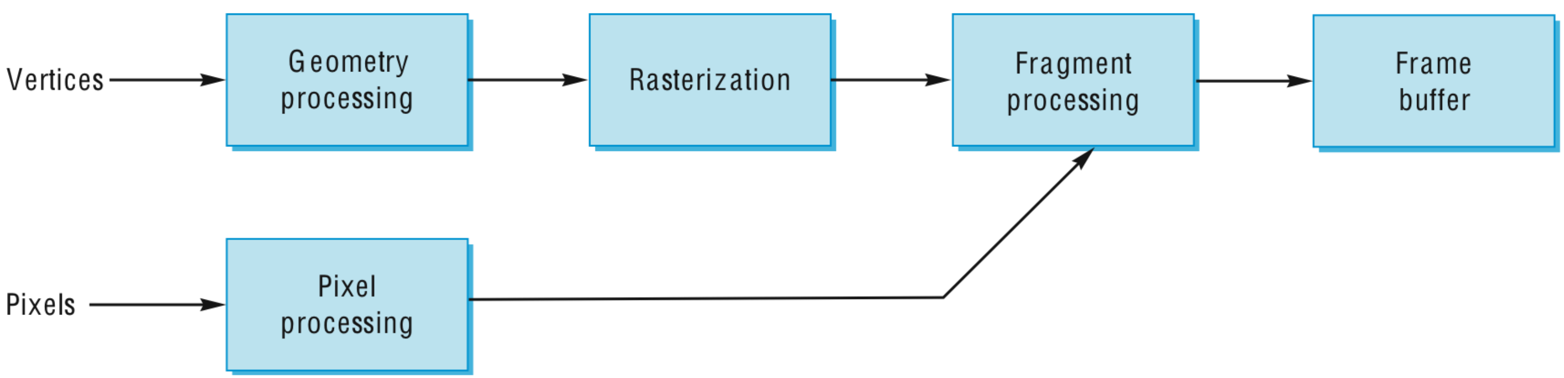


Magnification and minification (2)

- Since the texture is made up of discrete texels, filtering operations must be performed to map texels to pixels.
- For example, if many texels correspond to a pixel, they are averaged down to fit; if texel boundaries fall across pixel boundaries, a weighted average of the applicable texels is performed.
- Because of these calculations, texture mapping can be computationally expensive, which is why many specialised graphics systems include hardware support for texture mapping.

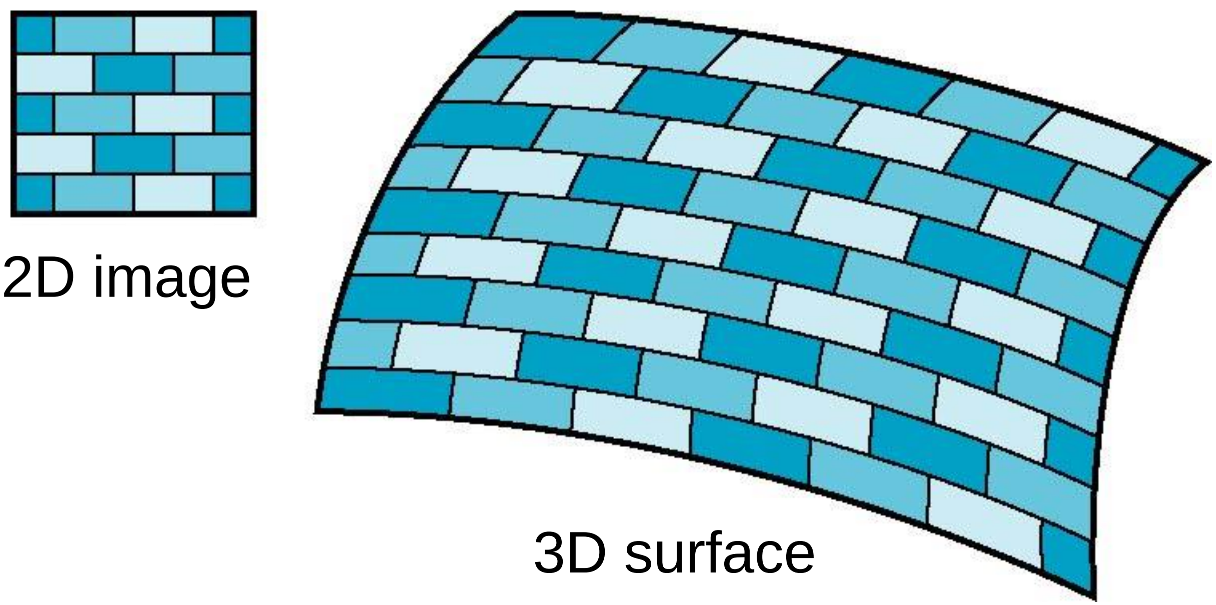
Where does it take place?

- Texture mapping techniques are implemented at the end of the rendering pipeline.
- It is very efficient because a few polygons make it past the clipper.



Is it simple?

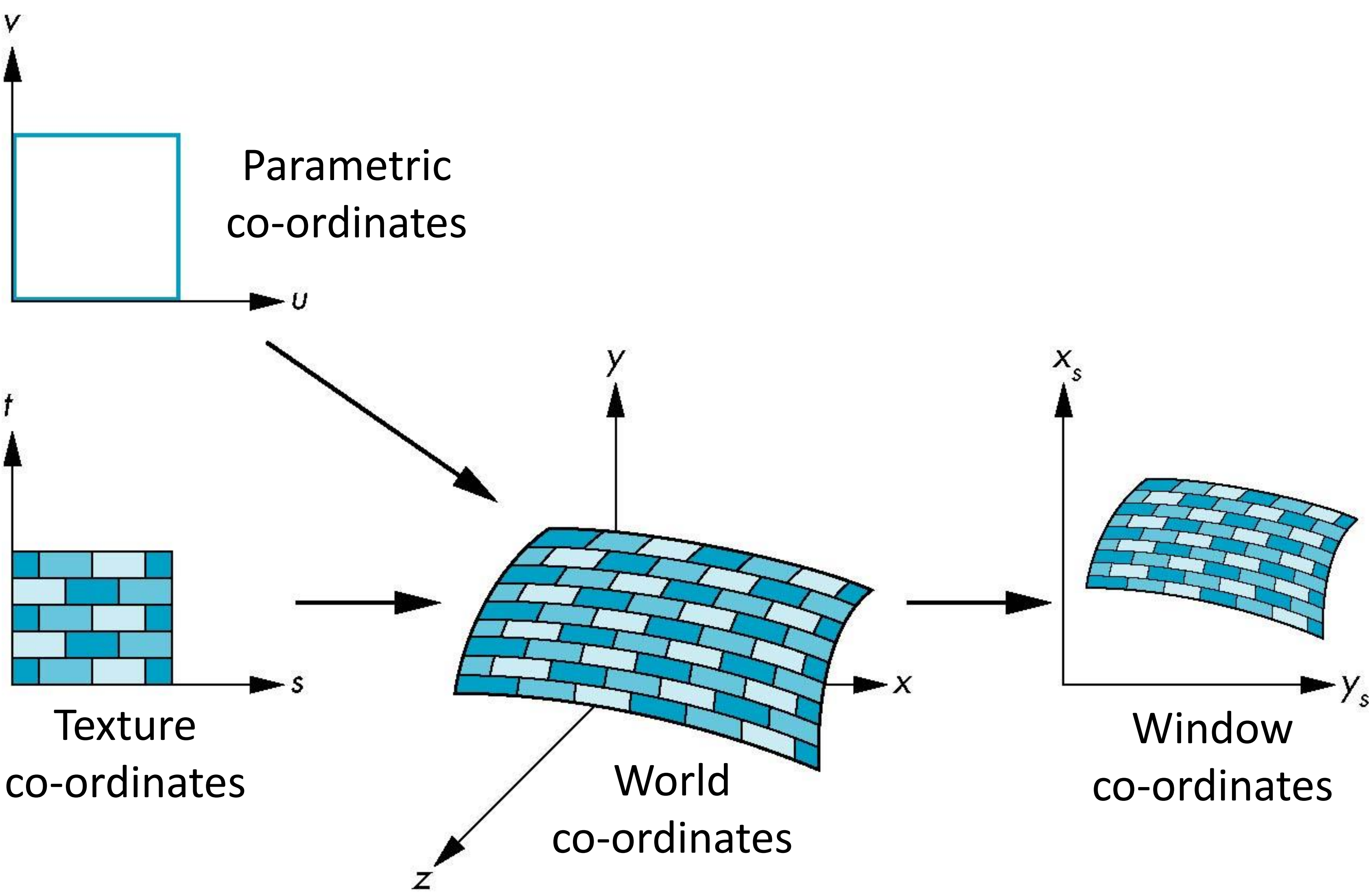
Although the idea is simple, mapping an image to a surface, 3 or 4 coordinate systems are involved.



Co-ordinate systems for texture mapping

- Parametric co-ordinates
  may be used to model curves and surfaces
- Texture co-ordinates
  is used to identify points in the image to be mapped
- Object or world co-ordinates
  conceptually, where the mapping takes place
- Window co-ordinates
  where the final image is finally produced

Co-ordinate systems for texture mapping

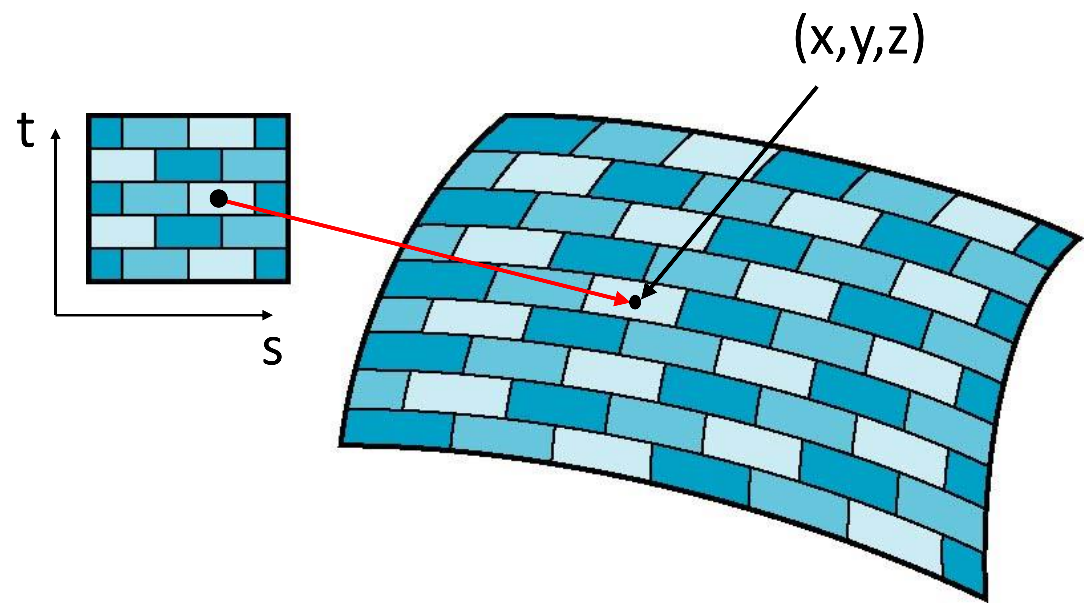


Co-ordinate systems for texture mapping

- The basic problem is how to find the maps.
- Consider mapping from texture co-ordinates to a point on a surface.
- Apparently three functions are needed

  $x = x(s,t)$
  $y = y(s,t)$
  $z = z(s,t)$

- But we really want to go the other way.



Backward mapping

- We really want to go backwards (screen order):
  - Given a pixel, we want to know to which point on an object it corresponds.
  - Given a point on an object, we want to know to which point in the texture it corresponds.

- Need a map of the form:

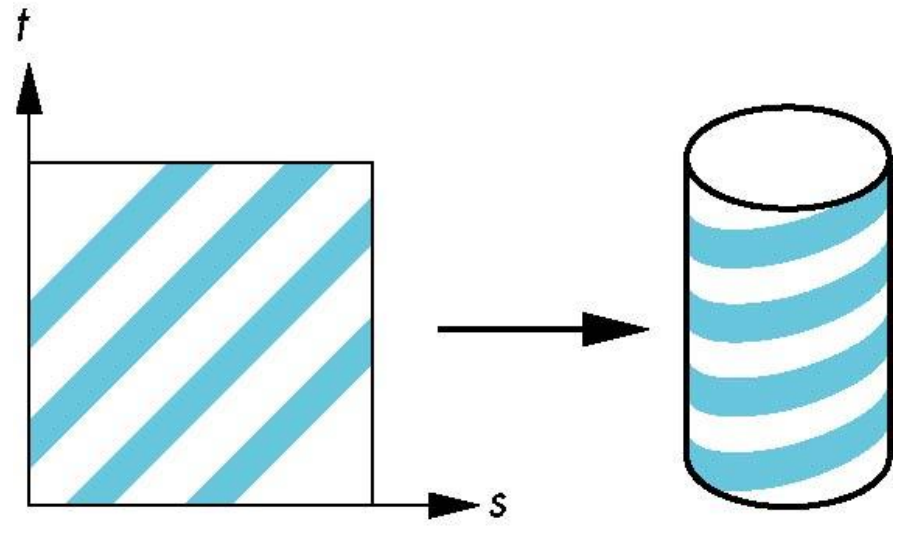
  $s = s(x,y,z)$

  $t = t(x,y,z)$

- Such functions are difficult to find in general.

Two-part mapping

- One solution to the mapping problem is to first map the texture to a simple <u>intermediate</u> surface.
- Example: mapping to cylinder.



Cylindrical mapping

A parametric cylinder

x = r*cos(2pi*u)

y = r*sin(2pi*u)

z = h*v

maps a rectangle in the (u,v) space to the cylinder of radius r and height h in the world co-ordinates

s = u

t = v

maps from the texture space.

Spherical mapping

We can map a parametric sphere

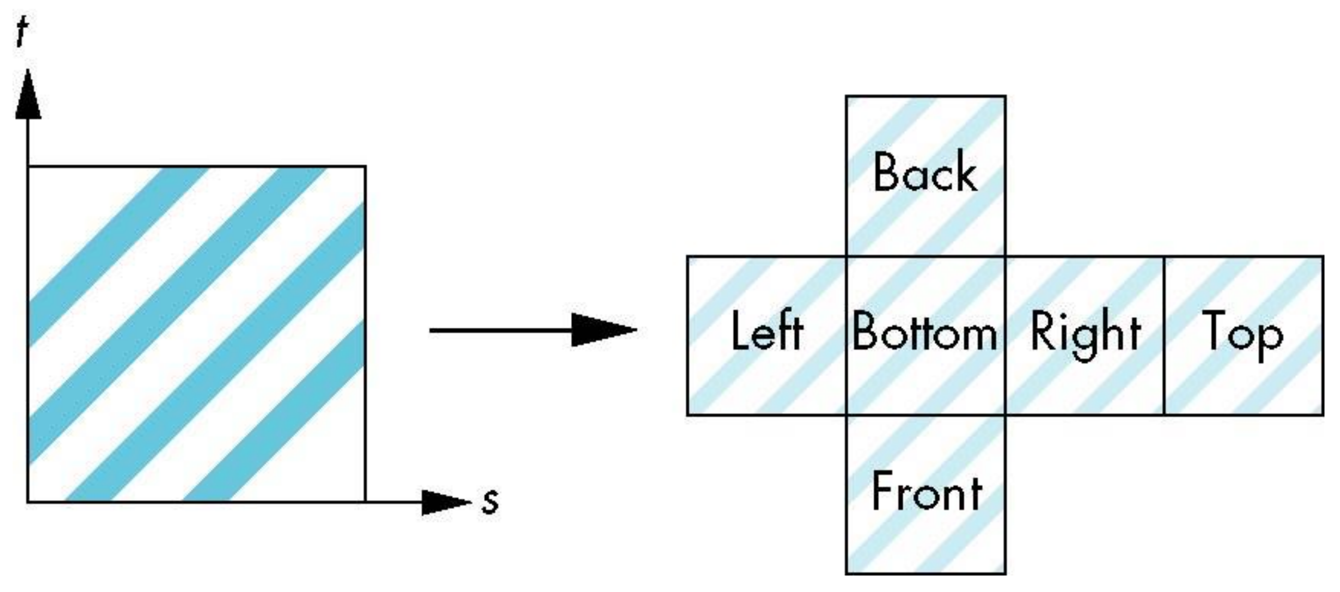
x = r*cos(2pi*u)
y = r*sin(2pi*u)*cos(2pi*v)
z = r*sin(2pi*u)*sin(2pi*v)

in a similar manner to the cylinder but have to decide where to put the distortion.

Spheres are used in environmental maps.

Box mapping

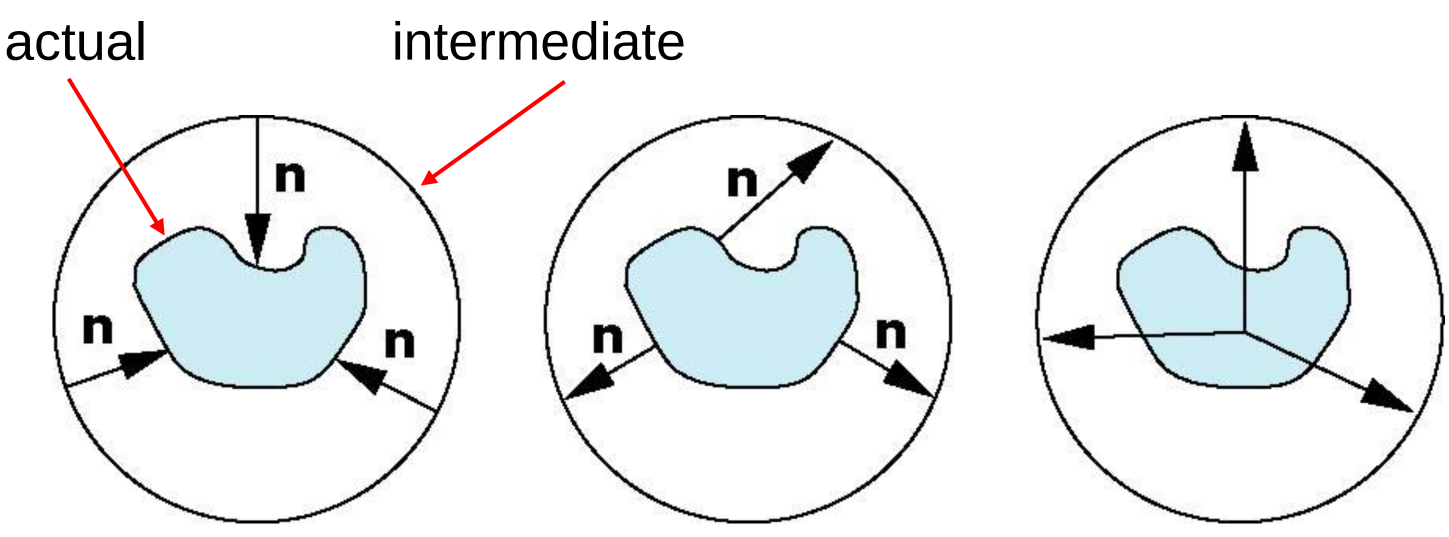
- Easy to use with simple orthographic projection
- Also used in environmental maps



Second mapping

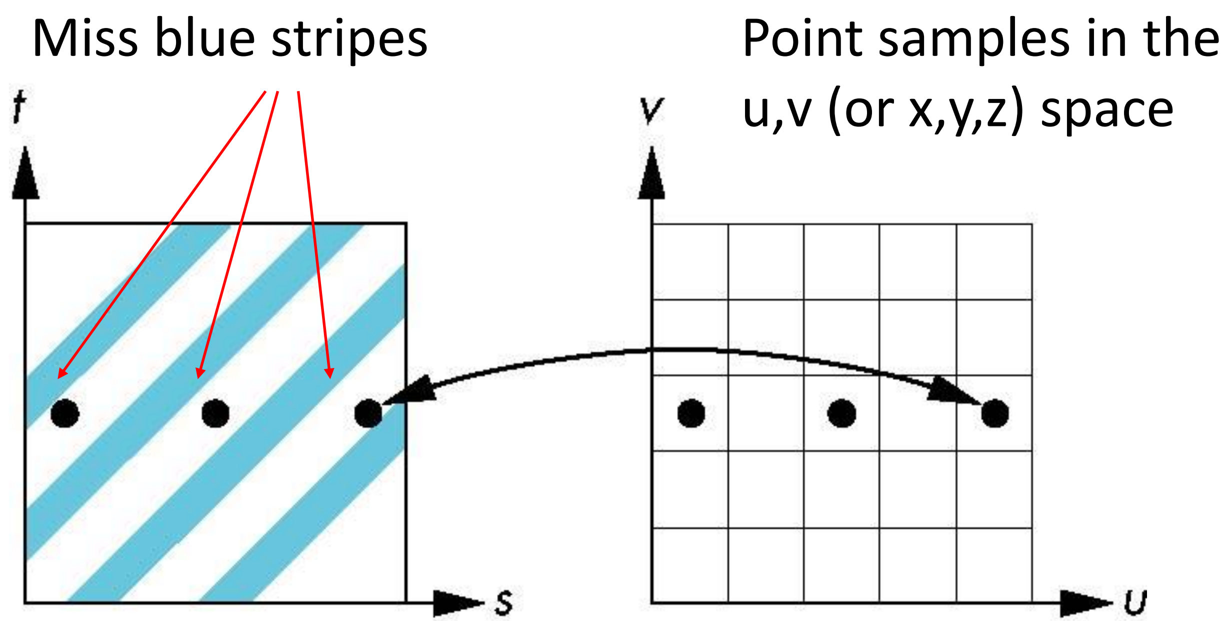
Mapping from an intermediate object to an actual object

- Normals from the intermediate to actual objects
- Normals from the actual to intermediate objects
- Vectors from the centre of the intermediate object



Aliasing

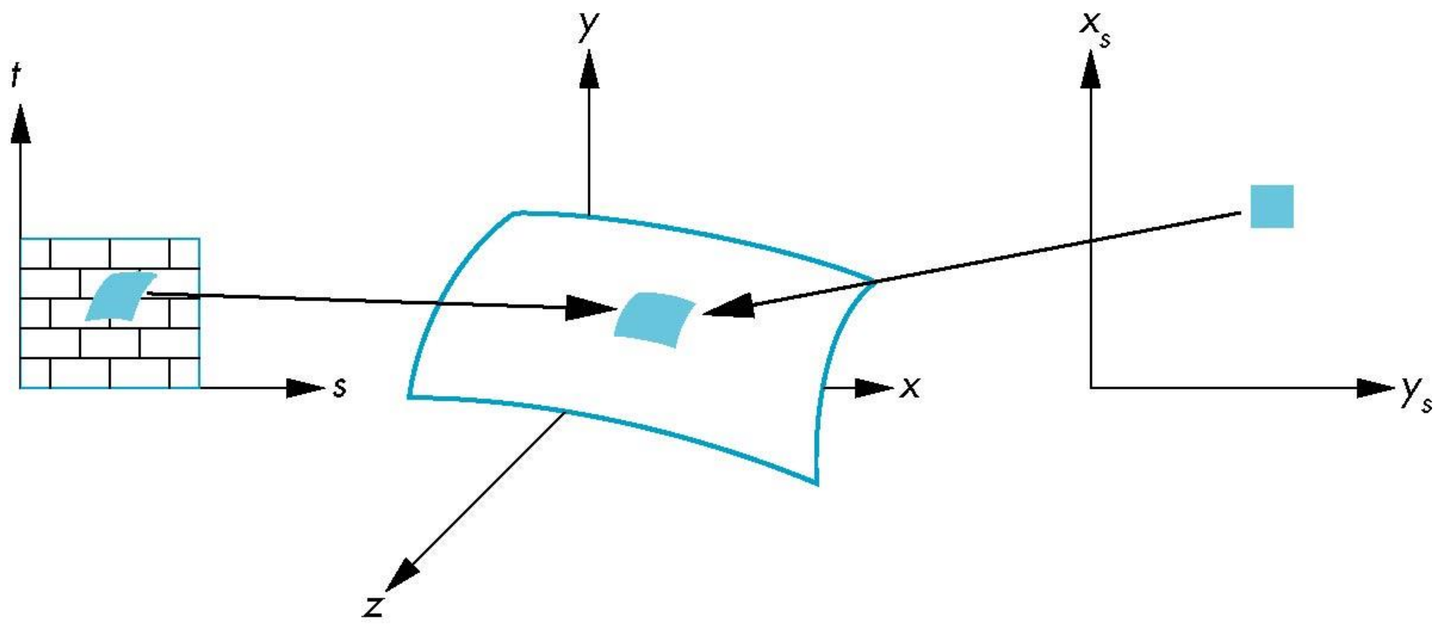
- Point sampling of the texture can lead to aliasing errors



Point samples in the texture space

Area averaging

A better but slower option is to use *area averaging.*



Note that the *pre-image* of the pixel is curved.

OpenGL steps in texture mapping

Three main steps to applying a texture

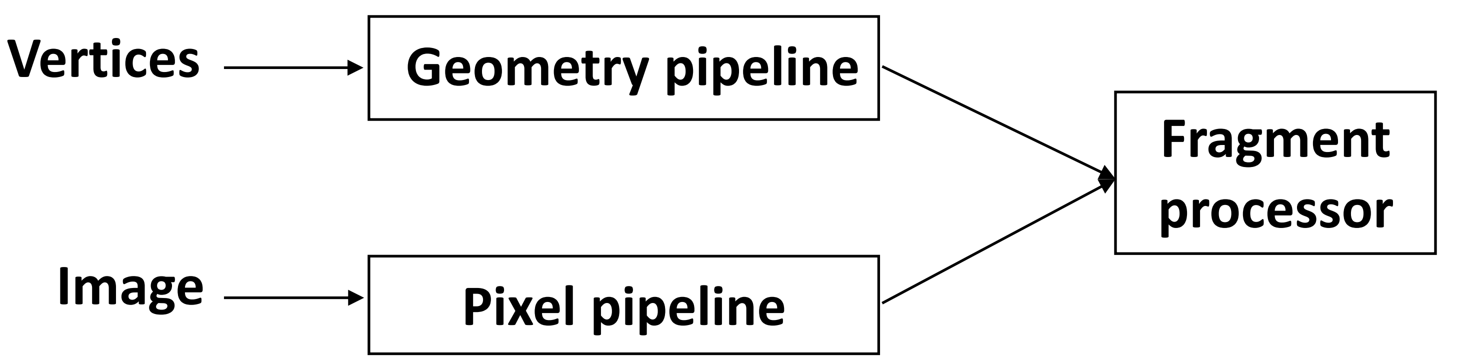
1. Specify the texture
   - Read or generate image
   - Assign to texture
   - Enable texturing
2. Assign texture coordinates to vertices
   - Proper mapping function is left to application
3. Specify texture parameters
   - Mode
   - Filtering
   - Wrapping

OpenGL functions

- **`glTexImage2D()`**: Defining image as texture (type, level and colour)
- **`glTexParameter*()`**: Specifying filtering and wrapping
- **`glTexEnv{fi}[v]()`**: Specifying mode (modulating, blending or decal)
- **`glTexCoord{1234}{sifd}{v}()`**: Specifying texture co-ordinates s and t while r is set to 0 and q to 1
- **`glTexGen{ifd}[v]()`**: Automatic generation of texture co-ordinates by OpenGL
- **`glHint()`**: Defining perspective correction hint
- **`glBindTexture()`**: Binding a named texture to a texturing target

Texture mapping and the OpenGL pipeline

- The images and geometry flow through separate pipelines that join during fragment processing.
- “Complex” textures do not affect geometric complexity.

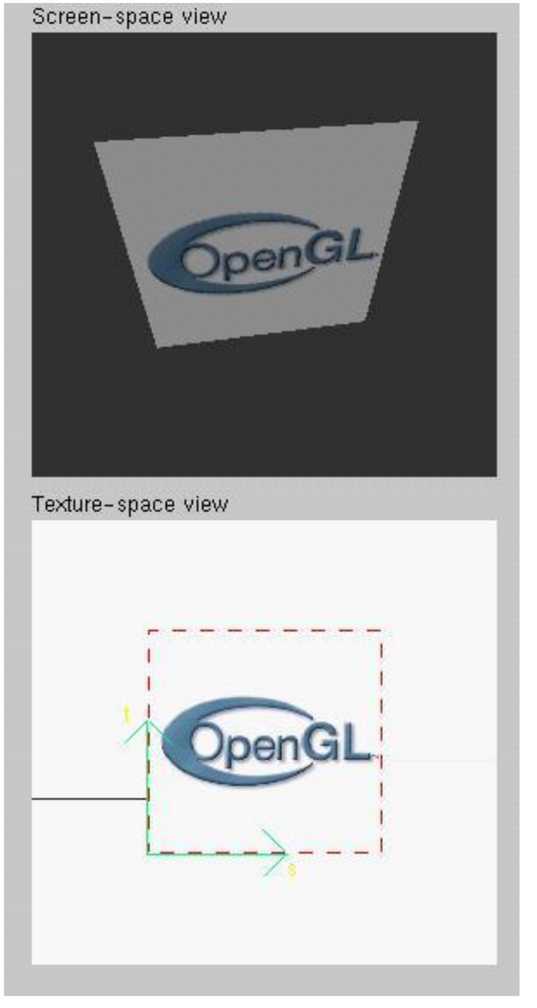


Specifying a texture image

- Define a texture image from an array of *texels* (texture elements) in CPU memory
  - **`Glubyte my_texels[512][512][4];`**
- Define a texture as any other pixel map
  - Scanned image
  - Generate by an application program
- Enable texture mapping
  - **`glEnable(GL_TEXTURE_2D)`**
  - OpenGL supports 1-4 dimensional texture maps

Texture example

- The texture (OpenGL logo) is a 256 x 256 image.
- It is mapped to a rectangular polygon.
- The polygon is viewed in perspective.



Defining image as a texture (1)

```
glTexImage2D(target, level, components,
     w, h, border, format, type, texels);
```

`target:` type of texture, e.g. `GL_TEXTURE_2D`

`level:` used for mipmapping (0 for only one/top resolution – to be discussed shortly)

`components:` colour elements per texel - an integer from 1 to 4 indicating which of the R, G, B and A components are selected for modulating or blending. A value of 1 selects the R component, 2 selects the R and A components, 3 selects R, G and B, and 4 selects R, B, G and A.

Defining image as a texture (2)

```
glTexImage2D(target, level, components,
    w, h, border, format, type, texels);
```

`w` and **`h`**: width and height of the image in pixels.

`border`: used for smoothing (discussed shortly), which is usually 0.

Both **`w`** and **`h`** must have the form $2^m + 2\mathbf{b}$, where **`m`** is an integer and **`b`** is the value of **`border`** though they can have different values. The maximum size of a texture map depends on the implementation of OpenGL, but it must be at least 64 × 64 (or 66 × 66 with borders).

Defining image as a texture (3)

```
glTexImage2D(target, level, components,
    w, h, border, format, type, texels);
```

`format` and **`type`**: describe the format and data type of the texture image data. They have the same meaning with **`glDrawPixels()`**. In fact, texture data has the same format as the data used by **`glDrawPixels()`**.

The **`format`** parameter can be **`GL_COLOR_INDEX`**, **`GL_RGB`**, **`GL_RGBA`**, **`GL_RED`**, **`GL_GREEN`**, **`GL_BLUE`**, **`GL_ALPHA`**, **`GL_LUMINANCE`**, or **`GL_LUMINANCE_ALPHA`** – i.e., the same formats available for **`glDrawPixels()`** with the exceptions of **`GL_STENCIL_INDEX`** and **`GL_DEPTH_COMPONENT`**.

Defining image as a texture (4)

```
glTexImage2D(target, level, components,
     w, h, border, format, type, texels);
```

Similarly, the **type** parameter can be **GL_BYTE**, **GL_UNSIGNED_BYTE**, **GL_SHORT**, **GL_UNSIGNED_SHORT**, **GL_INT**, **GL_UNSIGNED_INT**, **GL_FLOAT**, or **GL_BITMAP**.

texels: pointer to the texel array which contains the texture-image data. This data describes the texture image itself as well as its border.

For example:

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 514, 514, 1,
  GL_RGB, GL_UNSIGNED_BYTE, my_texels);
```

Texture parameters

OpenGL has a variety of parameters that determine how texture is applied:

- *Filter modes* allow us to use area averaging instead of point samples.
- *Wrapping parameters* determine what happens if s and t are outside the [0,1] range.
- *Mode/environment parameters* determine how texture mapping interacts with shading.
- *Mipmapping* (discussed shortly) allows us to use textures of multiple resolutions (levels of detail).

Controlling filtering (1)

- OpenGL allows us to specify any of several filtering options to determine the calculations. The options provide different trade-offs between speed and image quality.
- We can specify the filtering methods for magnification and minification independently.
- OpenGL can make a choice between magnification and minification that in most cases gives the best result possible.

Controlling filtering (2)

- The following examples show how to use **`glTexParameter*()`** to specify the magnification and minification filtering methods:

```
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

- The second argument is either **`GL_TEXTURE_MAG_FILTER`** or **`GL_TEXTURE_MIN_FILTER`** to indicate whether we are specifying the filtering method for magnification or minification. The third argument specifies the filtering method.

- Note that linear filtering requires a border of an extra texel for filtering at edges (border = 1).

Texture wrapping (1)

- We can assign texture co-ordinates outside the range [0,1] and have them either clamp or repeat in the texture map.

- With repeating textures, if we have a large plane with texture co-ordinates running from 0.0 to 10.0 in both directions, for example, we will get 100 copies of the texture tiled together on the screen. During repeating, the integer part of texture co-ordinates is ignored, and copies of the texture map tile the surface.

- For most applications of repeating, the texels at the top of the texture should match those at the bottom, and similarly for the left and right edges.

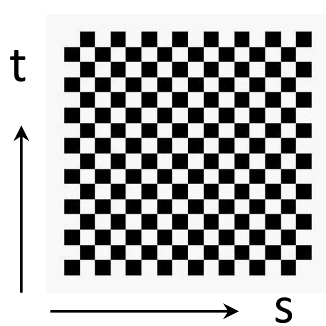
Texture wrapping (2)

- The other possibility is to clamp the texture co-ordinates: any values greater than 1.0 are set to 1.0, and any values less than 0.0 are set to 0.0.
- Clamping is useful when a single copy of the texture is to appear on a large surface.
- If the surface-texture co-ordinates range from 0.0 to 10.0 in both directions, one copy of the texture appears in the lower corner of the surface.
- The rest of the surface is painted with the texture border colours as needed.

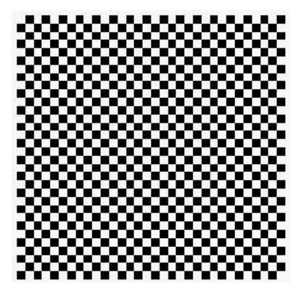
Texture wrapping (3)

Clamping: if s, t > 1 use 1, if s, t <0 use 0
Wrapping: use s, t modulo 1

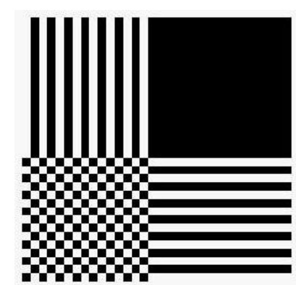
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
                GL_CLAMP)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
                GL_REPEAT)
```



Texture



GL_REPEAT wrapping



GL_CLAMP wrapping

Texture mode (1)

- Control how texture is applied
  `glTexEnv{fi}[v](GL_TEXTURE_ENV, prop, param)`

- `GL_TEXTURE_ENV_MODE` modes
  `GL_REPLACE` : use only texture colour
  `GL_MODULATE`: modulates with computed shade
  `GL_BLEND`: blends with an environmental colour

- Set blend colour with `GL_TEXTURE_ENV_COLOR`

Texture mode (2)

We can choose one of the three functions to compute the final RGBA value from the fragment colour and the texture-image data, by supplying appropriate arguments to

`glTexEnv{if}{v}(GLenum target, GLenum pname, TYPEparam)`,

- simply use the texture colour as the final colour, called the *decal* mode, in which the texture is painted on top of the fragment, just as a decal would be applied.

- use the texture to *modulate*, or scale, the fragment colour, useful for combining the effects of lighting with texturing.

- a constant colour can be blended with the fragment colour, based on the texture value.

Texture mode (3)

- The **`target`** must be **`GL_TEXTURE_ENV`**. If **`pname`** is **`GL_TEXTURE_ENV_MODE`**, **`TYPEparam`** can be **`GL_DECAL`**, **`GL_MODULATE`**, or **`GL_BLEND`**, to specify how texture values are to be combined with the colour values of the fragment being processed. In the decal mode and with a three-component texture, the texture colours replace the fragment colours.

- With either of the other two modes or with a four-component texture, the final colour is a combination of the texture and the fragment values. If **`pname`** is **`GL_TEXTURE_ENV_COLOR`**, **`TYPEparam`** is an array of four floating-point values representing the R, G, B and A components. These values are used only if the **`GL_BLEND`** texture function is specified as well.

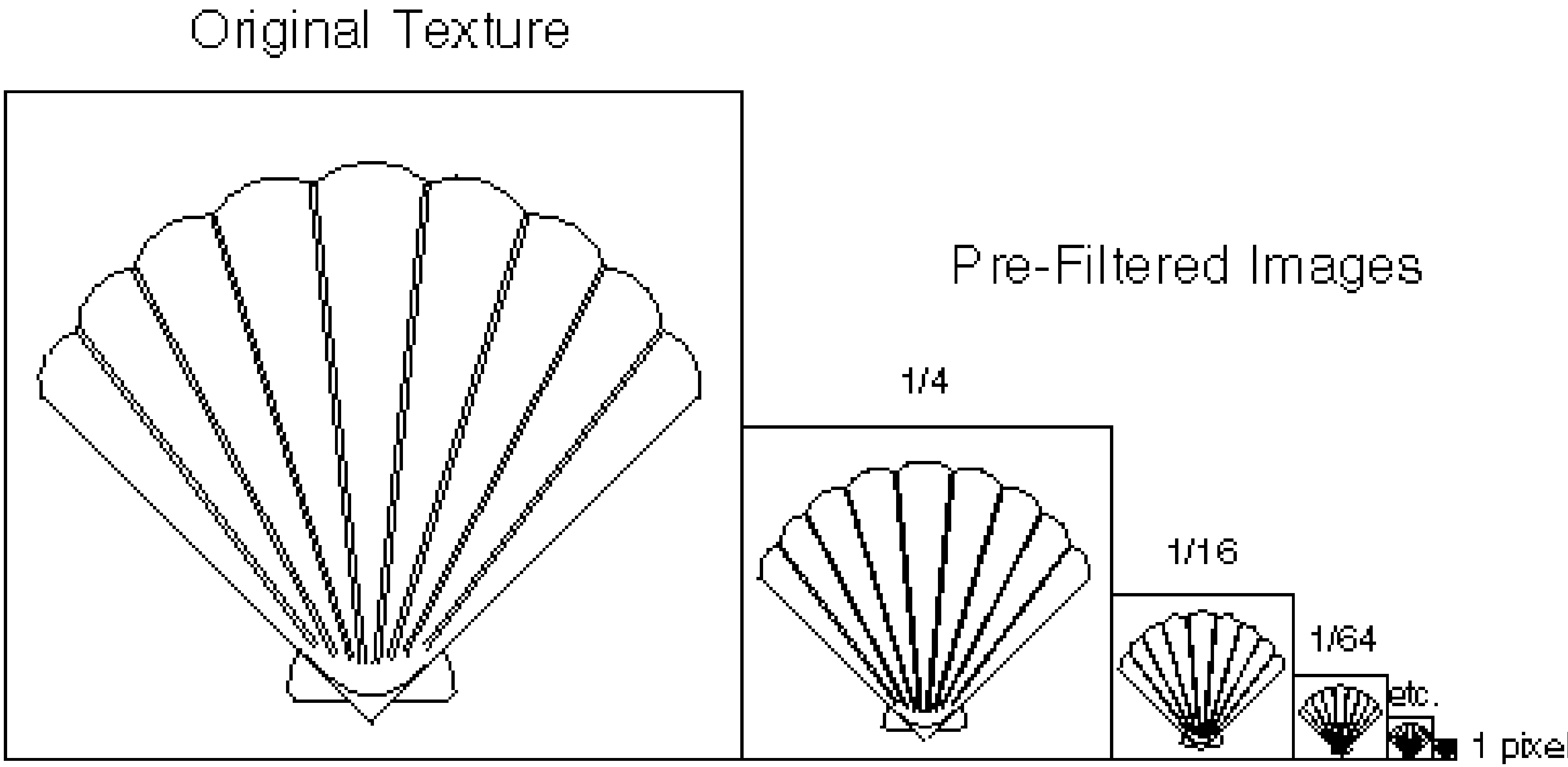
Mipmapping – Multiple levels of detail (1)

- Textured objects can be viewed, like any other objects in a scene, at different distances from the viewpoint. In a dynamic scene, as a textured object moves further from the viewpoint, the texture map must decrease in size along with the size of the projected image.

- To accomplish this, OpenGL has to filter the texture map down to an appropriate size for mapping onto the object, without introducing visually disturbing artefact.

- To avoid such artefact, we can specify a series of pre-filterred texture maps of decreasing resolutions, called *mipmaps*, as shown on the next slide.

Mipmapping – Multiple levels of detail (2)



Mipmapping – Multiple levels of detail (3)

- OpenGL automatically determines which texture map to use based on the size (in pixels) of the object being mapped.
- With this approach, the level of detail in the texture map is appropriate for the image that is drawn on the screen - as the image of the object gets smaller, the size of the texture map decreases.
- Mipmapping requires some extra computation to reduce interpolation errors, but, when it is not used, textures that are mapped onto smaller objects might shimmer and flash as the objects move.

Mipmapping – Multiple levels of detail (4)

- To use mipmapping, we provide all sizes of the texture in powers of 2 between the largest size and a 1×1 map.
- For example, if the highest-resolution map is 64×16, we must also provide maps of size 32×8, 16×4, 8×2, 4×1, 2×1 and 1×1.
- The smaller maps are typically filtered and averaged-down versions of the largest map in which each texel in a smaller texture is an average of the corresponding four texels in the larger texture.
- OpenGL does not require any particular method for calculating the smaller maps, so the differently sized textures could be totally unrelated.

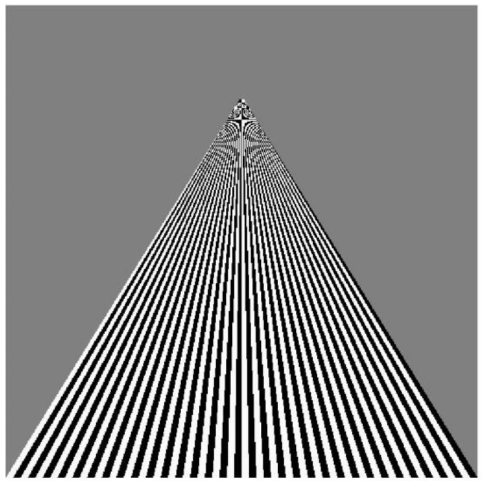
Mipmapping – Multiple levels of detail (5)

- Mipmapping level is declared during texture definition by calling **`glTexImage2D(GL_TEXTURE_*D, level, …)`** once for <u>each</u> resolution of the texture map, with different values for the **`level`**, **`width`**, **`height`** and **`image`** parameters.

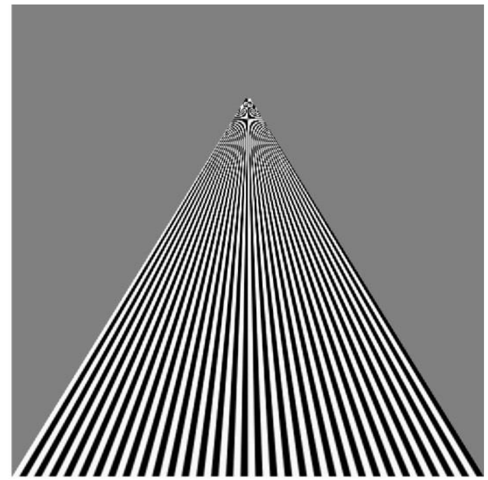
- Starting with zero, **`level`** identifies which texture in the series is specified; with the previous example, the largest texture of size 64×16 would be declared with **`level`** = 0, the 32×8 texture with **`level`** = 1, and so on.
- In addition, for the mipmapped textures to take effect, we need to choose an appropriate filtering method.

Mipmapping – Multiple levels of detail (6)

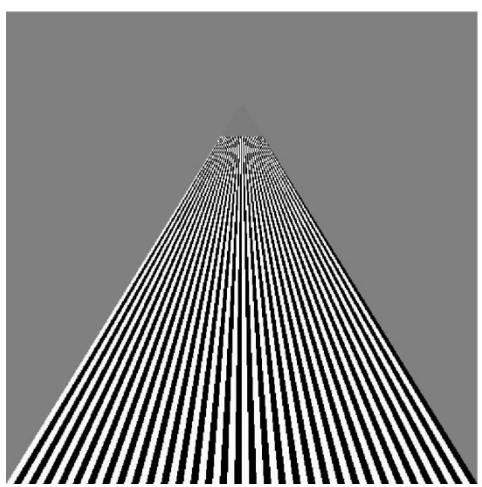
Point sampling



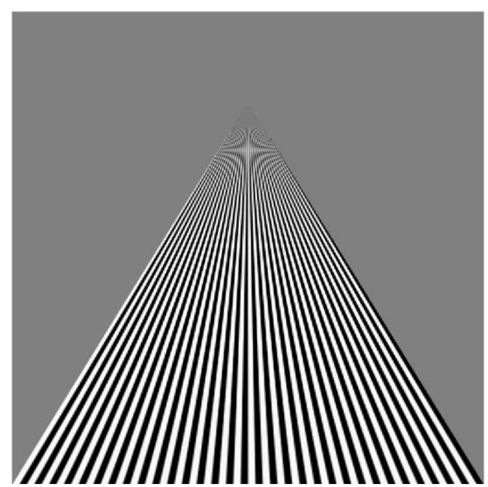
Linear filtering



Mipmapped point sampling



Mipmapped linear filtering



Generating texture co-ordinates (1)

- Texture co-ordinates are usually referred to as the *s, t, r* and *q* co-ordinates to distinguish them from object co-ordinates (*x, y, z* and *w*). For 2D textures, *s* and *t* are used. Currently, the *r* coordinate is ignored (although it might have meaning in the future). The *q* co-ordinate, like *w*, is typically given the value 1 and can be used to create homogeneous co-ordinates.

- We need to indicate how the texture should be aligned relative to the fragments to which it is to be applied. We can specify both texture co-ordinates and geometric co-ordinates as we specify the objects in the scene.

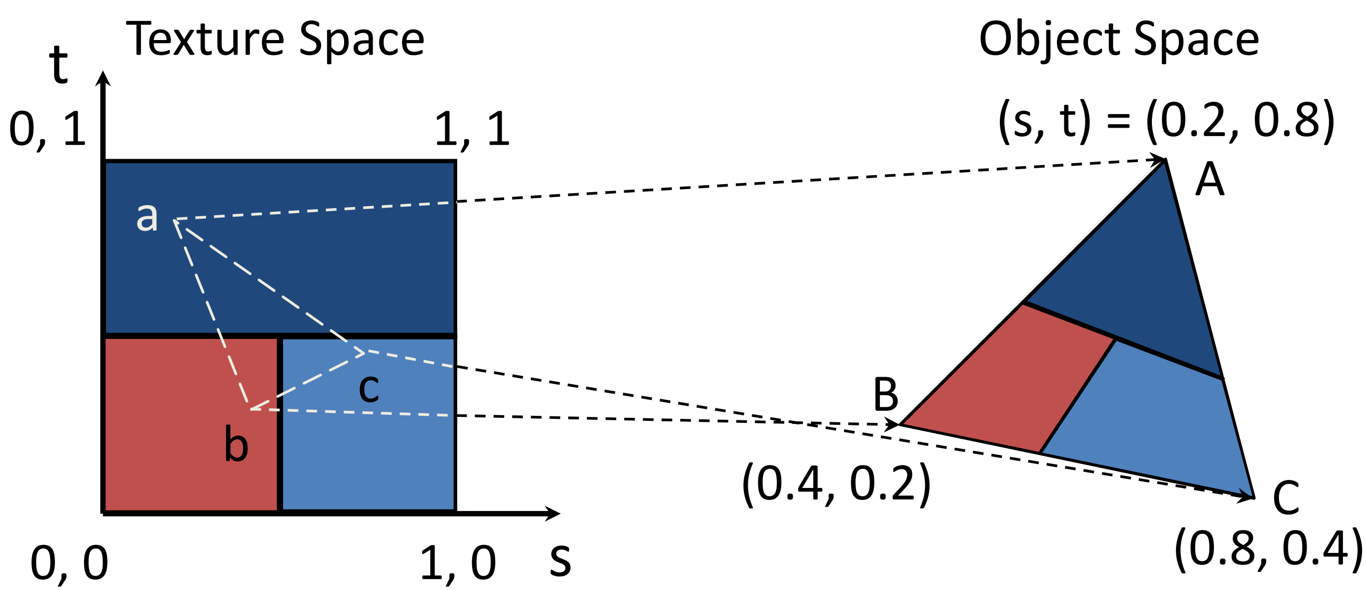
- The command to specify texture co-ordinates, **`glTexCoord*()`**, is similar to **`glVertex*()`**, **`glColor*()`** and **`glNormal*()`** - it comes in similar variations and is used in the same way between **`glBegin()`** and **`glEnd()`** pairs.

Generating texture co-ordinates (2)

- Usually, texture co-ordinate values range between 0 and 1.
- **`void glTexCoord{1234}{sifd}[v](TYPEcoords`**) sets the current texture co-ordinates (*s, t, r, q*).
- Subsequent calls to **`glVertex*()`** result in those vertices being assigned the current texture co-ordinates.
- Using **`glTexCoord2*()`** allows us to specify *s* and *t*; *r* and *q* are set to 0 and 1, respectively. We can supply the co-ordinates individually, or we can use the vector version of command to supply them in a single array. Texture co-ordinates are multiplied by the 4x4 texture matrix before any texture mapping occurs.

Generating texture co-ordinates (3)

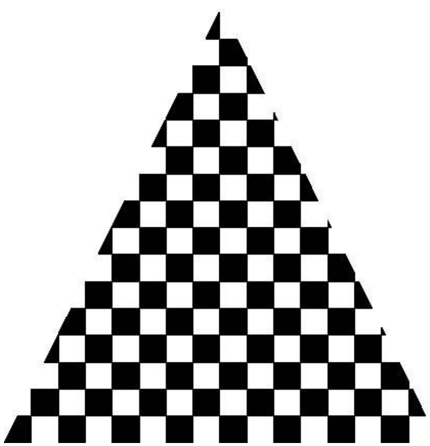
Based on parametric texture co-ordinates, `glTexCoord*()` is specified at each vertex.



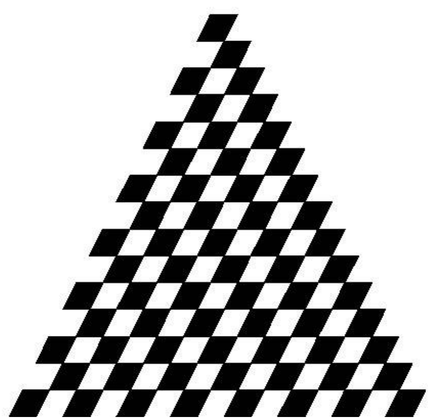
Generating texture co-ordinates (4)

- OpenGL uses interpolation to find proper texels from specified texture coordinates.
- There can be distortions.

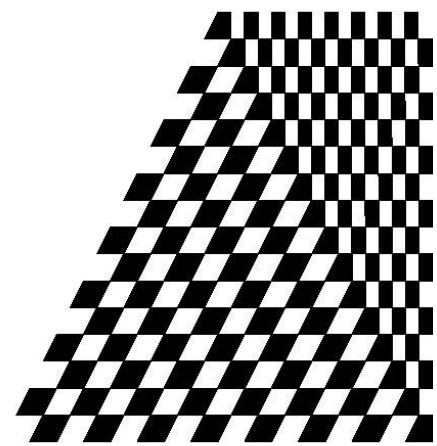
Good selection of texture co-ordinates



Poor selection of texture co-ordinates



Texture stretched over trapezoid showing effects of bilinear interpolation



Generating texture co-ordinates (5)

- OpenGL can generate texture co-ordinates automatically,
  `glTexGen{ifd}[v](GLenum coord, GLenum pname, GLint param)`
- Specify a plane and generates texture coordinates based on the distance from the plane
- **`coord`**: one of **`GL_S, GL_T, GL_R, or GL_Q`**
- **`pname`**: generation modes, one of
  `GL_OBJECT_LINEAR`
  `GL_EYE_LINEAR`
  `GL_SPHERE_MAP` (used for environmental maps)

Perspective correction hint

- Texture co-ordinate and colour interpolation
  - either linearly in the screen space
  - or using depth/perspective values (slower)

- One of the hint types

  `glHint(GL_PERSPECTIVE_CORRECTION_HINT, hint)`

  where **`hint`** is one of

  ```
  GL_DONT_CARE
  GL_NICEST
  GL_FASTEST
  ```

Summary

- Why texture mapping is needed and how it works
- Techniques for texture mapping
  - Specifying the texture
  - Magnification and minification
  - Two-step mapping
  - Multiple levels of detail (mipmapping)
  - Modes, filtering and wrapping
- Steps for texture mapping
- OpenGL functions