Discrete Techniques in Rasterisation

Angel Chapter 9

Discrete Techniques

Operations using the 2D image buffer

- several buffers: frame, depth ….

Surface Mapping

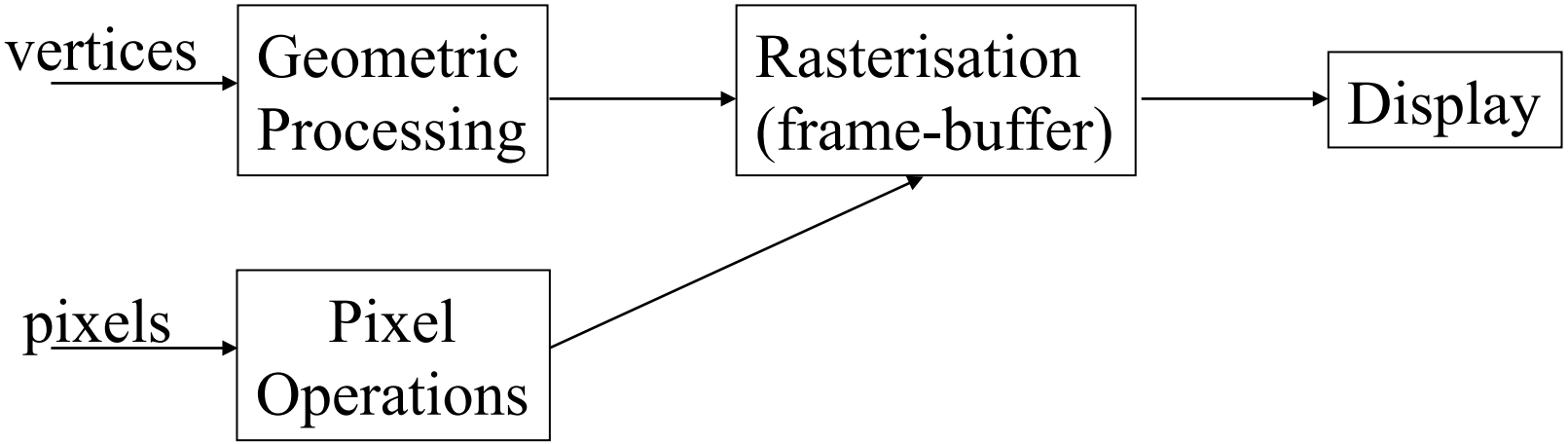
- change surface appearance at the image level
- texture mapping, bump mapping…..

Composting - overlay multiple images

Transparency - simulate opacity of surface by combining multiple layers

Antialiasing - shade pixels which contain multiple surfaces

Discrete techniques are applied at the rasterisation stage of the graphics pipeline (ie directly to the frame-buffer)



Pixel operations modify individual pixels in a buffer

Buffer is a discrete 2D grid of NxM pixels (picture elements)

- Pixel has k bits (byte, integer, float)
- k NxM **bitplanes**

Surface Mapping

Modify the surface of a geometric object at the raster level

- Geometric objects have smooth surfaces
- Requires large number of polygons to model detailed surface ie small bumps, texture etc.
- Add detail as part of the rendering process by modifying appearance at the pixel level
- 'Mapping' surface properties changes apperances (colour, normal, reflectance at pixel level)

Mapping techniques:

(1) Texture mapping - modify colour based on an image
(2) Environment mapping - modify colour from scene reflection
(2) Bump mapping - modify surface normals (shading)
(3) Displacement mapping - modify surface geometry

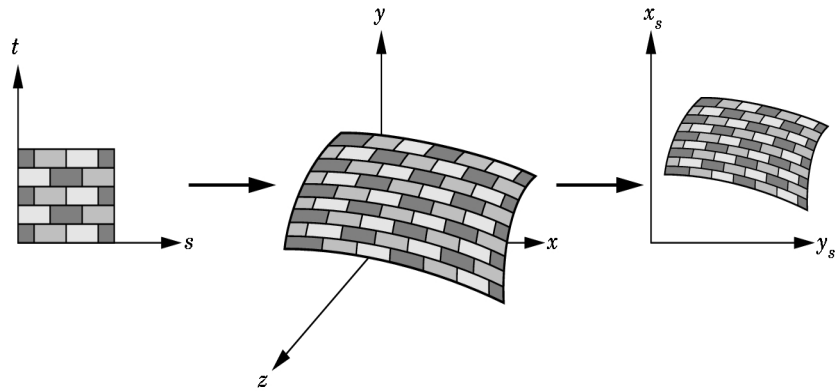
Texture Mapping

Application of a colour pattern to the geometric object surface

- pattern (texture) may be defined in 1D, 2D or 3D
- modify colour of rendered surface for pixels in frame-buffer to colour of texture (or a combination of shading + texture)

Two-Dimensional texture mapping

- map a 2D texture image T(s,t) onto surface
- Texture coordinates $0<s,t<1$
- T is a PxQ 2D grid of **textels** (texture elements)
- **Texture map** associates points (s,t) of T with geometric coordinates (x,y,z) of scene objects



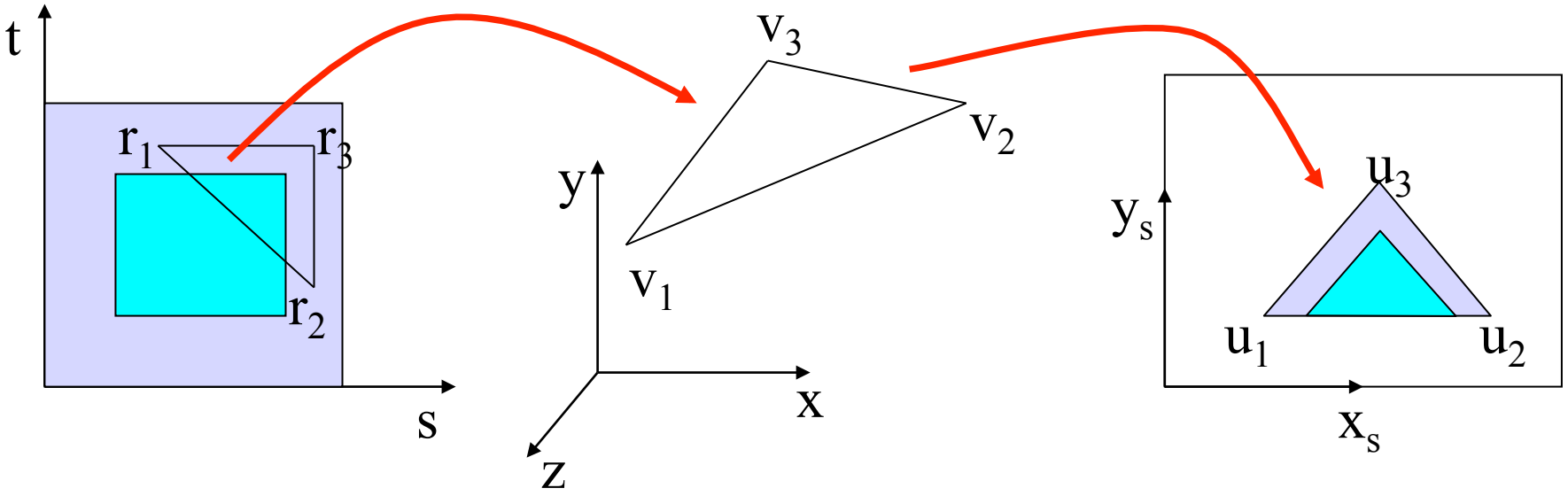
Texture mapping of polygonal objects

Polygonal object is defined by a set of vertices $v_i = (x_i, y_i, z_i) \quad i = 1,\ldots, N_v$

For each vertex we define a corresponding texture coordinate

$$r_i = (s_i, t_i) \quad i = 1,\ldots, N_v$$

Each polygon is then mapped by colour of corresponding region in texture map



Interpolation of texture coordinates

How to fill polygon surface with texture?

Texture mapping defines the mapping from texture to object coordinates
Model transformation + projection defines mapping to screen coordinates

$$r_i(s_i, t_i) \quad \Rightarrow \quad v_i(x_i, y_i, z_i) \quad \Rightarrow \quad u_i(x_{si}, y_{si})$$

To fill a polygon we require inverse mapping from screen to texture coordinates

$$u(x_s, y_s) \quad \Rightarrow \quad v(x, y, z) \quad \Rightarrow \quad r(s, t)$$

Use interpolation from vertex coordinates to map screen coordinates to texture coordinates.

Linear interpolation

$$u = \alpha u_1 + \beta u_2 + (1-\alpha-\beta)u_3$$
$$v = \alpha v_1 + \beta v_2 + (1-\alpha-\beta)v_3$$
$$r = \alpha r_1 + \beta r_2 + (1-\alpha-\beta)r_3$$
$$0 \le \alpha, \beta \le 1$$

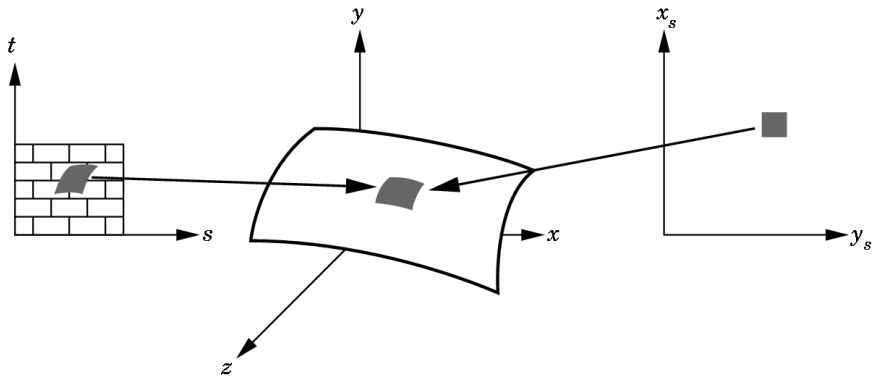
Coefficients determined by interpolation method

- barycentric coordinates
- bi-linear interpolation

Texture map aliasing

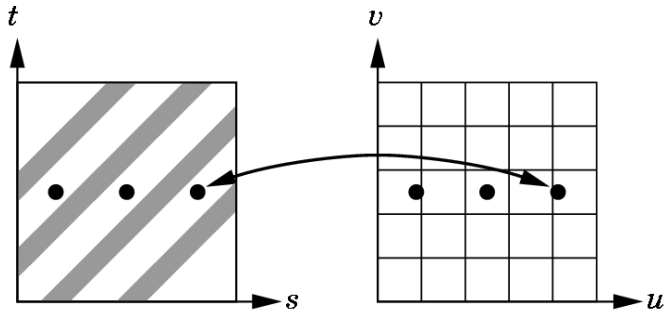
Inverse mapping gives the corresponding point in texture space
to point in screen space.

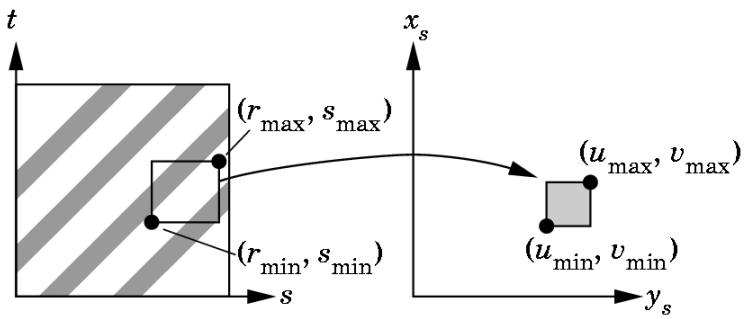
Pixels cover an area in screen space.
Require mapping of area in screen space to area in texture space
- mapping based on points only can give aliasing



Possible assignment of pixel colour
- inverse map of pixel centre => aliasing (periodic textures)
- average texture map colour over inverse map of pixel area (blur)

due to discrete frame buffer aliasing occurs if: texture freq. > 1/2x sampling resolution





Texture Mapping Complete Objects

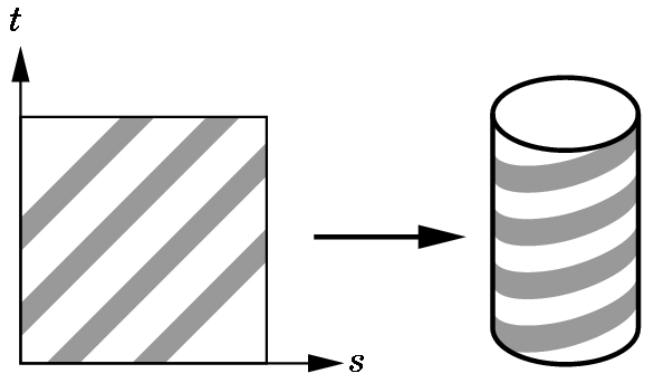
Normally texture image is applied across complete surface of object
Requires texture coordinates to be defined that wrap around surface

Define a simple intermediate 3D surface (cube, sphere, cylinder) to which object coordinates can be mapped

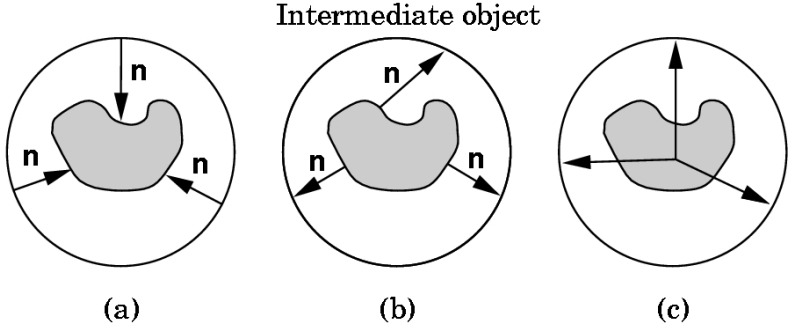
Cylindrical texture map

h - height
r - radius
(s,t) texture coordinates

$$x = r\cos(2\pi s)$$
$$y = r\sin(2\pi s)$$
$$z = \frac{t}{h}$$



maps (x,y,z) object coordinates to (s,t) texture coordinates on surface of a cylinder



Spherical texture map

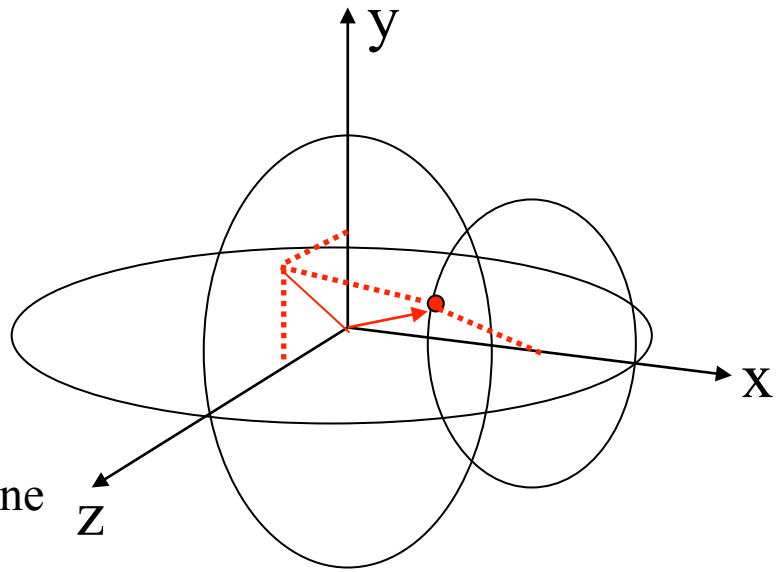
- no 2D parameterisation of spherical surface without singularities
- distorts surface
- Mercator projection (latitude/longitude) distorts map of earth at poles

Sphere radius r

$$x = r\cos(2\pi s)$$

$$y = r\sin(2\pi s)\cos(2\pi t)$$

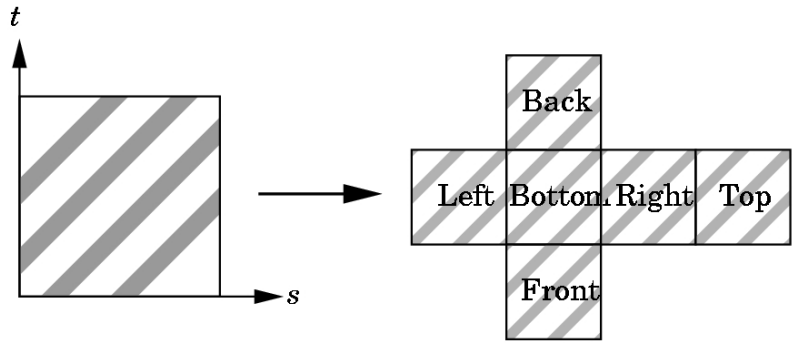
$$z = r\sin(2\pi s)\sin(2\pi t)$$



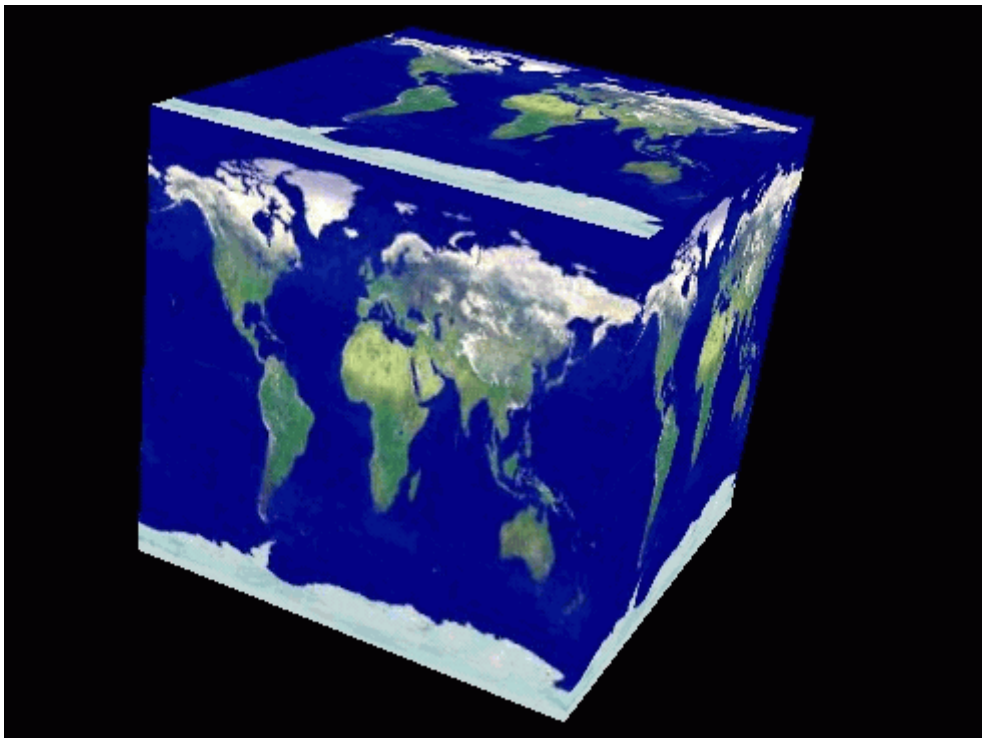
$2\pi s$ - angle to x-axis

$2\pi t$ - angle from y-axis in ys plane

Texture map to a cube



Examples of texture mapping



Texture Mapping in OpenGL

Supports mapping of 1, 2 and 3 dimensional textures

Pipeline architecture - pixel pipeline in parallel with geometric operations
- mapped onto geometric primitives at rasterisation stage

Geometric pipeline maps 3D primitives to pixels on the display (frame-buffer)

For each pixel:

(i) perform visibility test using the z-buffer
(ii) if visible the pixel shading is computed (from reflection model)
(iii) map vertices to texture coordinates
(iv) interpolate texture value for pixel
(v) combine texture colour and shading to give final pixel colour

Implementation of Texture Mapping in OpenGL

(1) Enable texture mapping

glEnable(GL_TEXTURE_2D);

(2) Define texture (generate or load from file)

Glubyte mytexture[512][512];

(3) Specify texture to be used

glTexImage2D(GL_TEXTURE_2D,level,components, width,height,border, format, type,mytexture);

width x height array 'mytexture'

components - number of colour components

format - texture form ie GL_RGB, GL_RGBA

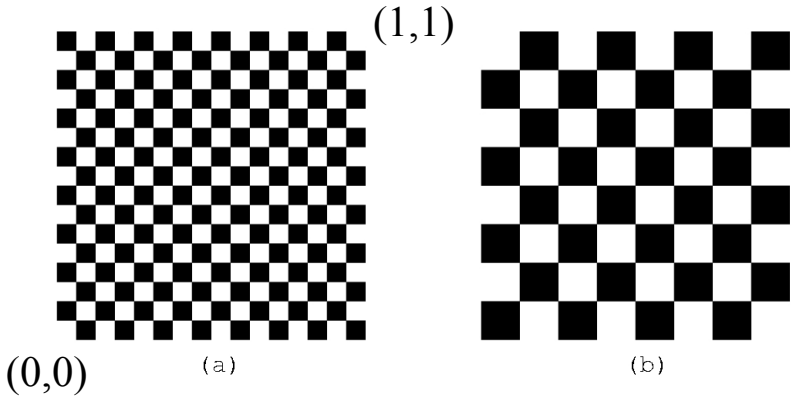
type - type of each component

level - used for multiple resolution textures

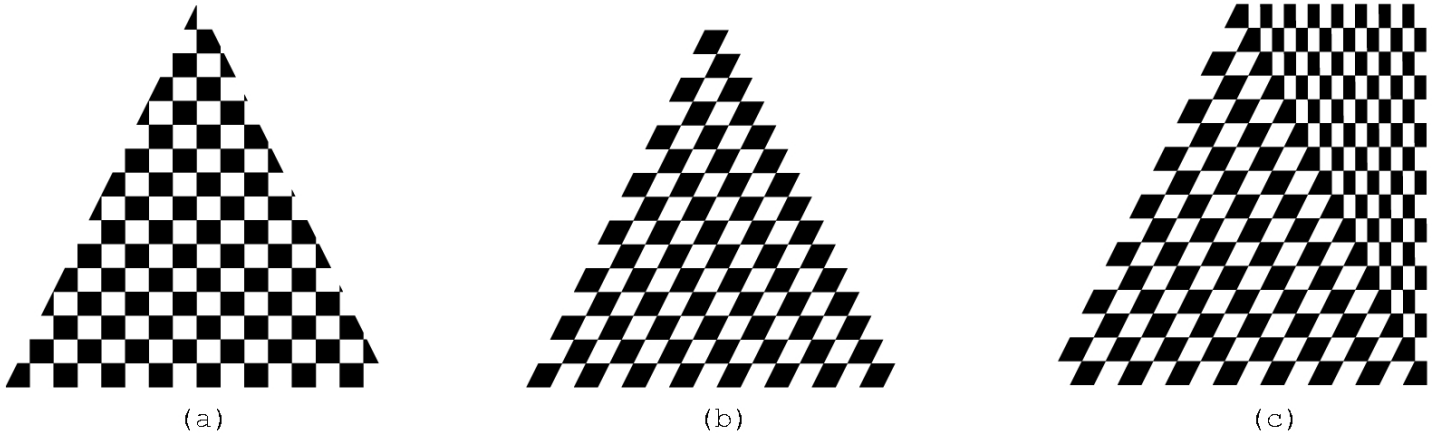
(4) Assign texture coordinates to vertices (0,1) => (0,width) or (0,height)

```
glBegin(GL_TRIANGLES);
    glTexCoord2f(s1,t1);
    glVertex2f(x1,y1,z1);
      ……..
glEnd();
```

Mapping with texture coordinates



Given texture (a) above what are the texture coordiates used to create each of the following?



Texture parameters in OpenGL

Wrapping (s,t) <0 or (s,t)>1

- clamp coordinates: s>1 => s=1

glTexParameter(GL_TEXTURE_WRAP_S,GL_CLAMP);

- wrap: s>1 => s = s%1

glTexParameter(GL_TEXTURE_WRAP_S,GL_REPEAT);

Sampling of Texture Image

- pixel on screen is smaller or larger than texel
- specify the sampling strategy to use nearest (fast) or linear interpolation

GL_NEAREST - fast takes texture value at nearest pixel centre

GL_LINEAR - average of 4 closest pixels

magnify texture

glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,GL_NEAREST);

minimise texture

glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,GL_NEAREST);

MipMapping

Define texture at multiple resolution levels

- use level nearest to required pixel sampling
- Sub-sample the image by factors of 2^l
- OpenGL automatically uses appropriate resolution

GLU function to build mipmaps down to l=0 1x1 image

gluBuild2DMipmaps(GL_TEXTURE_2D,
components, width,height,format,type,mytexture);

Mipmaps are automatically used if we set:

glTexParameterf(GL_TEXTURE_2D,
GL_TEXTURE_MIN_FILTER,GL_NEAREST_MIPMAP_NEAREST);

Texture Mapping Properties in OpenGL

Shading/Texture Integration

The colour of a pixel is determined by both shading colour and texture colour

To combine both shading and colour (default)

*glTexEnv(GL_TEX_ENV,GL_TEX_ENV_MODE,**GL_MODULATE**);*

To use just texture colour

*glTexEnv(GL_TEX_ENV,GL_TEX_ENV_MODE,**GL_DECAL**);*

Perspective Correction

By default OpenGL uses linear interpolation in screen space (orthographic)

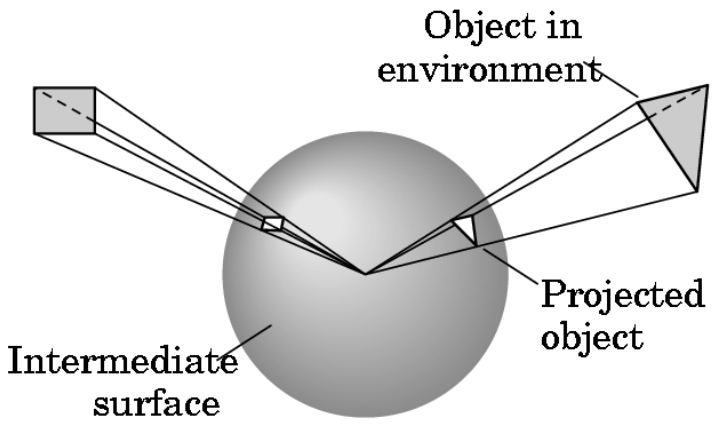
To include non-linear depth scaling due to perspective projection use a better interpolation scheme (if supported)

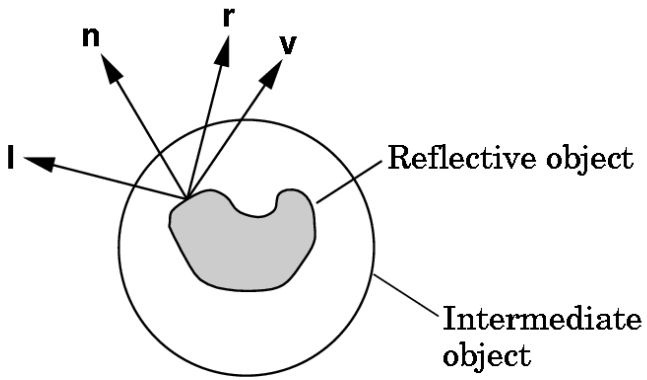
glHint(GL_PERSPECTIVE_CORRECTION,GL_NICEST);

Environment Maps

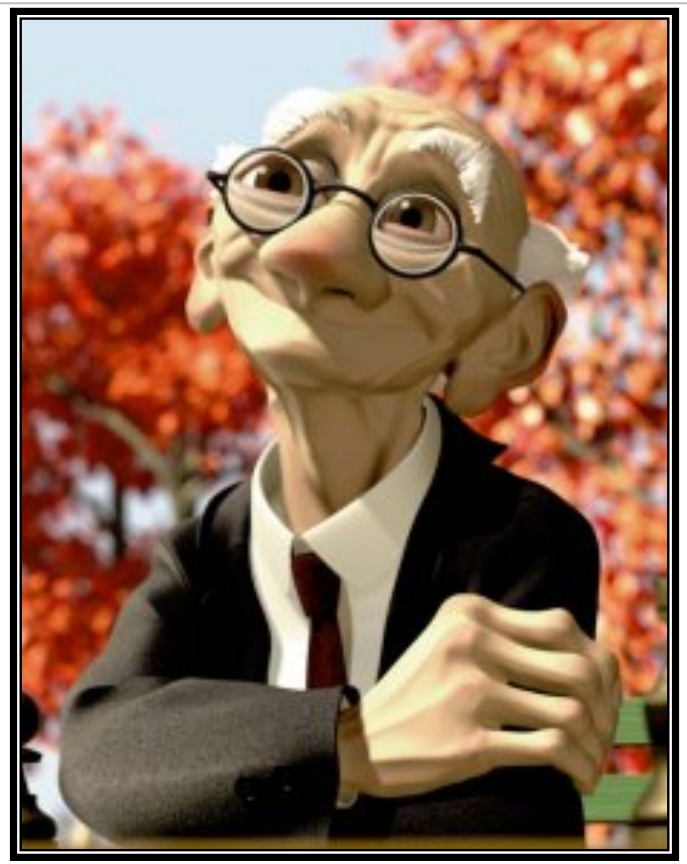
Texture map object surface with reflection of scene

- simulates highly reflective surface (mirror the environment)
- approximate effect of ray-tracing
- generate synthetic environment map by projecting the scene onto an object (sphere) located at the centre of an object
- apply synthetic environment map to object surface as a texture of surface colour with reflection computed by Phong model





Examples of Environment Maps

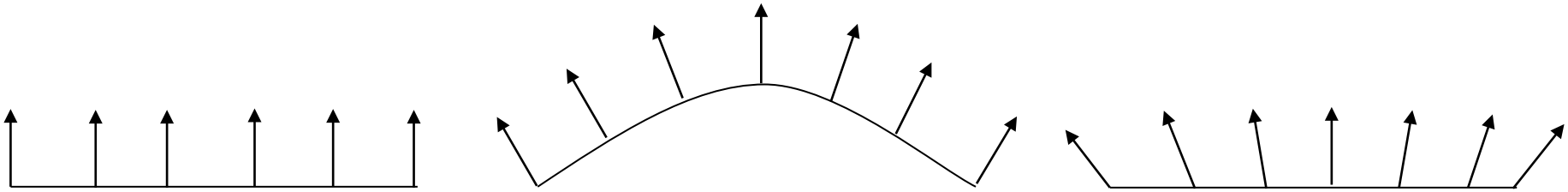


Bump Maps

Changes the apparent shape of the surface by varying the surface normals

- surface bumps result in different shading across the surface
- simulated by varying surface normals
- normals are used in shaping computation of Phong reflection model
  changing the normal result in a change in surface appearance

ie a flat surface can be shaded like a sphere setting normals appropriatly



Reflected light is proportional to angle between surface normal
and view direction

- surface shading of curved surface is simulated by surface normal

Bump Map Implementation for Parametric Surfaces

$p(u,v) = (x(u,v), y(u,v), z(u,v))$ point on the surface defined by 2 parameters (u,v)

The surface normal is given by: $n(u,v) = \dfrac{p_u \times p_v}{\|p_u \times p_v\|}$

$$p_u = \begin{bmatrix} dx/du \\ dy/du \\ dz/du \end{bmatrix} \quad p_v = \begin{bmatrix} dx/dv \\ dy/dv \\ dz/dv \end{bmatrix}$$

Let d(u,v) be a displacement function,
the resulting displaced surface is given by: $p'(u,v) = p(u,v) + d(u,v)n(u,v)$

$$n'(u,v) = \frac{p'_u \times p'_v}{\|p'_u \times p'_v\|}$$

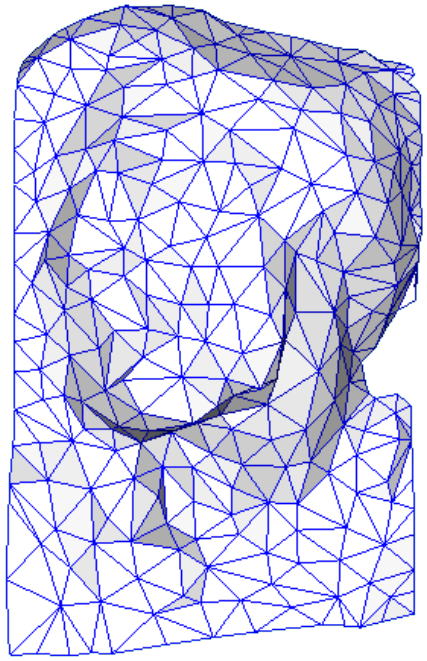
$$p'_u = p_u + \frac{\partial d(u,v)}{du} n(u,v) + d(u,v)n_u$$

If the displacement d is small then: $n'(u,v) = n + \dfrac{\partial d}{\partial u} n \times p_v + \dfrac{\partial d}{\partial v} n \times p_u$

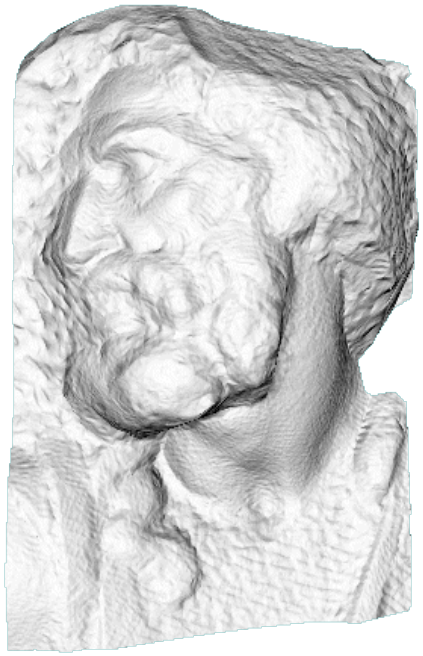
Using the normal n’(u,v) with the surface p(u,v) simulates the appearance of p’(u,v

Bump Mapping Example

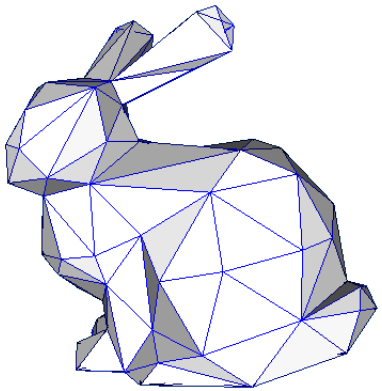
Original



Simplified Geometry



Bump Mapping

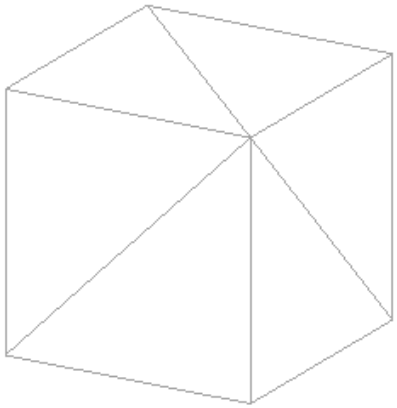


Displacement Mapping

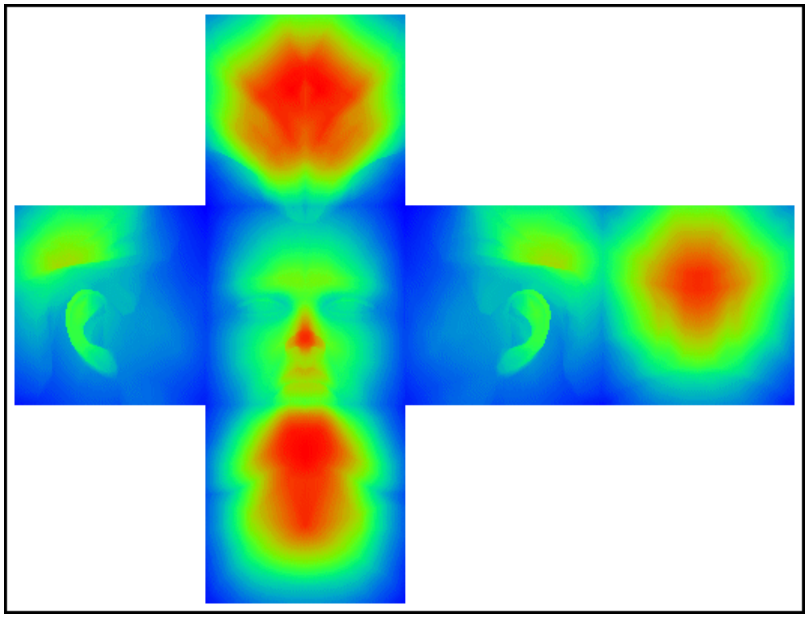
Apply a local displacement function d(u,v) to a smooth surface p(u,v)

- d(u,v) may be a displacement image (analogous to texture)
- efficient representation of local surface detail

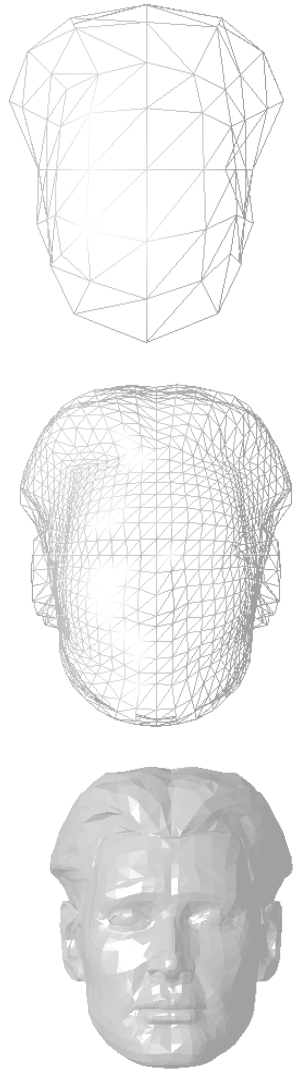
$$p'(u,v) = p(u,v) + d(u,v)n(u,v)$$



Base Model

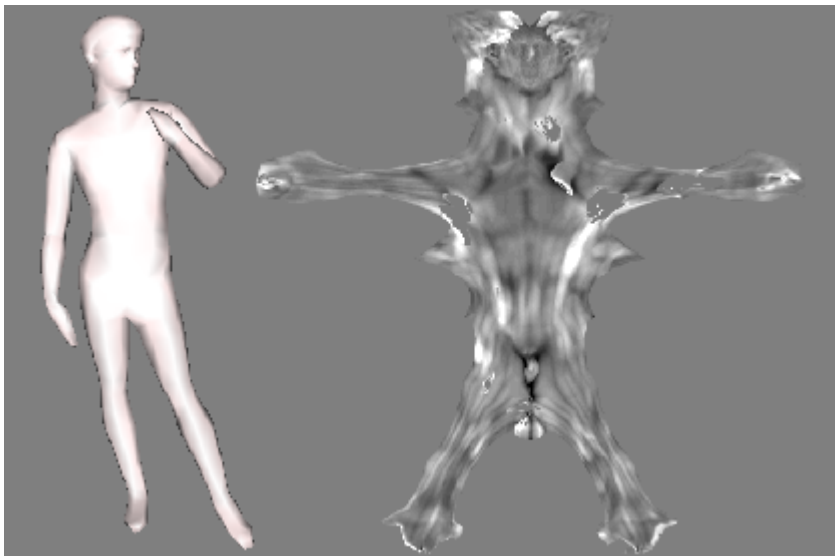


Displacement map (psuedo colour)



Displacement Mapping of Michelangelo's David

Original
>100Mb



Automatic Model Generation
Control Model + Displacement Map

Reconstruction
<1Mb

Data Courtesy of Stanford Computer Graphics Lab.

Compositing Techniques

Combination of multiple overlapping images or layers
- combine foreground/background image
  ie blue-screen studio
- simulate effect of transparency

Blending or Compositing
- multiple images can be combined together

$$I = \alpha I_1 + (1-\alpha) I_2$$

Alpha component 'A' of RGBA colour provides a blending factor $\alpha = A$

Foreground
(blue A=0)

Background

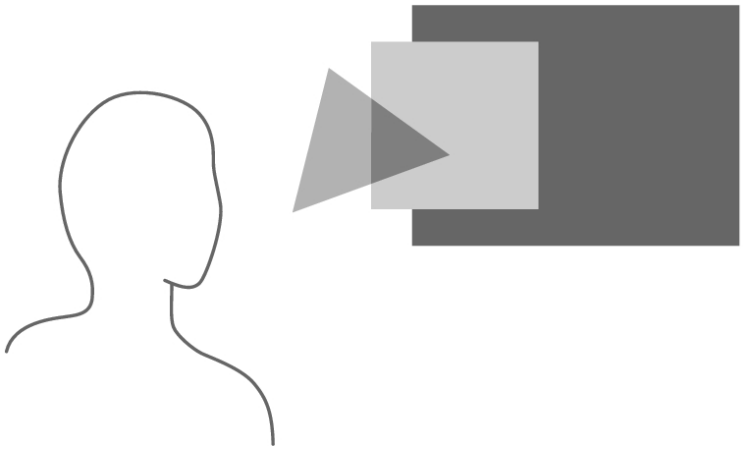
Composite

Surface Transparency

Opacity of a surface is a measure of how much light passes through

Let α be an opacity coefficient such that α=1 no light passes through (opaque)
α=0 all light passes through (transparent)

Transparency or **translucency** =(1- α)



Can simulate multiple overlapping semi-transparent surface by blending together their colour for each pixel:

$$c(r,g,b) = \alpha_1 c_1 + \alpha_2 c_2 + \alpha_3 c_3 + ...$$

Surface Transparency in OpenGL

1) Enable blending

glEnable(GL_BLEND);

2) Setup blending coefficients for source (new object) and destination (current content of frame-buffer)

glBlendFunc(source_factor,destination_factor)

factor can be GL_ONE, GL_ZERO,
GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA
GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA

Note: resulting image depends on order in which polygons are blended
must control order of rendering

Summary

- Discrete techniques applied to geometric primitives at rasterisation

- Mapping to change surface properties (colour, reflectance,shape)
  - Texture
  - Environment
  - Bump
  - Displacement

  Texture mapping is widely used in film games to create highly realistic appearance with few polygons

- Compositing or Blending
  - Simulate transparent surfaces
  - combine foreground/background images
  - reduce aliasing