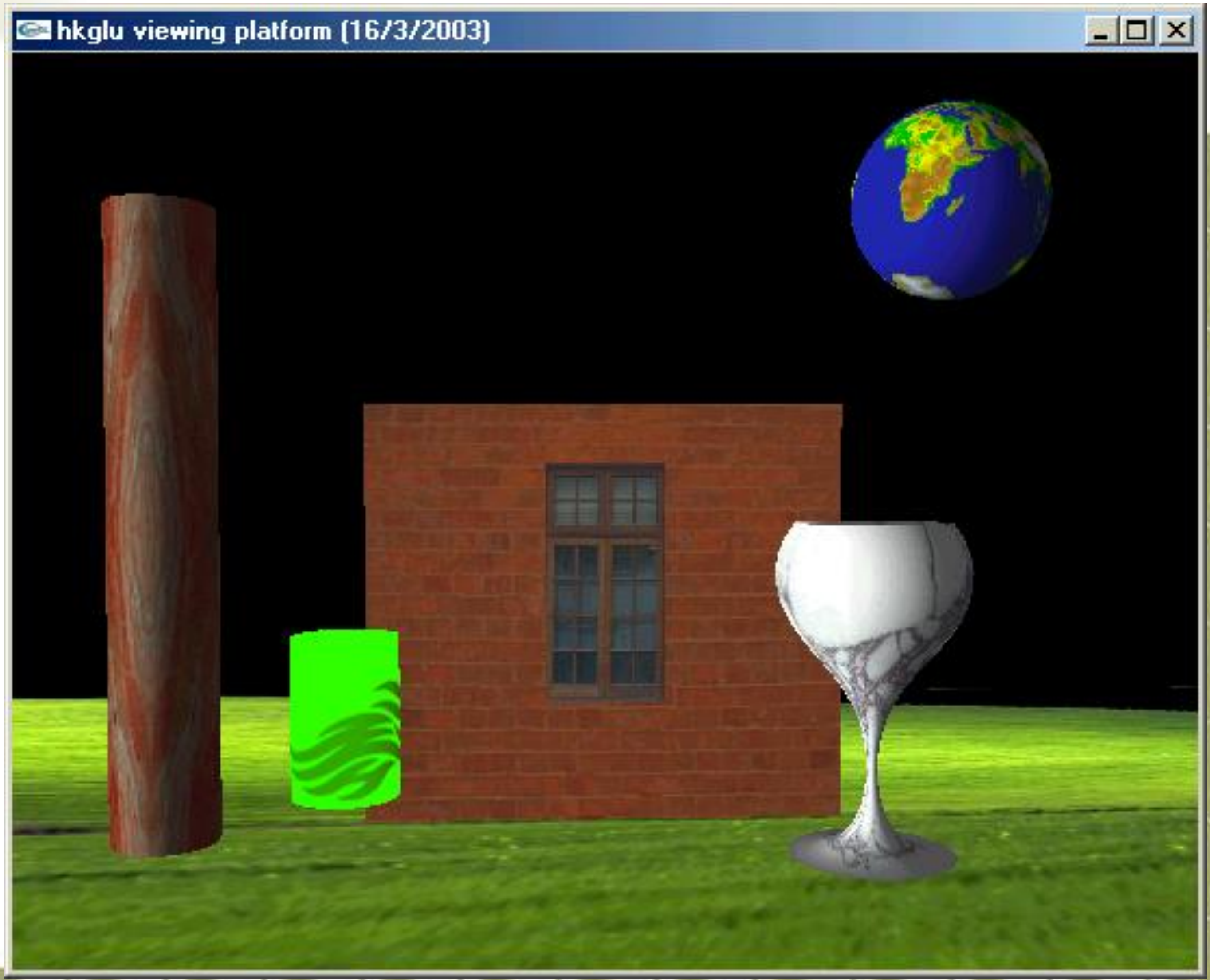
Chapter 7: Discrete Techniques

Creating more realistic and exciting images

Uses of Texturing

- simulating materials
- reducing geometric complexity

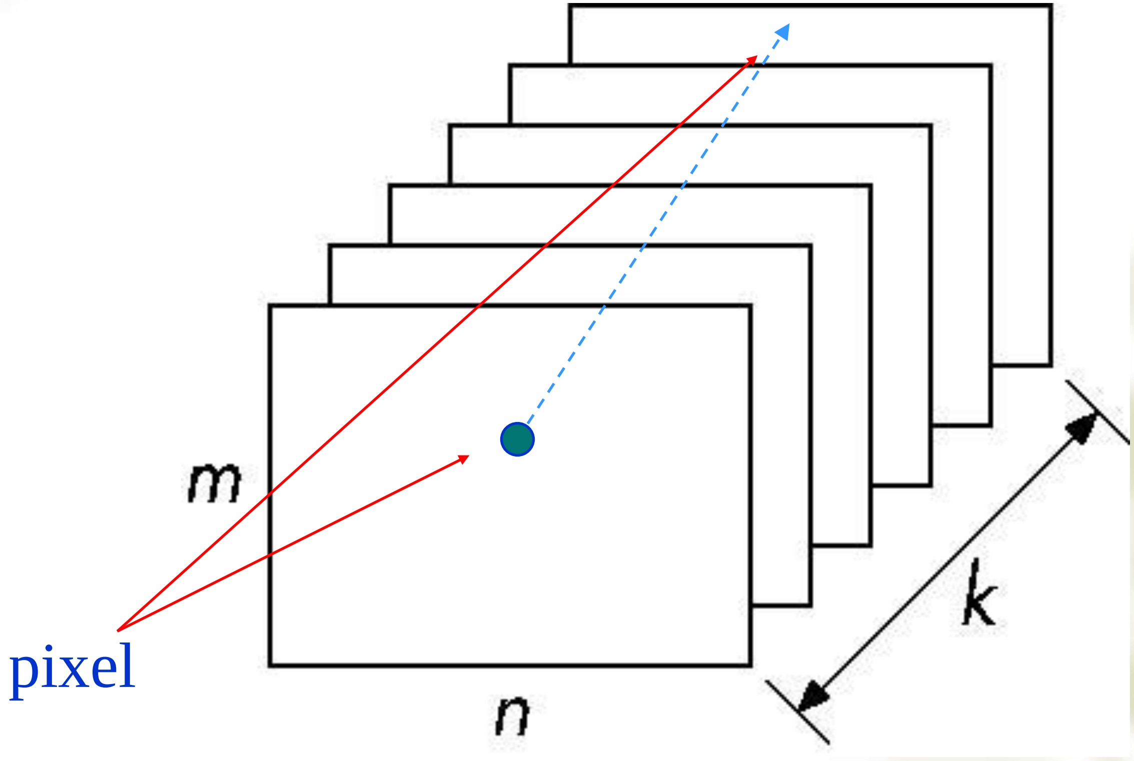


Key Contents

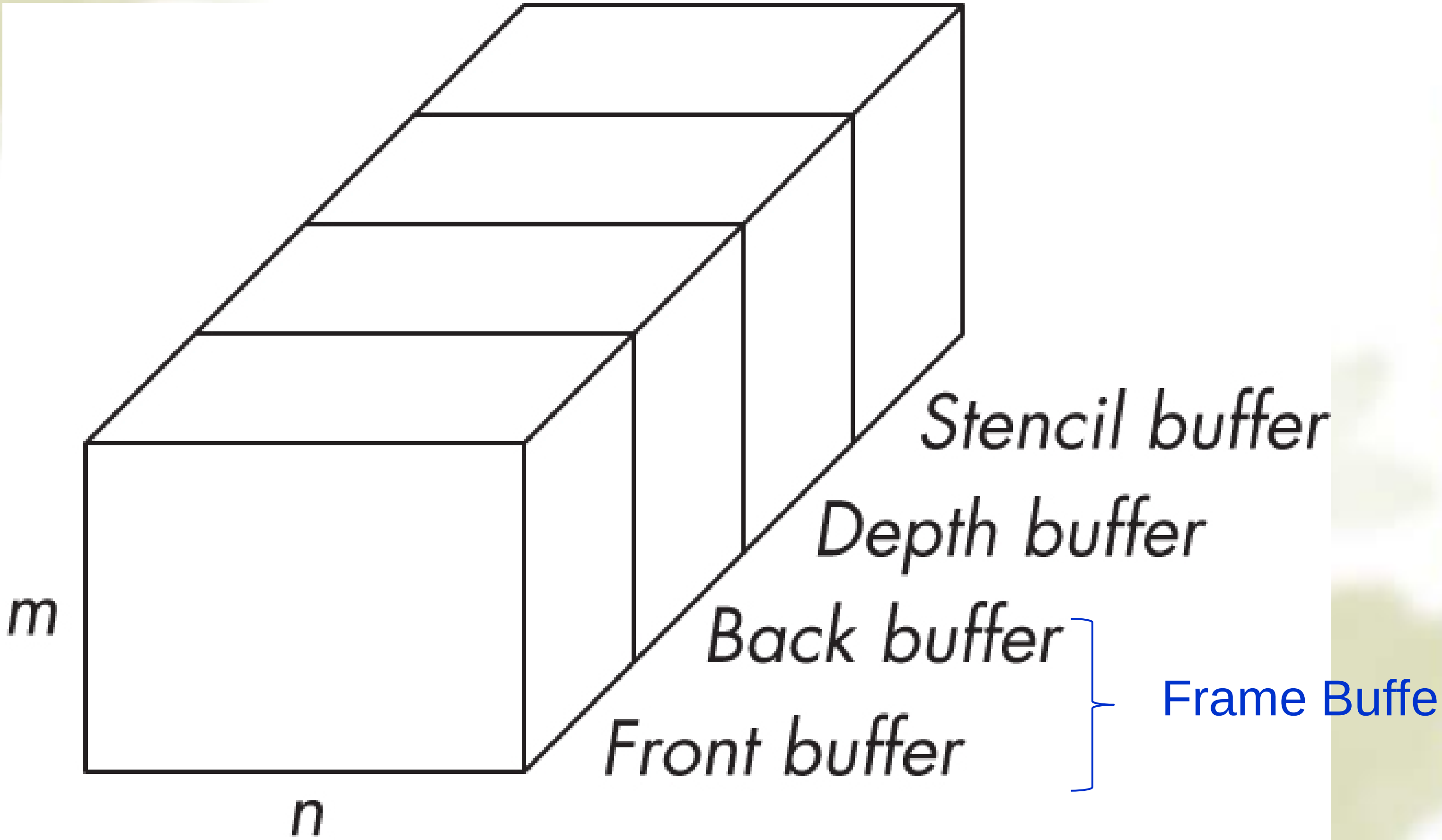
1. Buffers in OpenGL
2. Writing in Buffers
3. Mapping Methods
4. Texture Mapping in OpenGL
5. Texture Objects in OpenGL
6. The Example of Texture
7. Blend Model

1. Buffer in OpenGL

Define a buffer by its spatial resolution (n x m) and its depth (or precision) k, the number of bits/pixel



OpenGL Frame Buffer

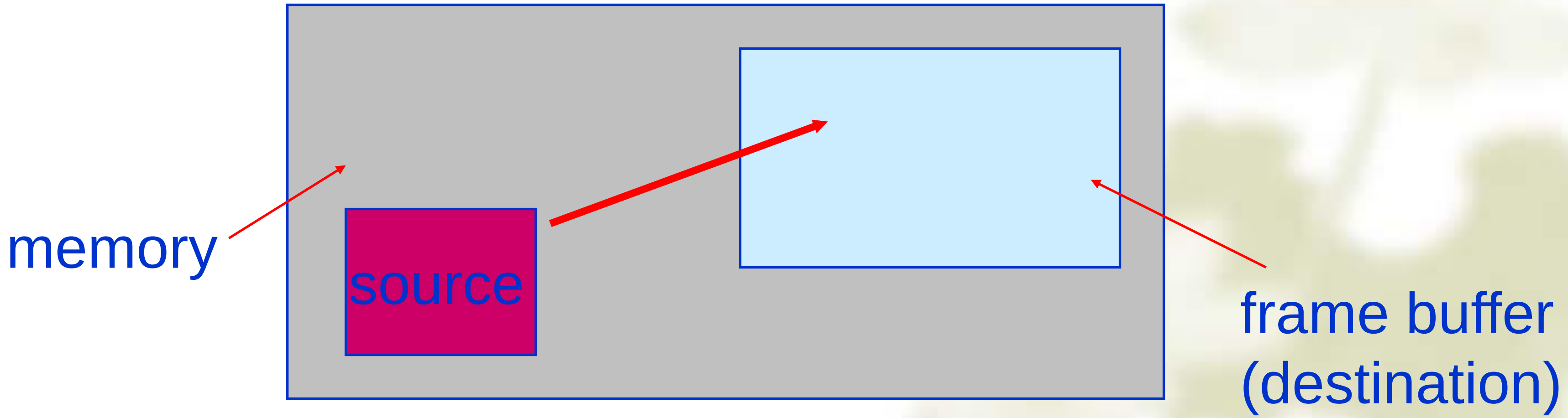


OpenGL Buffers

- Color buffers can be displayed
  - Front
  - Back
  - Auxiliary
  - Stereo
- Depth
- Stencil
  - Holds masks
- Most RGBA buffers 8 bits per component
- Latest are floating point (IEEE)

2. Writing in Buffers

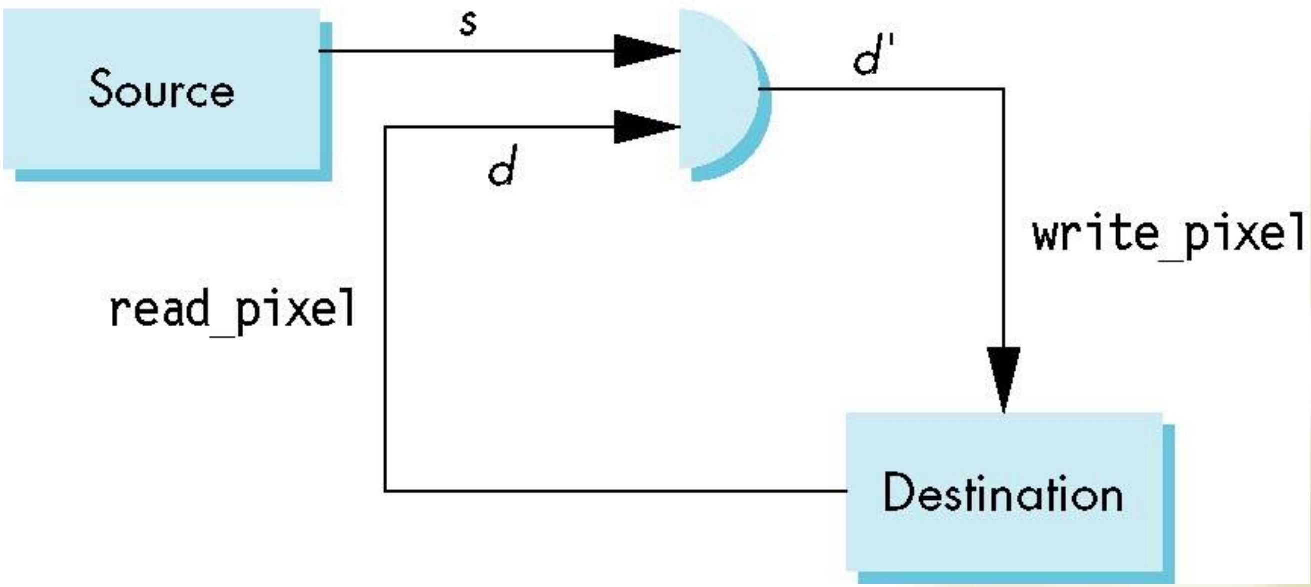
- Conceptually, we can consider all of memory as a large two-dimensional array of pixels
- We read and write rectangular block of pixels
  - *Bit block transfer* (*bitblt*) *operations*
- The frame buffer is part of this memory



writing into frame buffer

Writing Model

Read destination pixel before writing source



Bit Writing Modes

- Source and destination bits are combined bitwise
- 16 possible functions (one per column in table)

  glLogicOp(mode); -- default: mode=GL_COPY

		CLEAR	AND		s		SET/COPY	XOR	OR	NOR				NOR s			
s	*d*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

XOR mode

- Recall from Chapter 3 that we can use XOR by enabling logic operations and selecting the XOR write mode
- XOR is especially useful for swapping blocks of memory such as menus that are stored off screen

If S represents screen and M represents a menu the sequence:

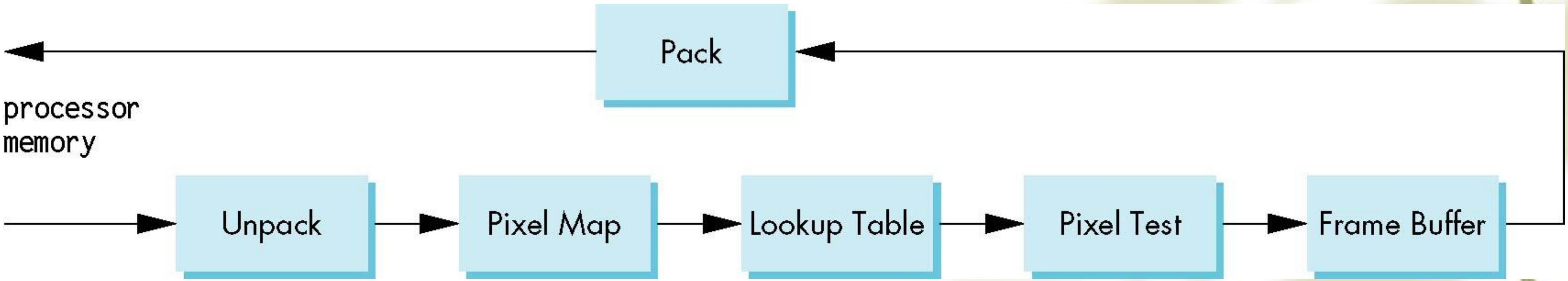
$$M \leftarrow S \oplus M$$
$$S \leftarrow S \oplus M$$

swaps the S and M

drag and drop

The Pixel Pipeline

- OpenGL has a separate pipeline for pixels
  - Writing pixels involves
    - Moving pixels from processor memory to the frame buffer
    - Format conversions
    - Mapping, Lookups, Tests
  - Reading pixels



OpenGL Pixel Functions

`glReadPixels(x,y,width,height,format,type,myimage)`

start pixel in frame buffer

size

type of image

type of pixels

pointer to processor memory

```
GLubyte myimage[512][512][3];
glReadPixels(0,0, 512, 512, GL_RGB,
        GL_UNSIGNED_BYTE, myimage);
```

Deprecated Functionality

- ~~glDrawPixels~~
- ~~glCopyPixels~~
- ~~glBitMap~~

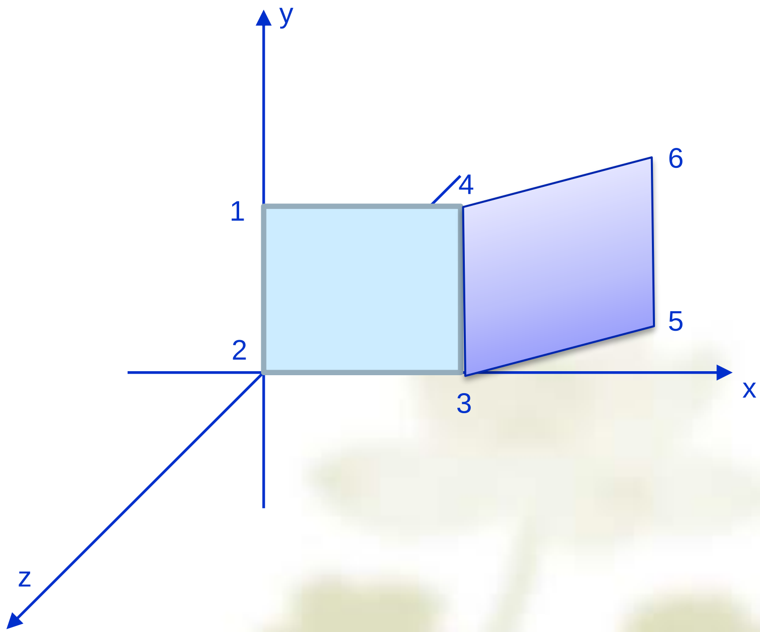
- Replace by use of texture functionality, glBltFrameBuffer, frame buffer objects
- GPUs now include a large amount of texture memory that we can write into
- Advantage: fast (not under control of window system)

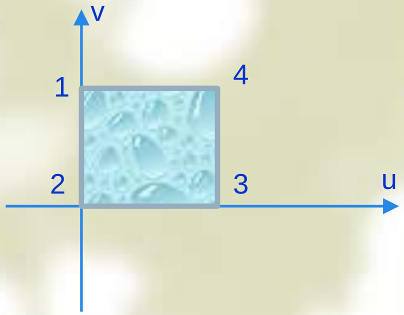
OBJ File Format

- **v x y z** #vertex
- **vn x y z** #normal vector. Not yet associated to any vertex.
- **vt u v [w]** #texture. u and v are the x and y coordinates in [0,1] of the texture map.
- **f v1[/vt1][/vn1] v2[/vt2][/vn2] v3[/vt3][/vn3] …** # v_i is vertex index number. vn_i is normal index number. vt_i is texture index number.

OBJ Example

```
v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
v 4.000000 0.000000 -1.255298
v 4.000000 2.000000 -1.255298
vn 0.000000 0.000000 1.000000
vn 0.000000 0.000000 1.000000
vn 0.276597 0.000000 0.960986
vn 0.276597 0.000000 0.960986
vn 0.531611 0.000000 0.846988
vn 0.531611 0.000000 0.846988
vt 0.000000 1.000000
vt 0.000000 0.000000
vt 1.000000 0.000000
vt 1.000000 1.000000
# 6 vertices
# 6 normal
# 4 texture points

f 1/1/1 2/2/2 3/3/3 4/4/4
f 4/1/4 3/2/3 5/3/5 6/4/6
# 2 elements
```



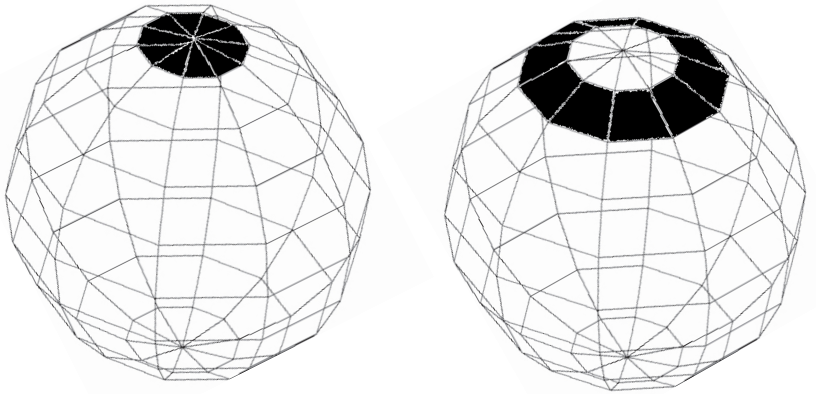


3. Mapping Methods

Three Types of Mapping:

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

Texture Mapping



geometric model

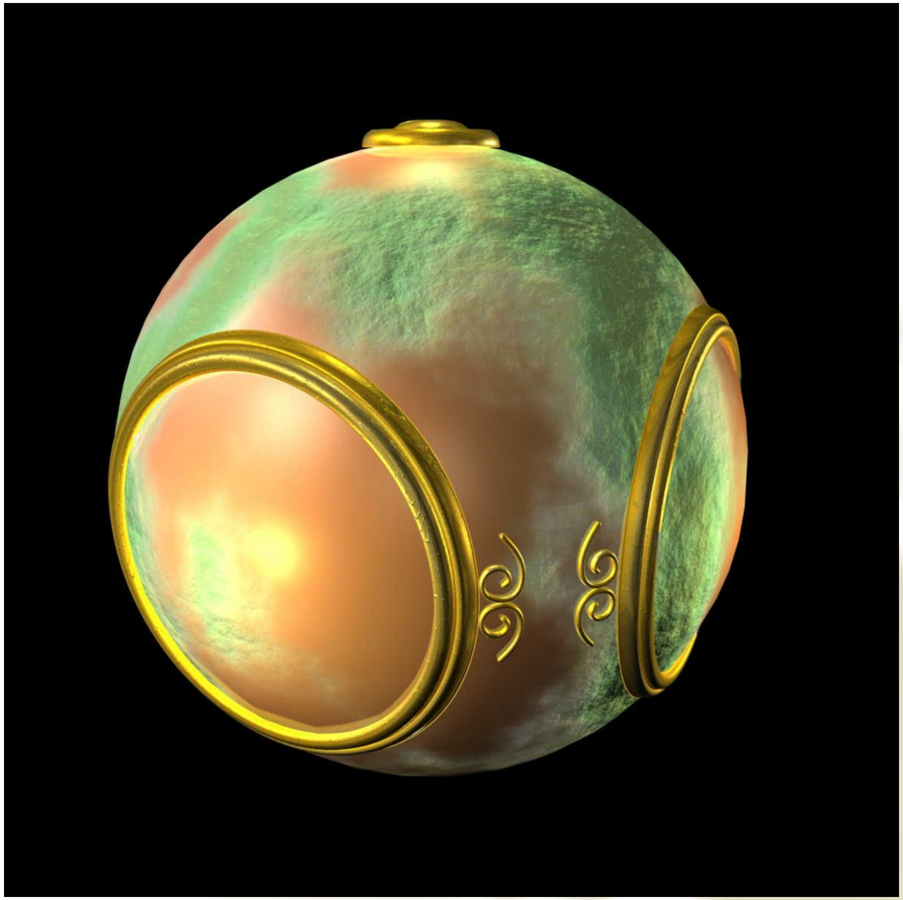
texture mapped

Environment Mapping

- highly specular surfaces

Bump Mapping

- altering normal vectors
- to process each fragment independently with a fragment shader



Bump Mapping

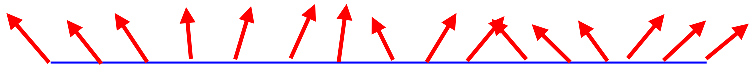
Ordinary texture mapping
(No bump mapping)

With bump mapping

Shading of Bump Mapping

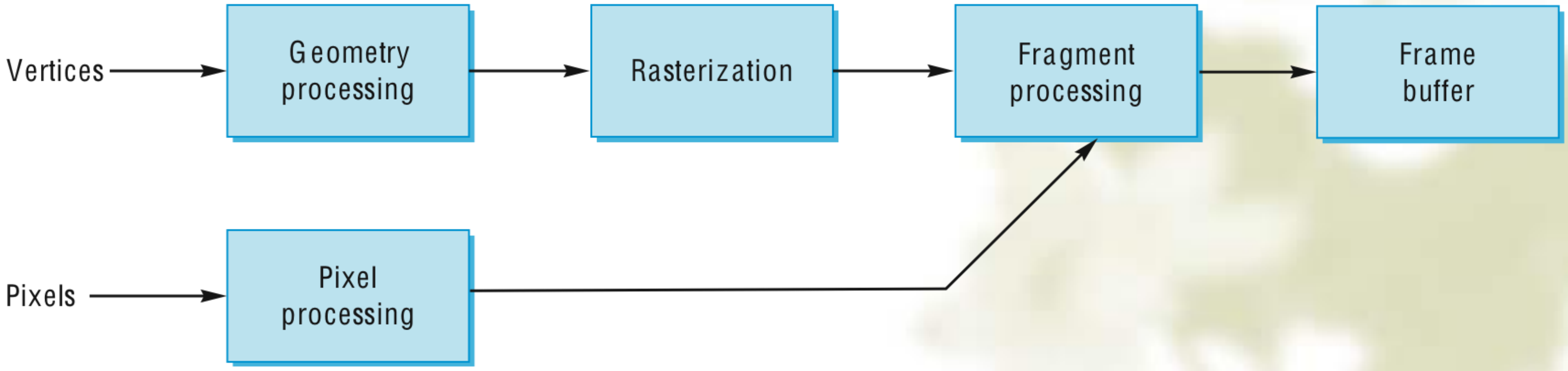
- In Gouraud shading, light intensity is computed at vertices. The intensities of other pixels on the line are blended from the intensities of the two vertices

- In bump mapping, light intensity is computed at every pixel. The normals of the original surface are used. (Note that the curved is still approximated by the line.)



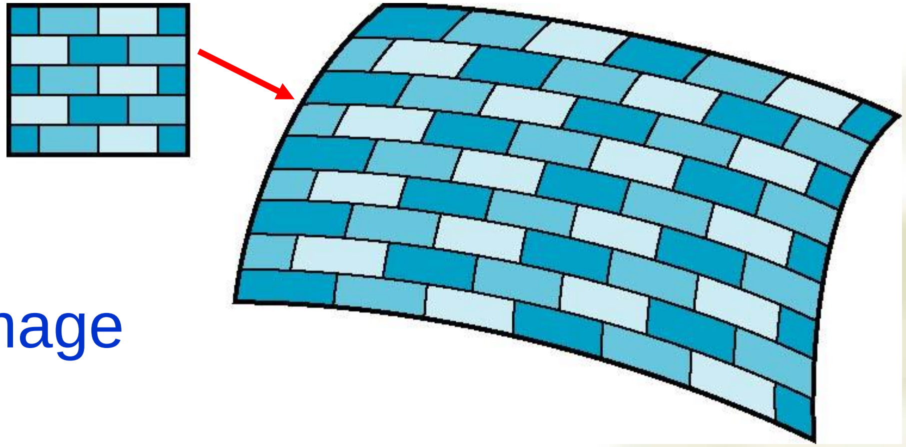
Where does mapping take place?

- Mapping techniques are implemented at the end of the rendering pipeline
  - Very efficient because few polygons make it past the clipper



Is it simple?

- Although the idea is simple---map an image to a surface---there are 3 or 4 coordinate systems involved



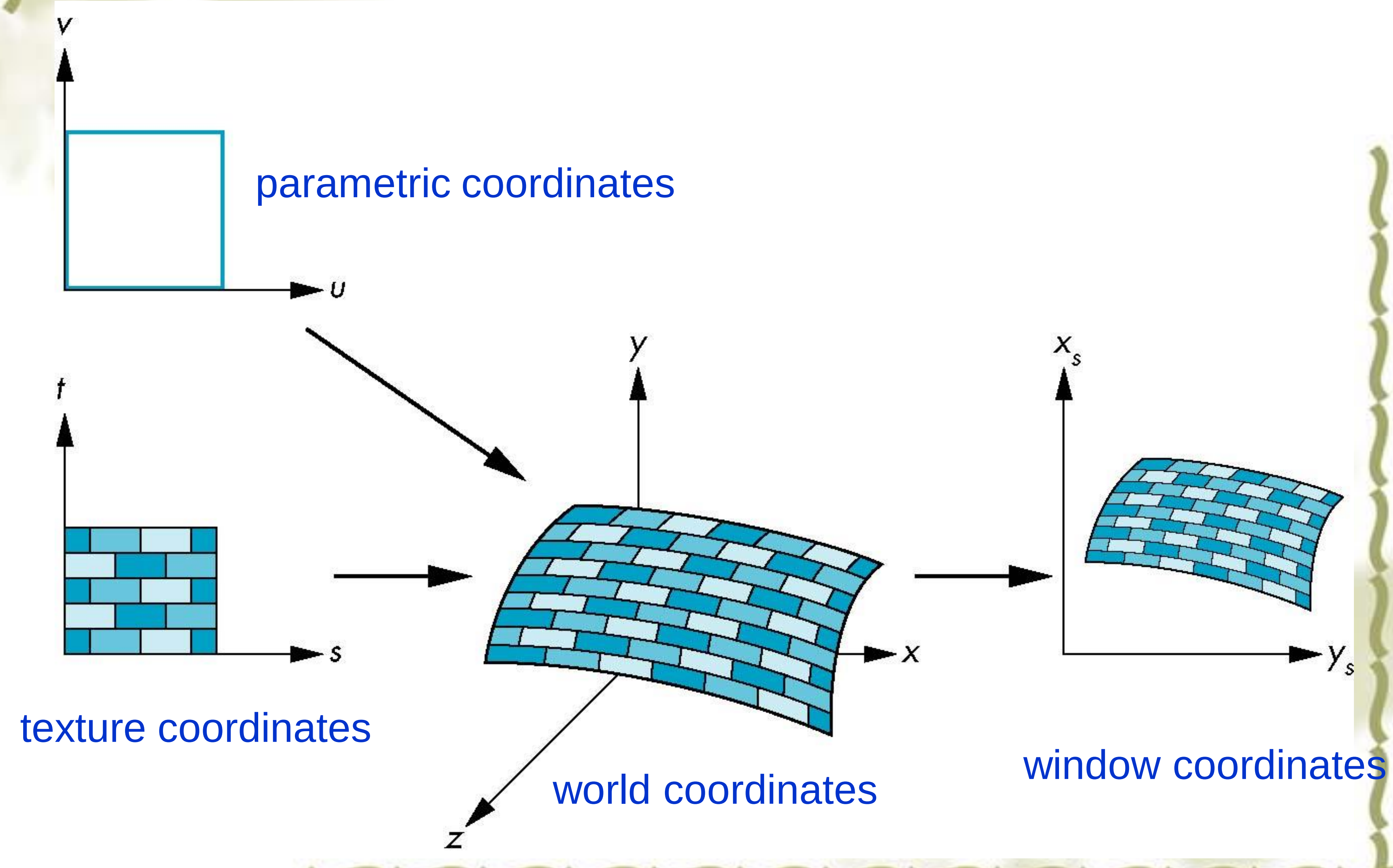
2D image

3D surface

Coordinate Systems

- Parametric coordinates
  - May be used to model curves and surfaces
- Texture coordinates
  - Used to identify points in the image to be mapped
- Object or World Coordinates
  - Conceptually, where the mapping takes place
- Window Coordinates(pixel)
  - Where the final image is really produced

Texture Mapping



Mapping Functions

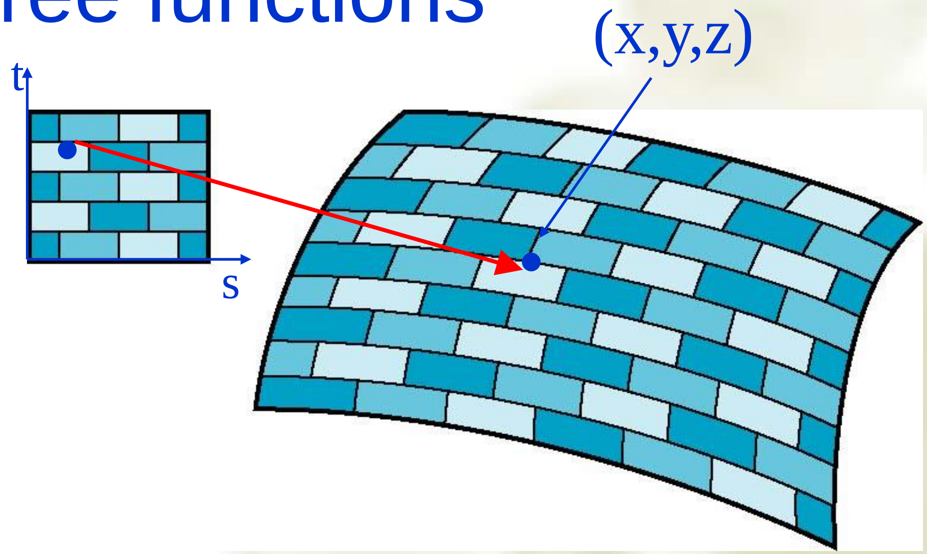
- Basic problem is how to find the maps
- Consider mapping from texture coordinates to a point a surface
- Appear to need three functions

  $x = x(s,t)$

  $y = y(s,t)$

  $z = z(s,t)$

- But we really want to go the other way

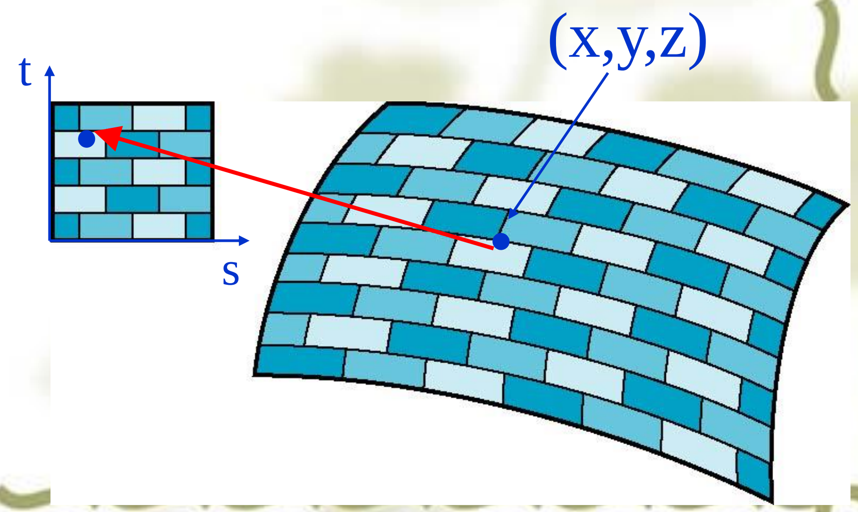


Backward Mapping

- We really want to go backwards
  - 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

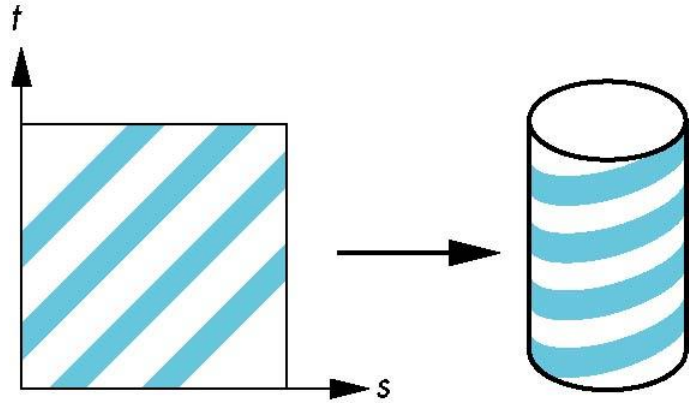
  s = s(x,y,z)

  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 intermediate surface
- Example: map to cylinder

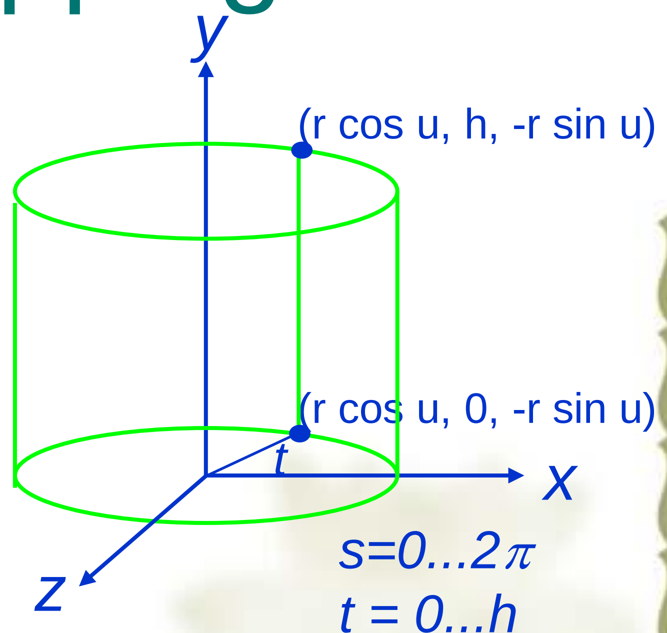


Cylindrical Mapping

parametric cylinder

$$\begin{cases} x = r\cos u \\ y = v/h \\ z = -r\sin u \end{cases}$$

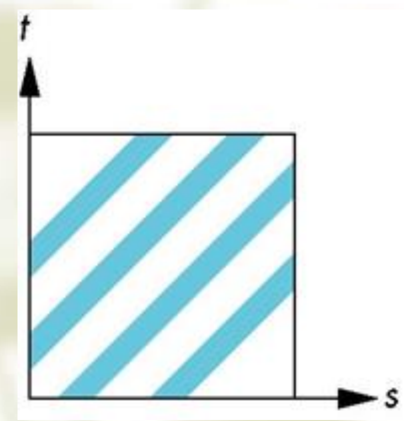
u=0…2π
v=0…h



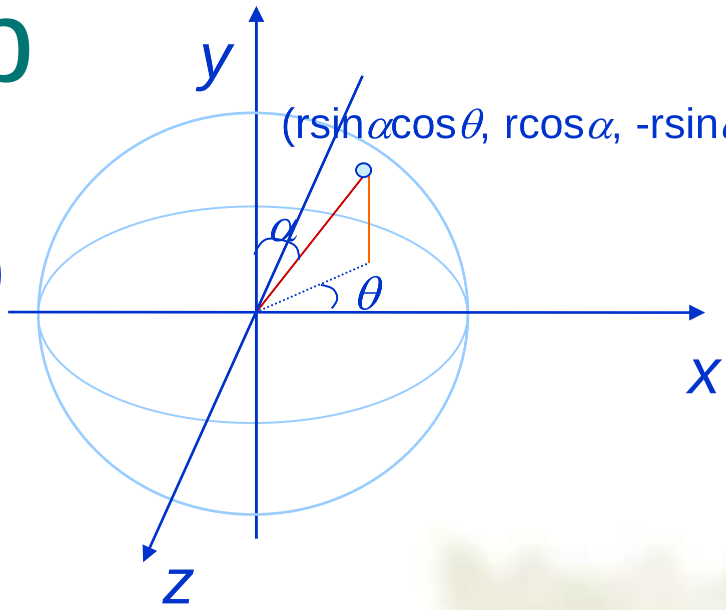
maps rectangle in u,v space to cylinder of radius r and height h in world coordinates

$$s = u$$
$$t = v$$

maps from texture space



Spherical Map

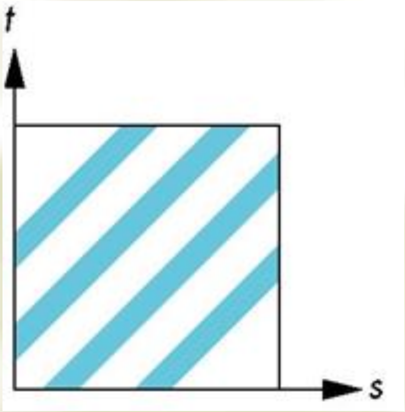


We can use a parametric(α,θ) sphere

$$\begin{cases} x = r \sin\alpha \cos\theta \\ y = r \cos\alpha \\ z = -r \sin\alpha \sin\theta \end{cases}$$

$\alpha = 0 \ldots \pi$
$\theta = 0 \ldots 2\pi$

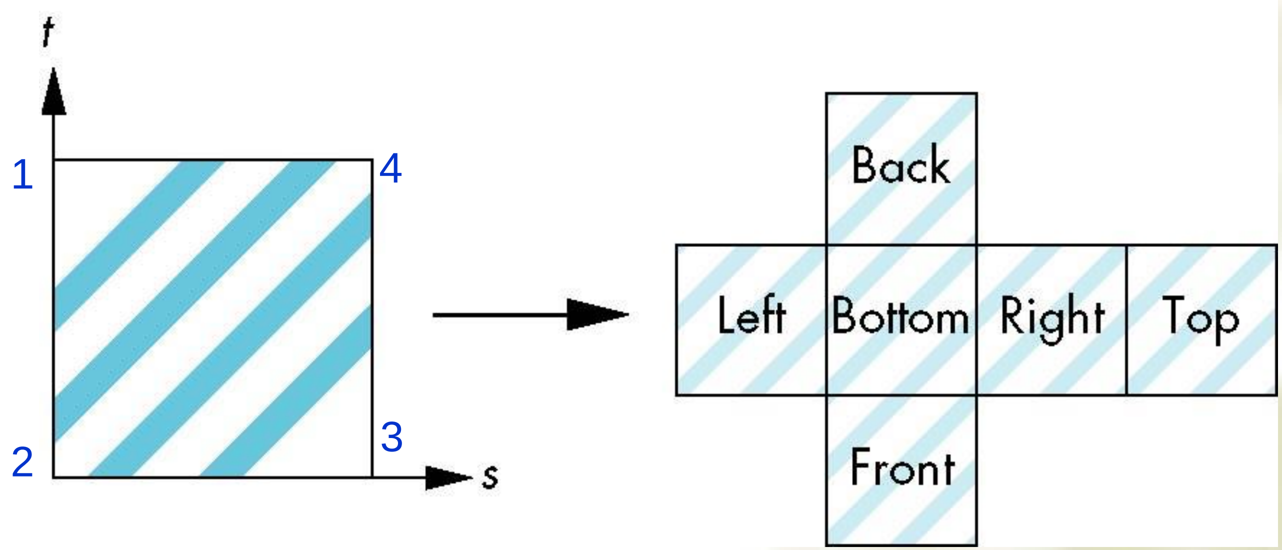
in a similar manner to the cylinder but have to decide where to put the distortion(Mercator Projection)



Spheres are used in environmental maps

Box Mapping

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



4. Texture Mapping in OpenGL

Three steps to applying a texture

a) specify the texture
   - read or generate image
   - assign to texture
   - enable texturing

b) assign texture coordinates to vertices
   - Proper mapping function is left to application

c) specify texture parameters
   - wrapping, filtering

a) Specify the Texture

- Define a texture image from an array of *texels* (texture elements) in CPU memory

```
Glubyte my_texels[512][512];
```

- Define as any other pixel map
  - Scanned image
  - Generate by application code
- Enable texture mapping
  - `glEnable(GL_TEXTURE_2D)`
  - OpenGL supports 1-3 dimensional texture maps

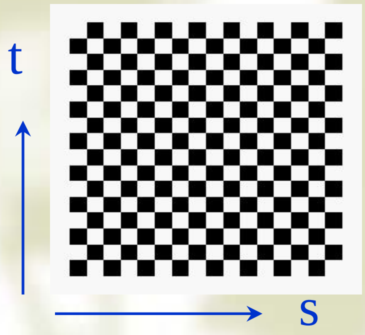
Creating Texture Maps (1)

- From an image: digital photo or scan
  - Penguins from a trip to South Africa

Checkerboard Texture(2)

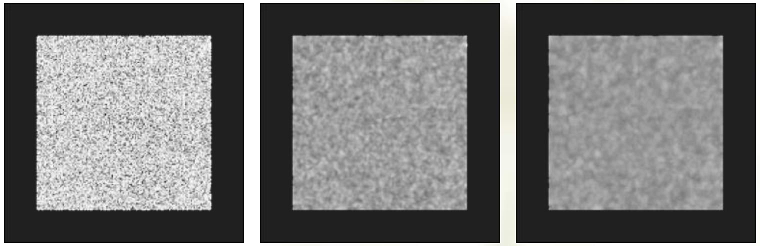
The texture is created by a program.

```
GLubyte image[64][64][3];

// Create a  64 x 64 checkerboard pattern
    for ( int i = 0; i < 64; i++ ) {
        for ( int j = 0; j < 64; j++ ) {
            GLubyte c = (((i & 0x8) == 0) ^ ((j & 0x8)  == 0)) * 255;
            image[i][j][0]  = c;
            image[i][j][1]  = c;
            image[i][j][2]  = c;
```

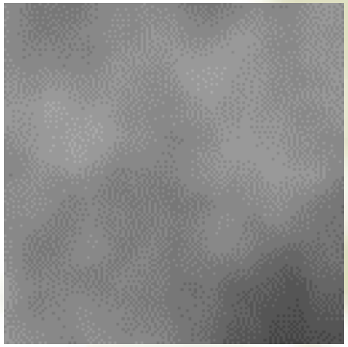


Noise as Texture(3)

- Noise is a procedural texture with only one component
- Filtered random noise



- 1/f noise



Format for Texture Files(4)

- The only question is reading the file and writing the contents to memory
- If you use *sophisticated files*, you may need to use tools to read the files
- Here we choose to use only raw files (the contents are rgbrgbrgb…) to keep things simple

Texture Mapping Needs…

- to develop the texture data to apply to each fragment
- to associate *pixels* in the screen space with *texels* in the texture space using many individual parameters, called a *binding* operation
- to associate each vertex with the texture so the vertex can get an appearance's value from the texture array

Define Image as a Texture

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

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

`level:` used for mipmapping (discussed later)

`components:` elements per texel

`w, h:` width and height of **`texels`** in pixels

`border:` used for smoothing (discussed later)

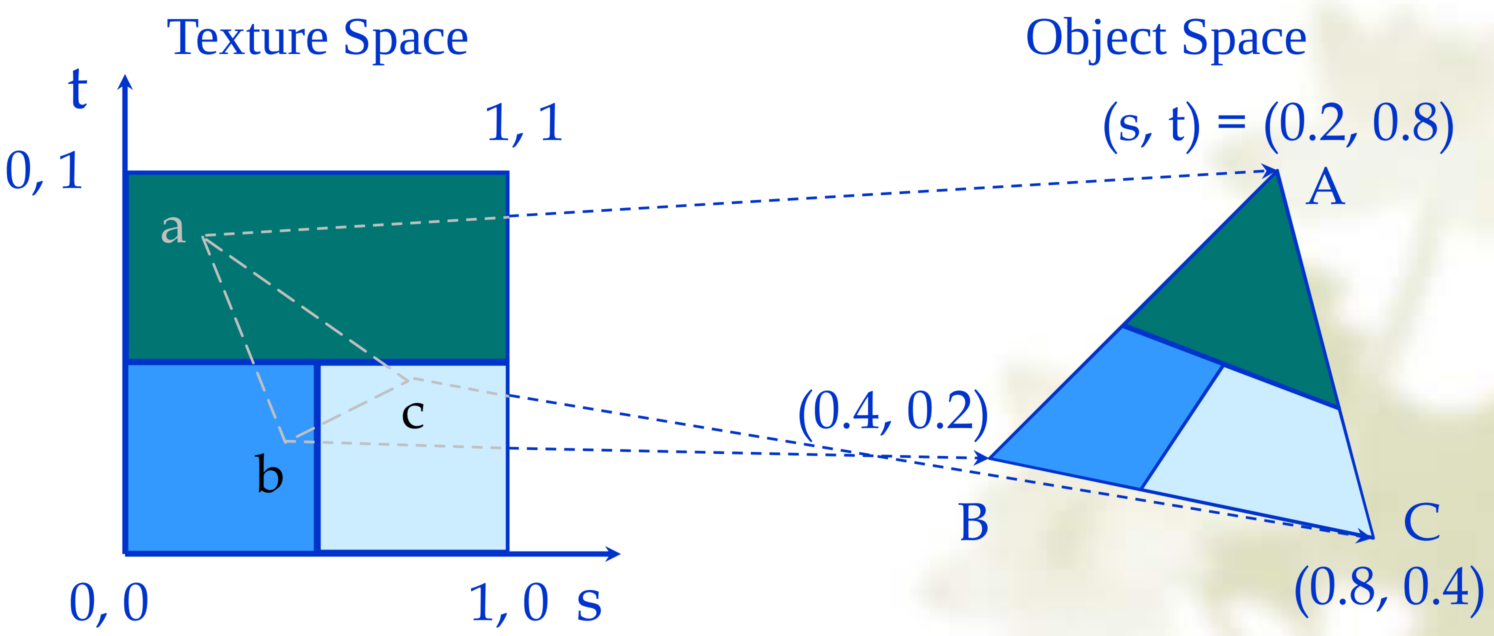
`format and type:` describe texels

`texels:` pointer to texel array

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0,
  GL_RGB, GL_UNSIGNED_BYTE, my_texels);
```

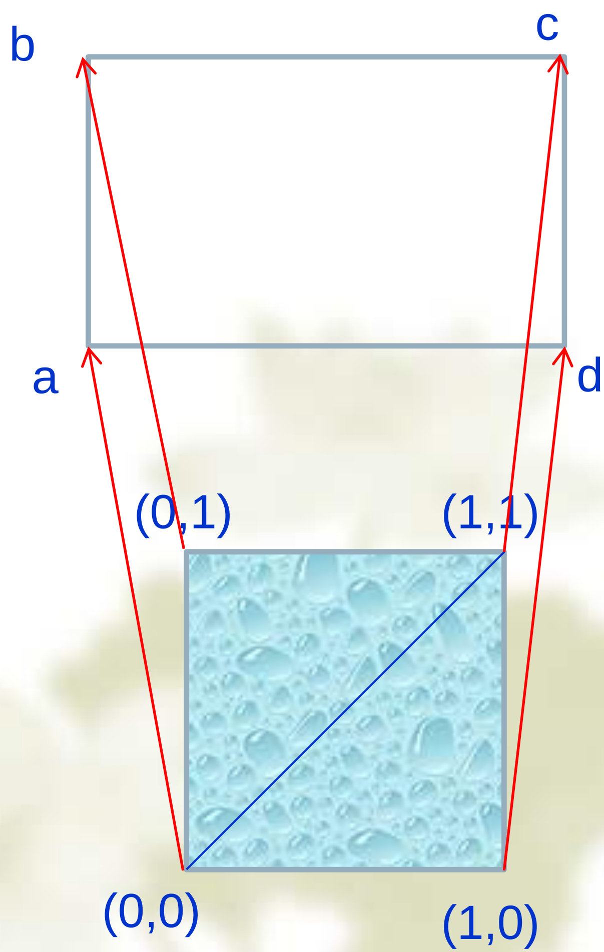
b) Mapping a Texture

- Based on parametric texture coordinates



Mapping Texture Coordinates

```
var texCoord = [
   vec2(0, 0),
   vec2(0, 1),
   vec2(1, 1),
   vec2(1, 0)
];

function quad(a, b, c, d) {
     pointsArray.push(vertices[a]);
     colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[0]);

     pointsArray.push(vertices[b]);
     colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[1]);
  // etc
}
```



Typical Code

```
offset = 0;
GLuint vPosition = glGetAttribLocation( program,
   "vPosition" );
glEnableVertexAttribArray( vPosition );
glVertexAttribPointer( vPosition, 4, GL_FLOAT,
   GL_FALSE, 0,BUFFER_OFFSET(offset) );

offset += sizeof(points);
GLuint vTexCoord = glGetAttribLocation( program,
   "vTexCoord" );
glEnableVertexAttribArray( vTexCoord );
glVertexAttribPointer( vTexCoord, 2,GL_FLOAT,
     GL_FALSE, 0, BUFFER_OFFSET(offset) );
```

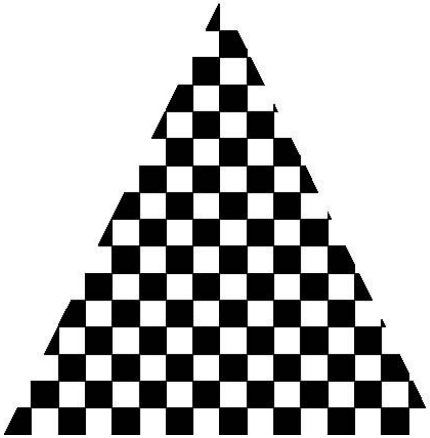
c) Parameters of Texture Mapping

- … because many things have to go right to make it work, including
  - Texture memory in the system must be loaded and its format (size and data) known
  - What is the meaning of the texture (color, …)
  - How will the texture value be combined with the object’s pixels
  - How will the texture value be computed when a pixel does not have an exact texture coordinate or the texture coordinates go out of the unit range

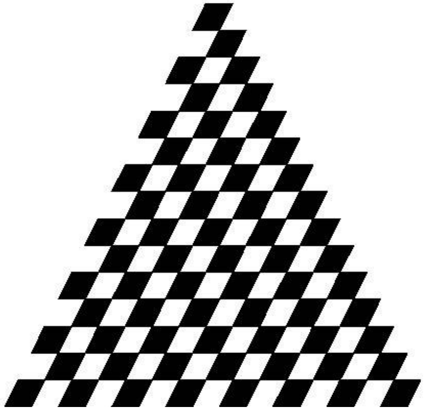
Interpolation

- OpenGL uses interpolation to find proper texels from specified texture coordinates
- Can be distortions

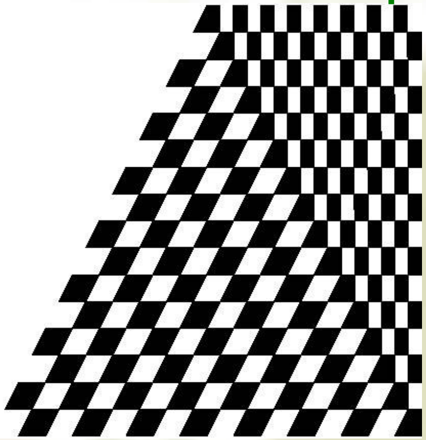
good selection
of tex coordinates



poor selection
of tex coordinates

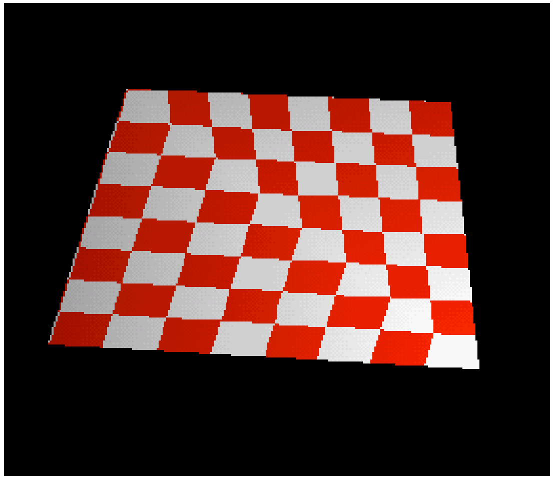


texture stretched
over trapezoid
showing effects of
bilinear interpolation

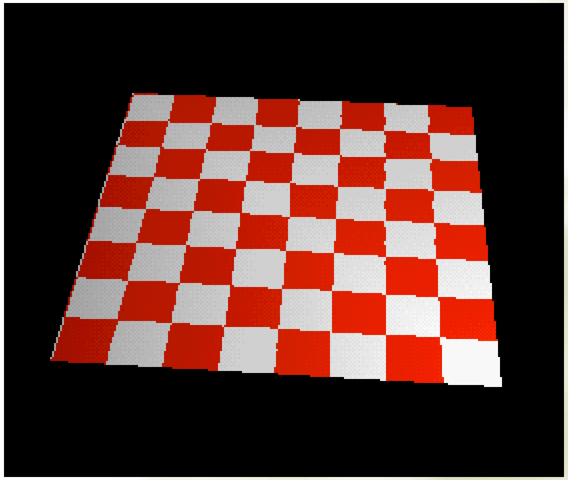


Texture Interpolation

```
glHint(GL_PERSPECTIVE_CORRECTION_HINT, hint);
//hint can be GL_DONT_CARE(default), GL_NICEST,
// GL_FASTEST(don't perform the perspective correction to maximize
   speed)
```



GL_DONT_CARE



GL_NICEST

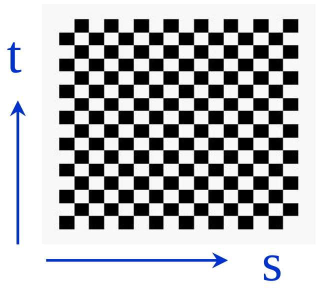
Texture Parameters

- OpenGL has a variety of parameters that determine how texture is applied
    1. Wrapping parameters determine what happens if s and t are outside the (0,1) range
    2. Filter modes allow us to use area averaging instead of point samples
    3. Mipmapping allows us to use textures at multiple resolutions
    4. Environment parameters determine how texture mapping interacts with shading

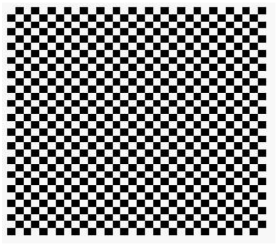
c.1) Wrapping Mode

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

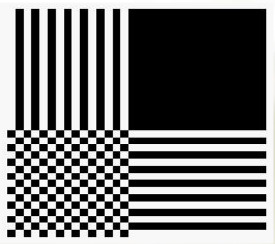
```
glTexParameteri( GL_TEXTURE_2D,
        GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE )
glTexParameteri( GL_TEXTURE_2D,
        GL_TEXTURE_WRAP_T, GL_REPEAT )
```



texture



GL_REPEAT
wrapping

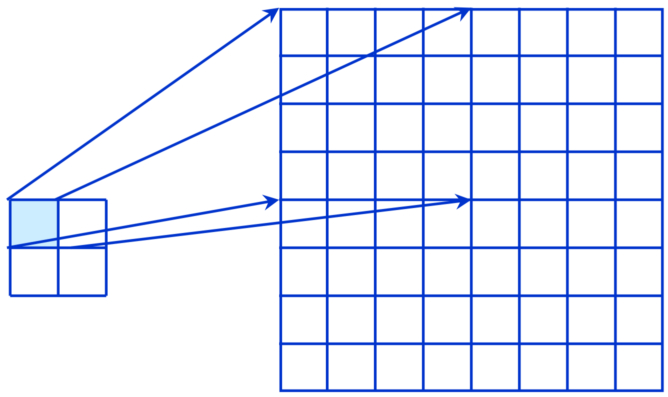


GL_CLAMP_TO_EDGE
wrapping

c.2) Magnification and Minification

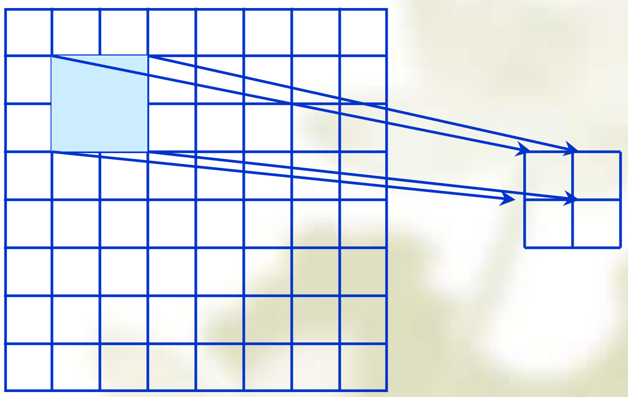
More than one texel can cover a pixel (*minification*) or more than one pixel can cover a texel (*magnification*)

Can use point sampling (nearest texel) or linear filtering (2 x 2 filter) to obtain texture values



Texture Polygon

Magnification



Texture Polygon

Minification

c.2) Filter Modes

Modes determined by

- `glTexParameteri( target, type, mode )`

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MAG_FILTER,
                GL_NEAREST);

glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MIN_FILTER,
                GL_LINEAR);
```

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

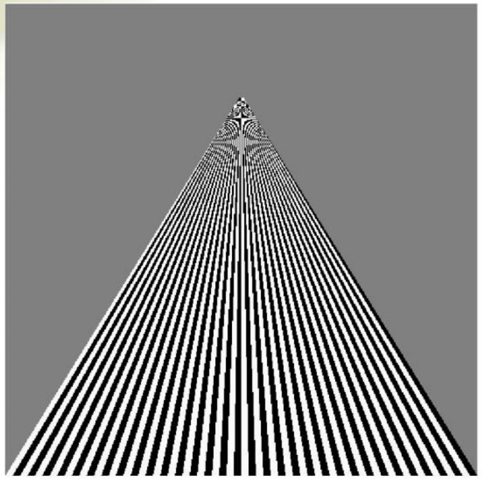
c.3) Mipmapped Textures

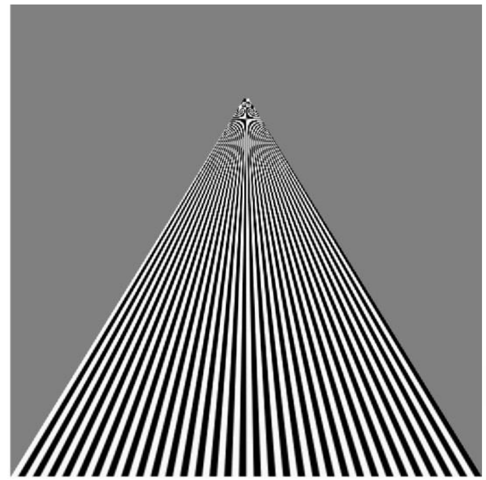
- *Mipmapping* allows for prefiltered texture maps with *different resolutions* – LOD(Level Of Detail)
- Lessens interpolation errors for smaller textured objects
- You can select the version that *best fits* your geometry, controlling the quality of the image with *less computation*
- Declare mipmap level during texture definition

`glTexImage2D( GL_TEXTURE_*D, level, … )`

Example -- Mipmap

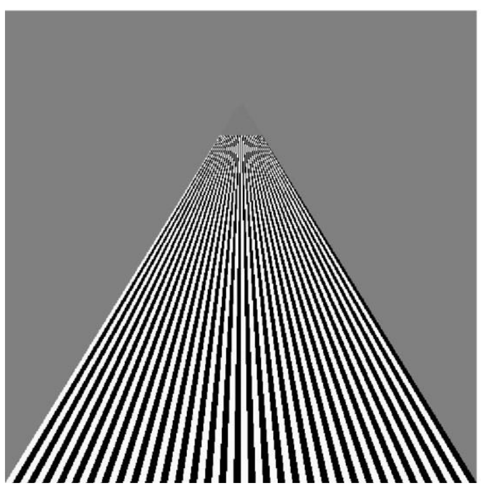
point sampling

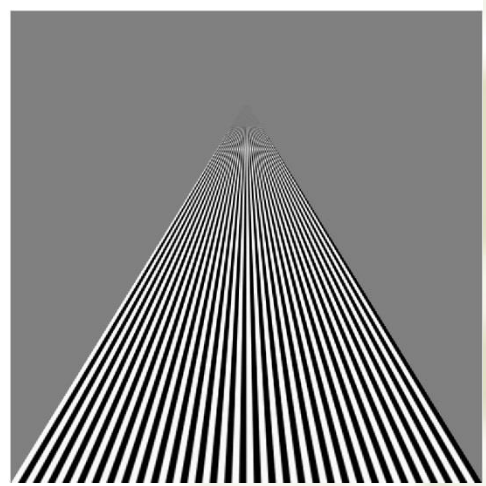




linear filtering

mipmapped point sampling





mipmapped linear filtering

5. Texture Objects in OpenGL

1. Create a *texture object* and load texel data into it
2. Supply texture coordinates for your vertices
3. Associate a *texture sampler* with each texture map you intend to use in your shader
4. Retrieve the texel values through the texture sampler from your shader

Texture Object(1)

Set the OpenGL system so that it is prepared to use texture mapping

A typical 2D texture setup with the texture already in the array `texImage`

```
// in init()
GLuint textures[1];
glEnable(GL_TEXTURE_2D);              //open texture computing
texName = gl.createTexture();          // create a texture name
glActiveTexture( GL_TEXTURE0 );
glBindTexture(GL_TEXTURE_2D,texName);  // Create texture objects with texture data and state
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);       //texture parameters
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
 //Define a texture image from an array of texels in CPU memory
glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,TEX_WIDTH,TEX_HEIGHT, 0, GL_RGB,
                GL_UNSIGNED_BYTE, texImage);

// in display()......//Associates a texture coordinate to a vertex of a graphic object
```

Texture Object(1)

Define how the texture is to be applied to a graphics object as it s rendered by the system

he texture already in the array texImage

```
glEnable(GL_TEXTURE_2D);                //open texture computing
glGenTextures(1, texName);             // generates a texture name
glActiveTexture( GL_TEXTURE0 );
glBindTexture(GL_TEXTURE_2D,texName[0]); // Create texture objects with texture data and state
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);       //texture parameters
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
 //Define a texture image from an array of texels in CPU memory
glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,TEX_WIDTH,TEX_HEIGHT, 0, GL_RGB,
             GL_UNSIGNED_BYTE, texImage);
```

// in display()......//Associates a texture coordinate to a vertex of a graphic object

Texture Object(1)

Identifies the data array that is to be used by the texture and how it is to be interpreted as the texture is loaded into texture memory

A t … xture already in the array texImage

```
                                     //open texture computing
gl…                                 // generates a texture name
glActiveText… _TEXTURE0 );
glBindTexture(GL_TEXTURE_2D,texName[0]); // Create texture objects with texture data and state
glTex…rameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);      //texture parameters
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
 //Define a texture image from an array of texels in CPU memory
glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,TEX_WIDTH,TEX_HEIGHT, 0, GL_RGB,
              GL_UNSIGNED_BYTE, texImage);
```

// in display()....../ /Associates a texture coordinate to a vertex of a graphic object

Linking with Shaders

```
GLuint vTexCoord = gl.getAttribLocation( program, "vTexCoord" );
glEnableVertexAttribArray( vTexCoord );
glVertexAttribPointer( vTexCoord, 2, GL_FLOAT, GL_FALSE, 0,
                       BUFFER_OFFSET(offset) );

//  Set the value of the fragment shader texture sampler variable
//   ("texture") to the appropriate texture unit. In this case,
//   zero, for GL_TEXTURE0 which was previously set by calling
//   glActiveTexture(GL_TEXTURE0).
//   active texture unit
glUniform1i( glGetUniformLocation(program, "texture"), 0 );
```

Vertex Shader

- Usually vertex shader will output texture coordinates to be rasterized

```
in vec4 vPosition; //vertex position in object coordinates
in vec4 vColor;  //vertex color from application
in vec2 vTexCoord; //texture coordinate from application

out vec4 color; //output color to be interpolated
out vec2 texCoord; //output tex coordinate to be interpolated

void main()
{   gl_Position=vPosition;
    color = vColor;
    texCoord = vTexCoord;
}
```

Applying Textures

- Textures are applied during fragments shading by a **sampler**
- Samplers return a texture color from a texture object

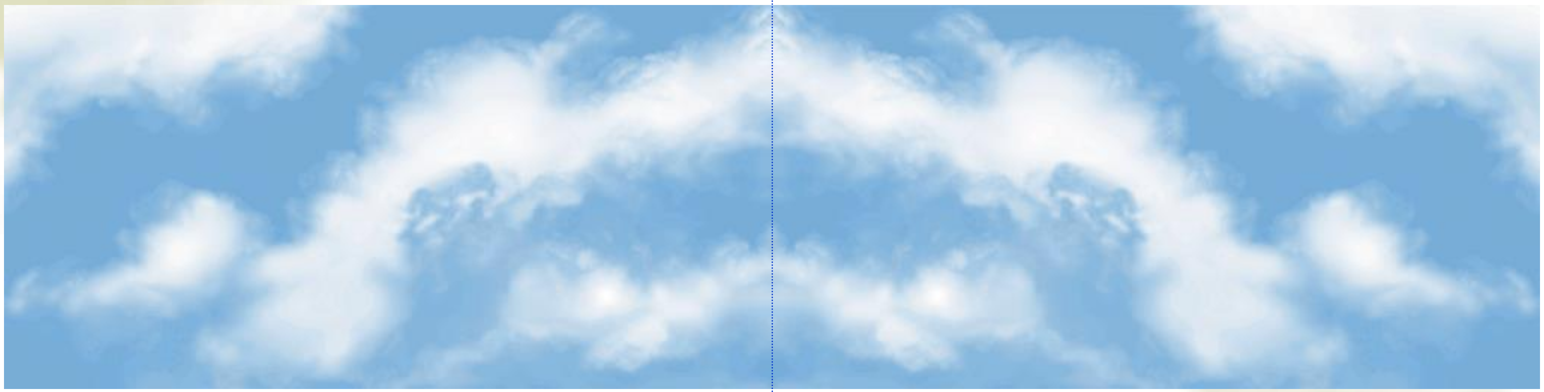
```
in vec4 color;        //color from rasterizer
in vec2 texCoord;   //texure coordinate from rasterizer
uniform sampler2D texture;  //texture object from application

void main()  {
   gl_FragColor = color * texture2D( texture, texCoord );
}
```

Other Texture Features

- Multitexturing
  - Apply a sequence of textures through cascaded texture units

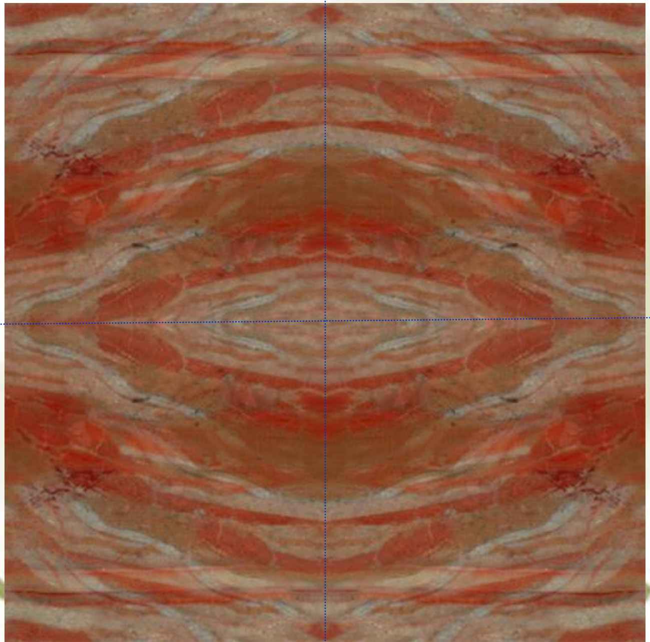

```
// in fragment shader
in vec2 tex_coord0;
in vec2 tex_coord1;
layout (location = 0) out vec4 color;
uniform sampler2D tex1;
uniform sampler2D tex2;
void main(void)
{
color = texture(tex1, tex_coord0) + texture(tex2, tex_coord1);
}
```

To avoid unwanted shape boundary appeared when an image is repeated horizontally, a quick fix is to provide an image such that its left half is a mirror image of the right half.

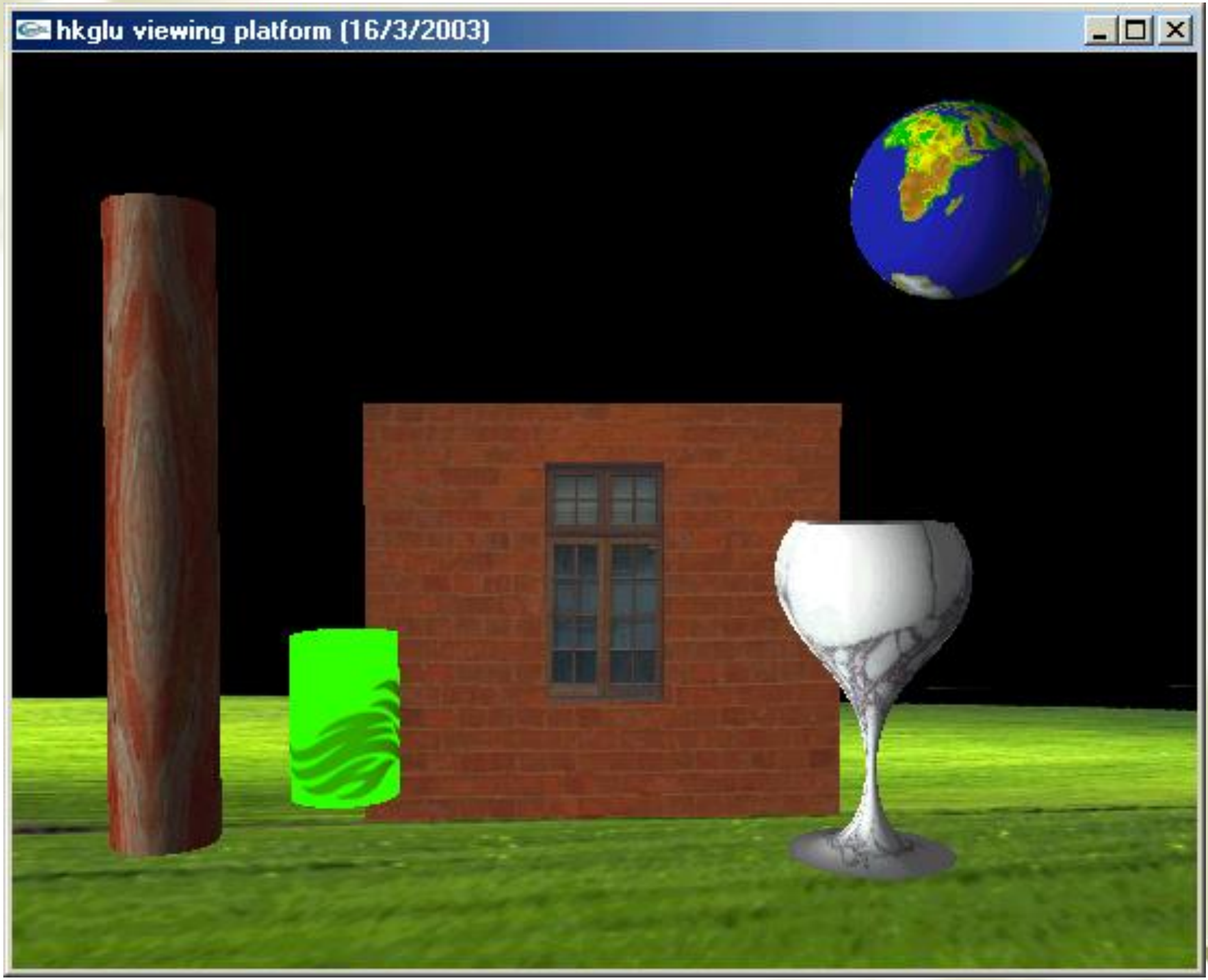


```
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_MIRRORED_REPEAT);
```

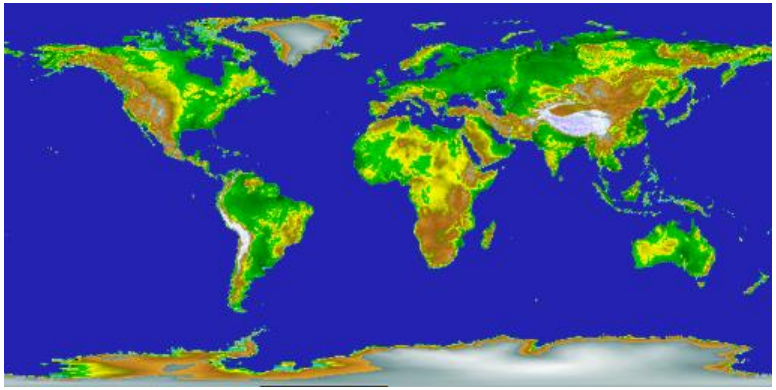
```
glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_WRAP_S,GL_MIRRORED_
REPEAT);
glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_WRAP_T,GL_MIRRORED_
REPEAT);
```



6. The Example of Texture



Texture in the Example



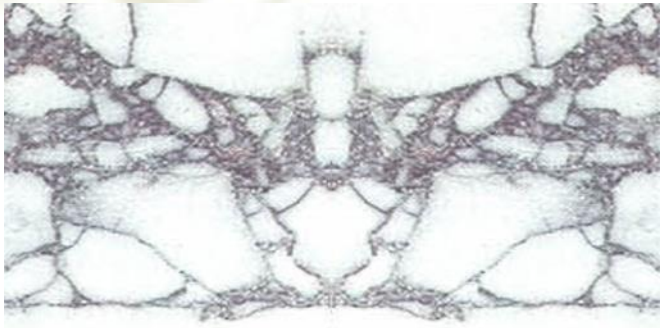
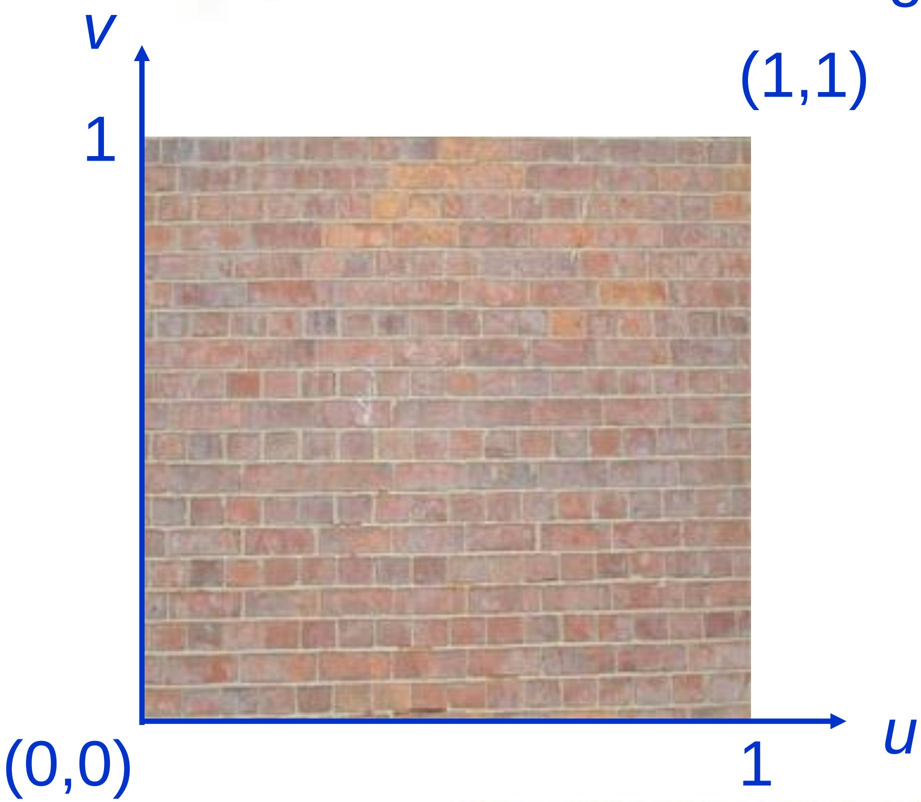
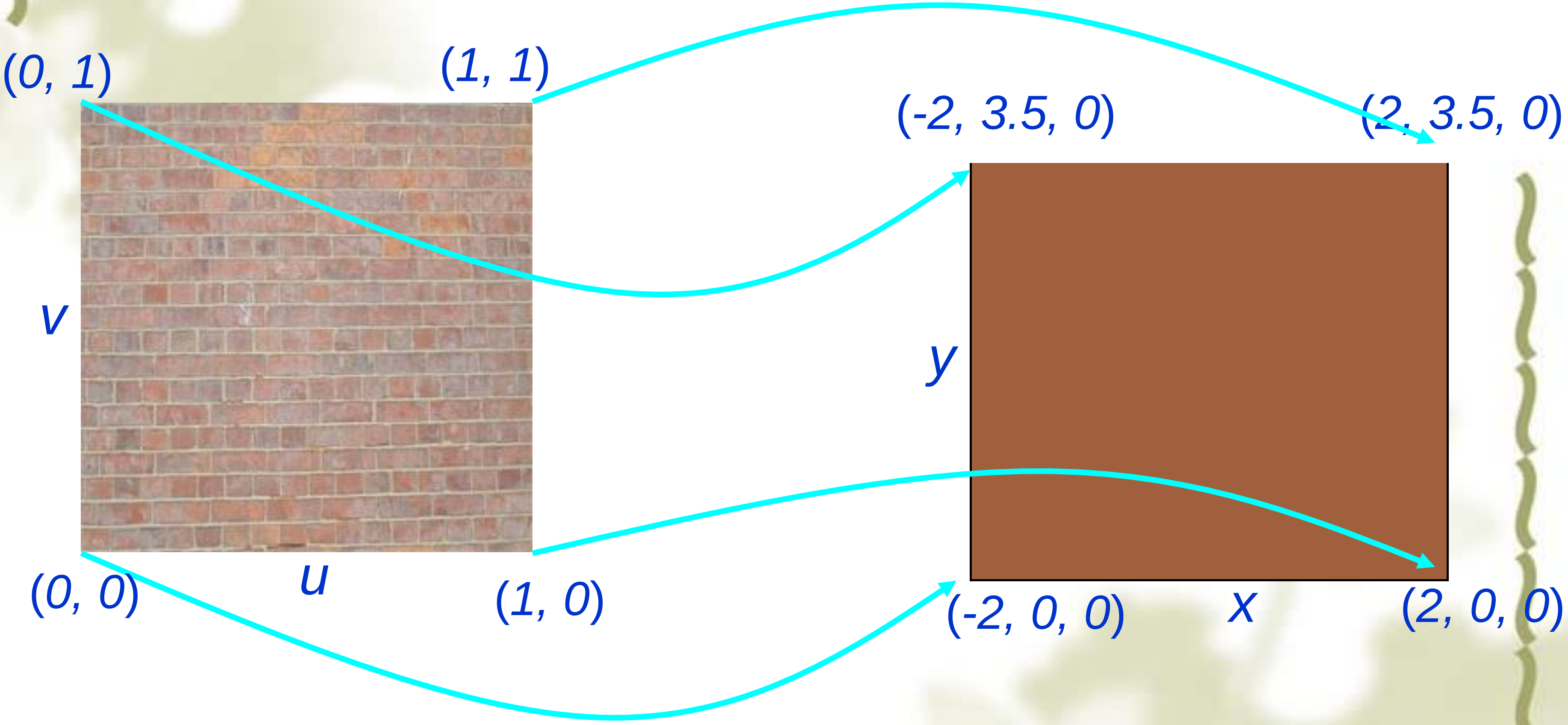


Image in the texture coordinates

- a 2-D texture image has a horizontal axis *u*, and a vertical axis *v*. The bottom left corner of the image is at the origin (0, 0), the top right corner is always at (1, 1). No matter what the size of a rectangle texture image are.



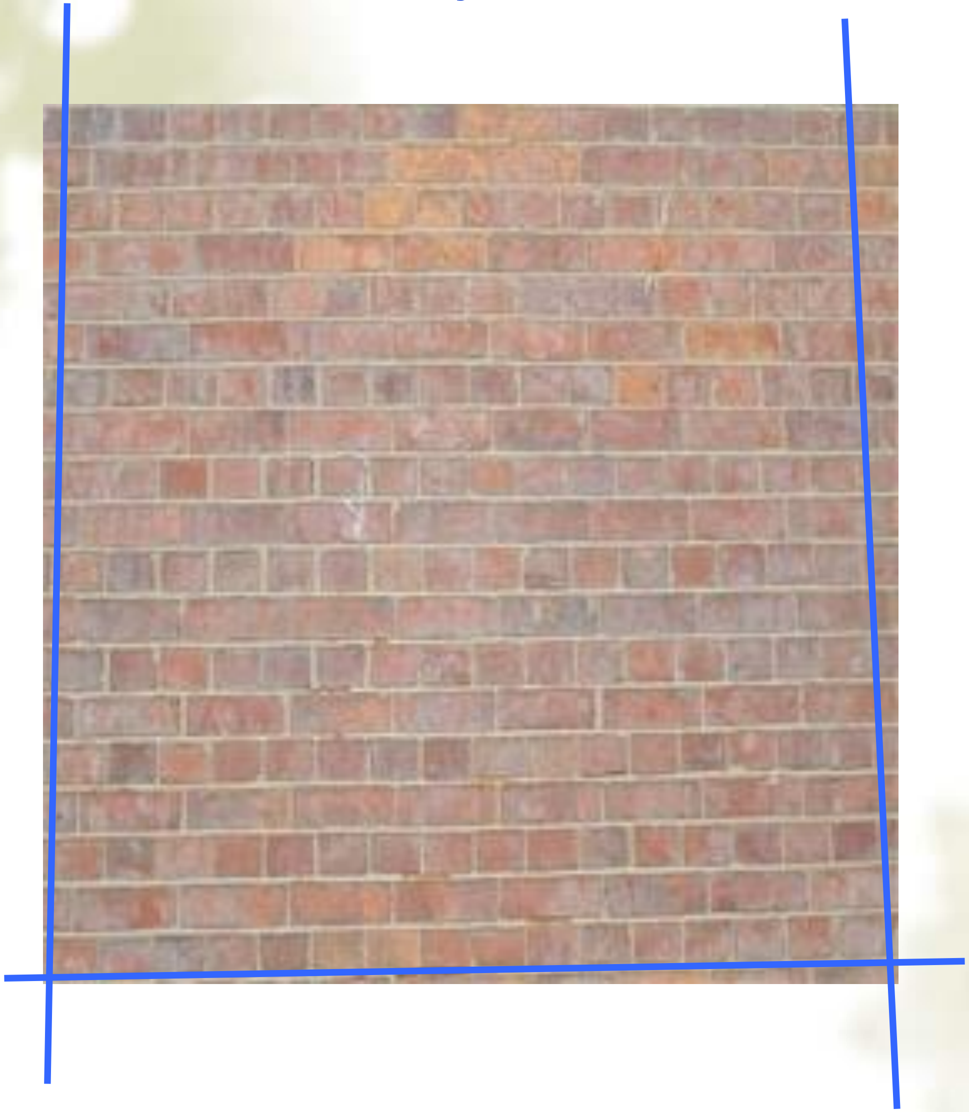
Texture Mapping



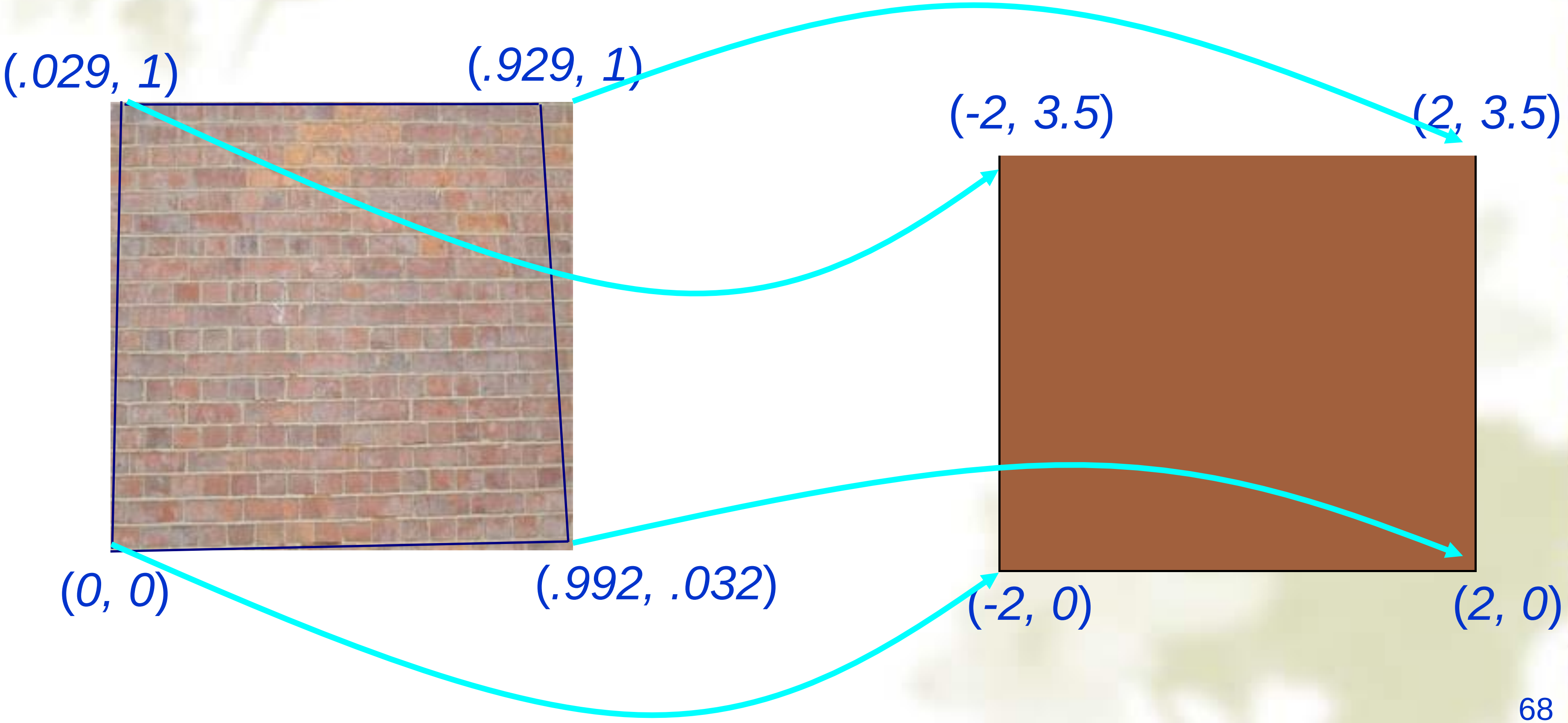
$0 \le u \le 1 \longrightarrow -2 \le x \le 2$

$0 \le v \le 1 \longrightarrow 0 \le y \le 3.5$

- It happens that the bricks in the image are not straightly horizontal and they are not vertical too

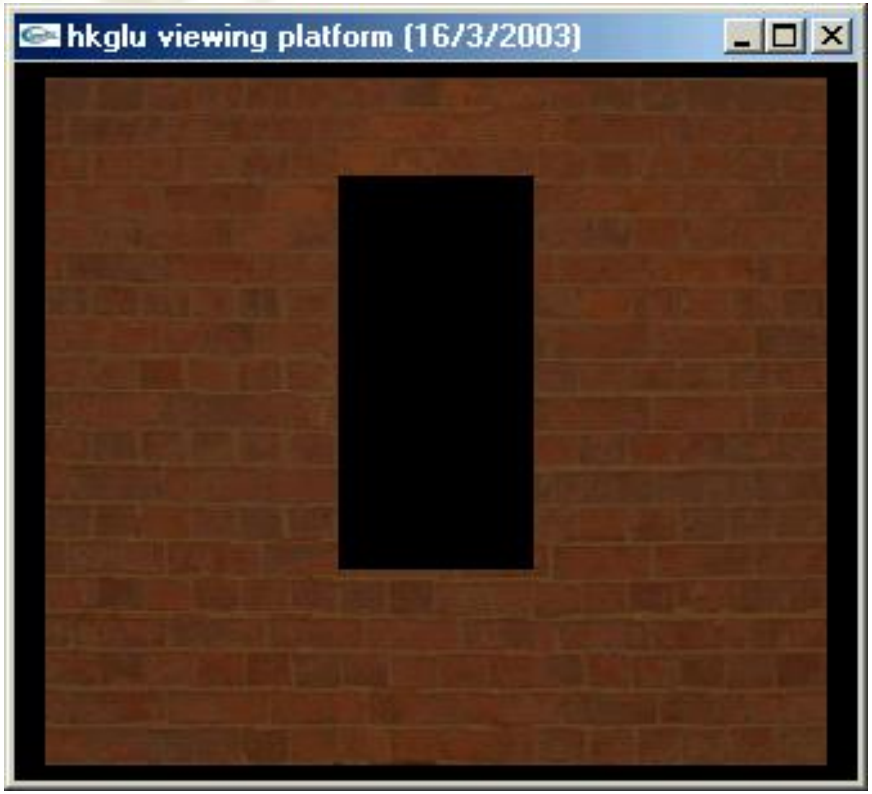


- Fix the vertices of partial texture to map to the vertices of the surface.

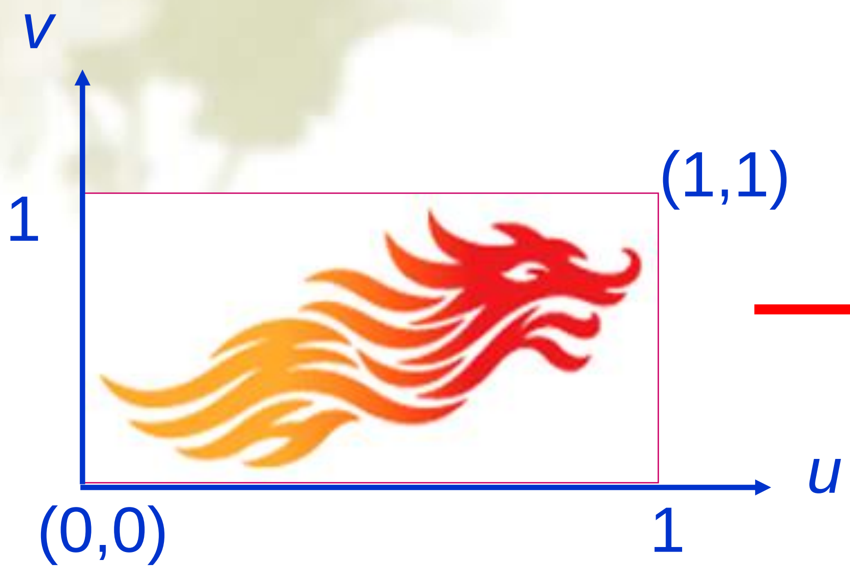


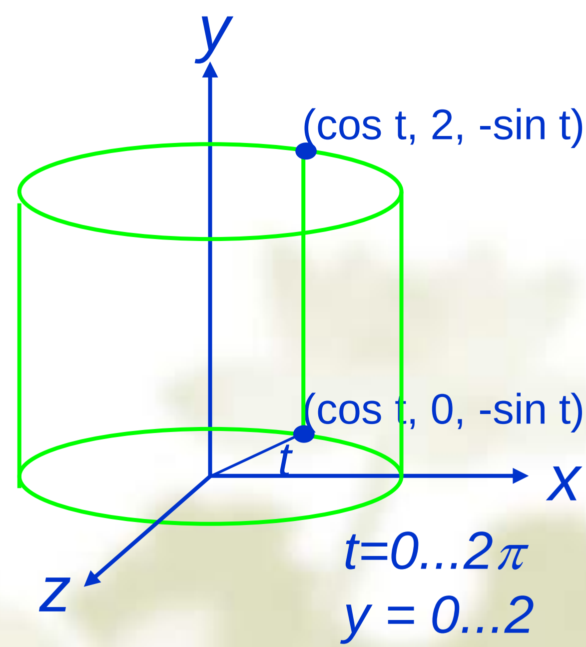
Due to rounding errors in depth testing, flaws may appear if two surfaces are drawn very close to each other.

OpenGL new version can read texture depth with GL_DEPTH_STENCIL



The mapping of an image on a cylinder is similar to wrap the cylinder with a paper

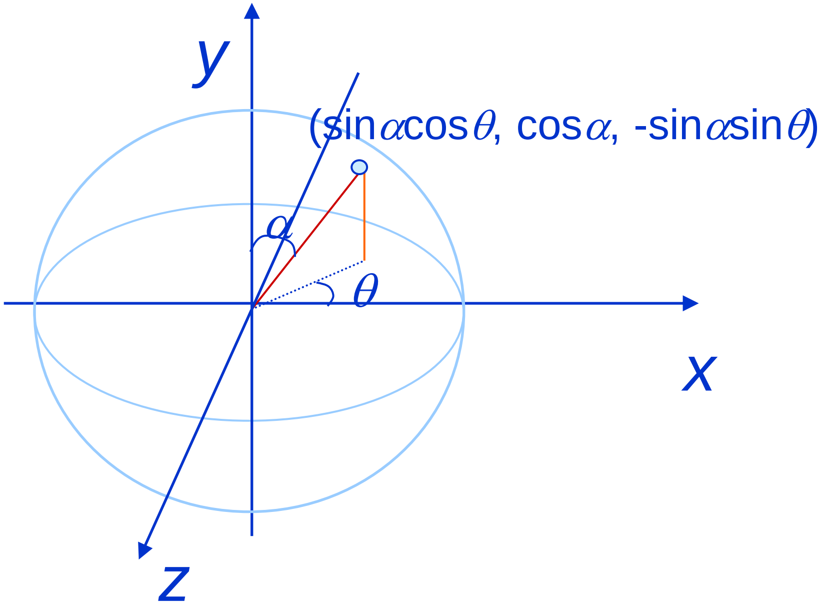




$0 \le u \le 1 \longrightarrow 0 \le t \le 2\pi$

$0 \le v \le 1 \longrightarrow 0 \le y \le 2$

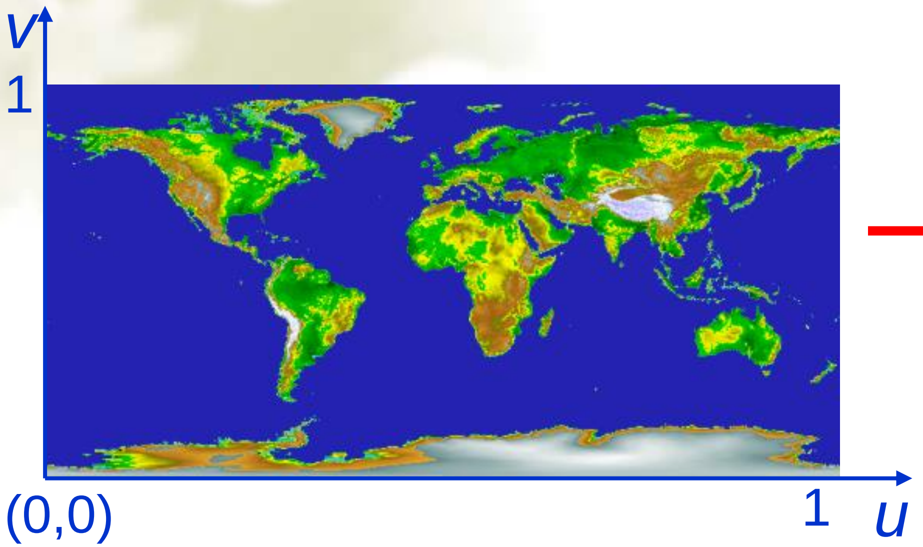
- $0 \le \alpha \le \pi$ is the angle between the *y* axis and the line that connect the origin to the point.

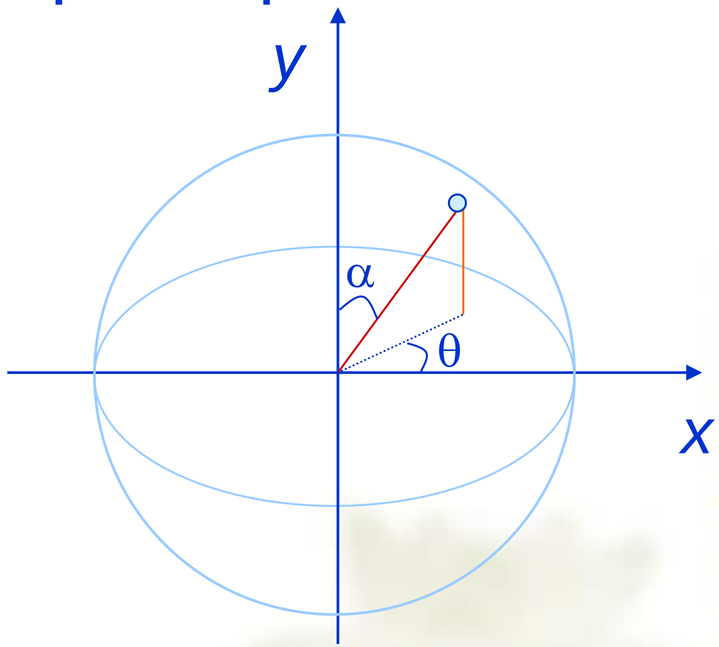


- $0 \le \theta \le 2\pi$ is the angle between the *x* axis and the projection of the line on *x-z* plane, measured in counterclockwise.

- For a unit sphere
  - The projection of the point on *y* axis is $\cos(\alpha)$
  - The projection of the point on *x* axis is $\sin(\alpha)\cos(\theta)$
  - The projection of the point on *z* axis is $-\sin(\alpha)\sin(\theta)$

The mapping of a cylindrical map of a planet onto a sphere is



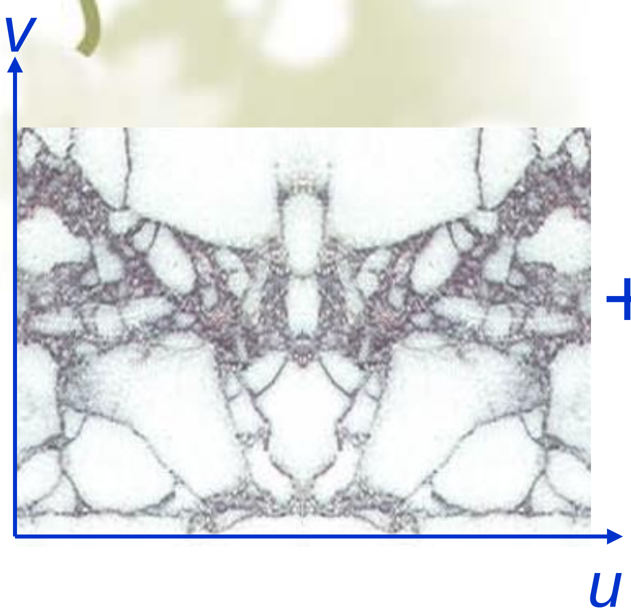


$0 \le u \le 1 \longrightarrow 0 \le \theta \le 2\pi$

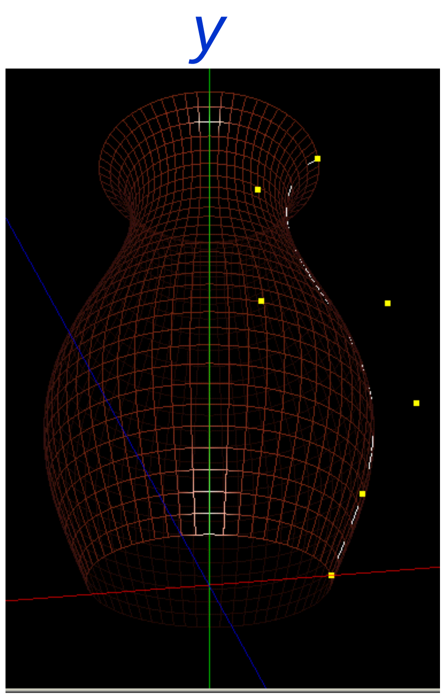
$1 \ge v \ge 0 \longrightarrow 0 \le \alpha \le \pi$

The texture coordinates for the point, $(\sin\alpha \cos\theta, \cos\alpha, -\sin\alpha \sin\theta)$, on the sphere surface are $(\theta/(2\pi), 1 - \alpha/\pi)$.

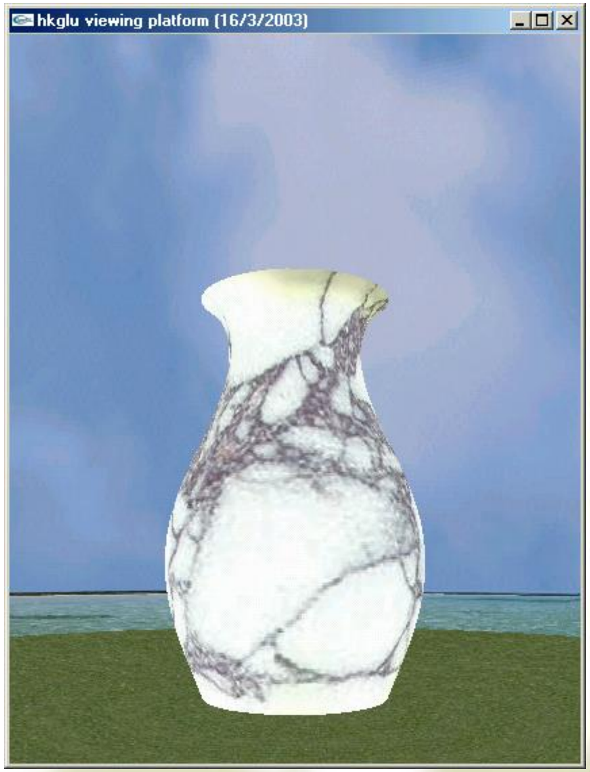
To map a texture onto a revolution surface



+



=



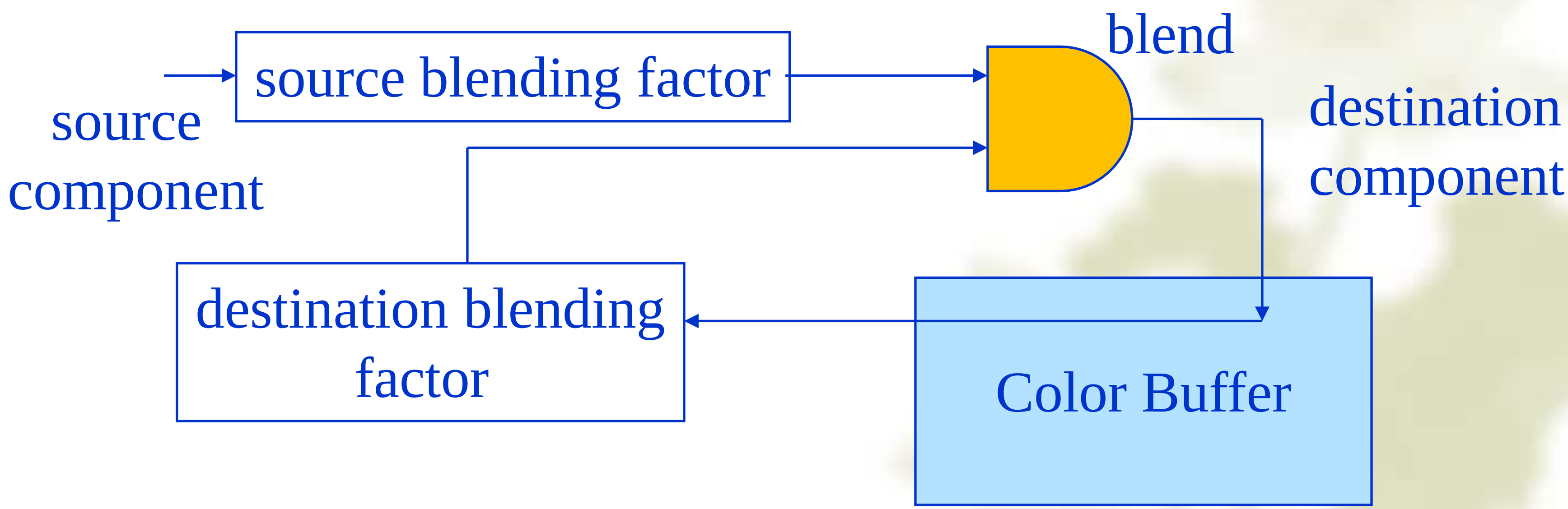
$0 \le u \le 1 \longrightarrow 0 \le \theta \le 2\pi$

$0 \le v \le 1 \longrightarrow 0 \le t \le 1$

The position of a point on a Bezier curve is determined by a parameter t. The point moves from the bottom to the top when t changes from 0 to 1.

7. Blend Model

- Use A component of RGBA (or RGBα) color to store opacity
- During rendering we can expand our writing model to use RGBA values



Color Blending and the Alpha Channel

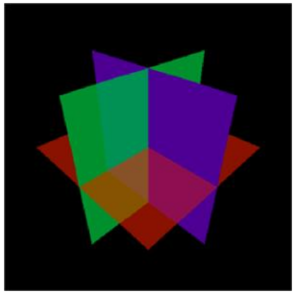
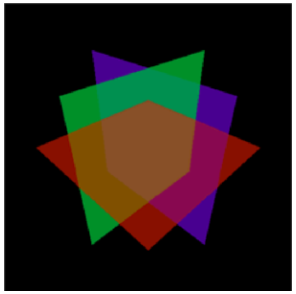
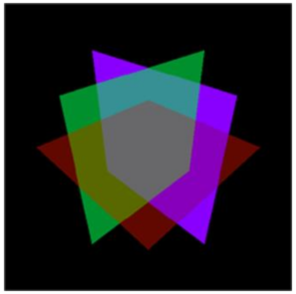
- The RGB color model is extended to the RGBA model by adding an *alpha* value
- This value represents the proportion of color this contributes when it is blended with other colors
- 1 = complete coverage; 0 = no coverage
- Blending models transparency but does not actually provide it

Creating Transparency with Blending Takes Some Work

(left) Three planes with alpha = .5, order B-G-R

(middle) Three planes as above, no depth testing

(right) Three planes as at top, varying alpha values

But if you divide each plane into four parts and draw them back to front, you get an accurate model of transparency

Draw partially transparent faces

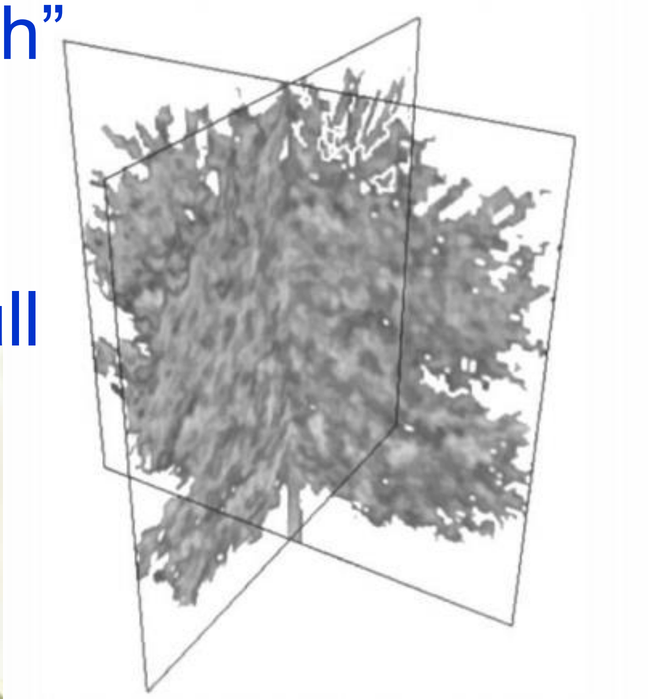
- glEnable(GL_DEPTH_TEST)
- *Draw all opacity*
- glEnable(GL_BLEND)
- glDepthMask(GL_FALSE) //depth only read
- glBlendFunc(GL_SRC_ALPHA, GL_ONE)
- *Draw all transparence from back to front*

//compare depth but not write to depth, blending with opacity

- glDepthMask(GL_TRUE)
- glDisable(GL_BLEND)

Billboards

- A billboard is a *plane object* (usually simple, like a rectangle) on which an image is texture mapped
- The image often includes *zero-alpha* areas so they can be "seen through"
- The object is rotated to *face the viewer* so that the viewer sees the image in 3D space, simulating a full 3D object



作业

- #7.2