Texture Mapping

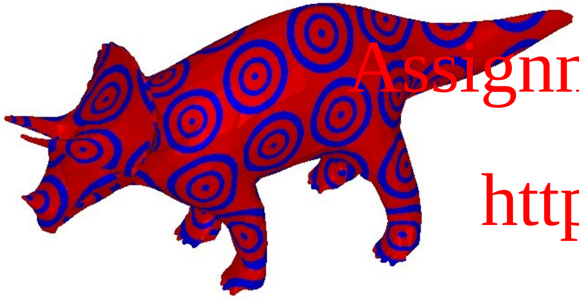
Frederick Li

This Lesson

- Introduction to texture mapping
- Mapping Methods
  - Forward and backward mapping
  - Two-part mapping
- WebGL Implementation

Texture Mapping

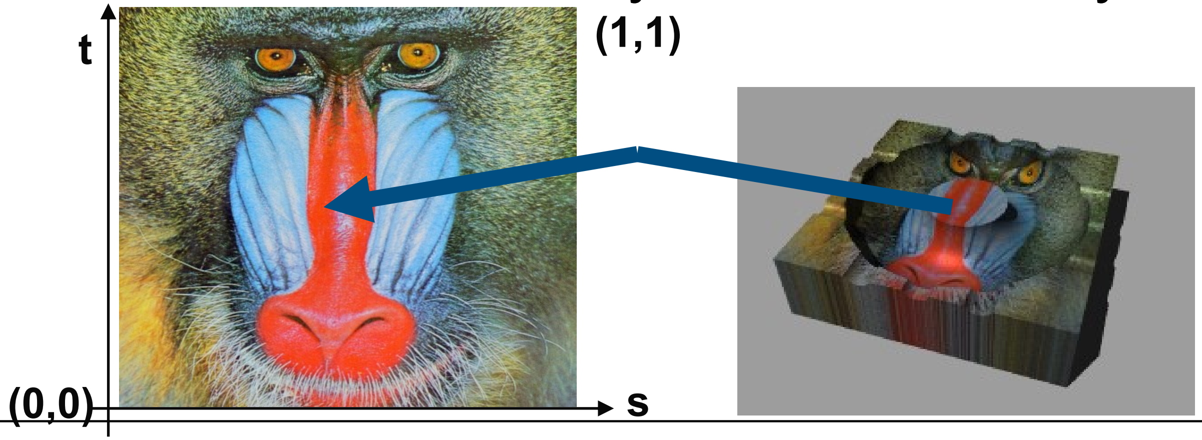
- A method for adding surface details, e.g. color and patterns, over the surface of a 3D model



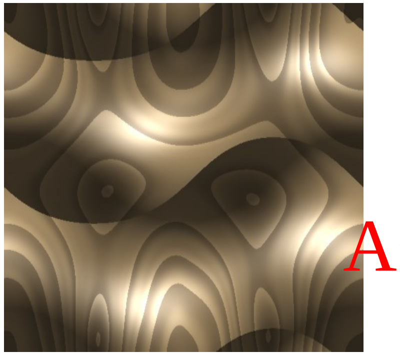
- Aims at increasing realism
  - Relying on mesh geometry to create such details is expensive
  - lighting/shading models are not enough
- Associate 2D information with 3D surface
  - Mapping process: point on surface corresponds to a point in texture, i.e. to “paint” image onto polygon

Texture and 3D Object

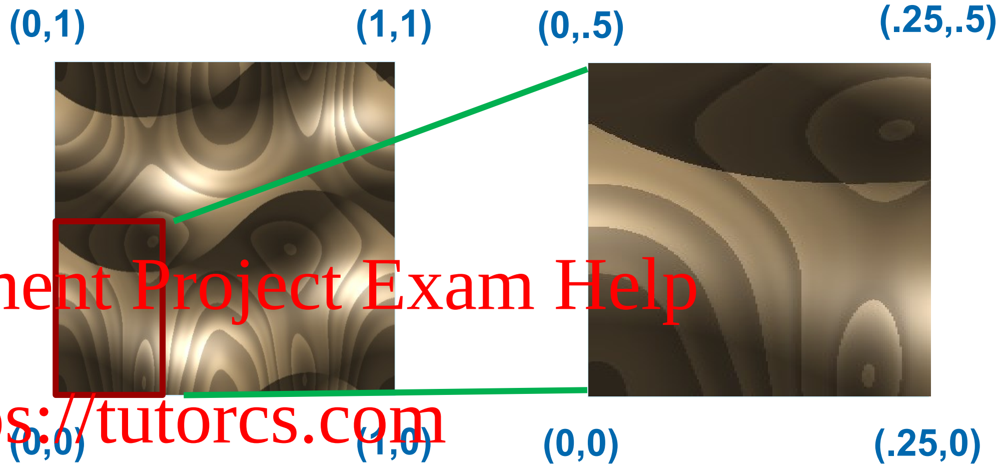
- Texture image: 2D array of color values (texels)
- Assigning texture coordinates (s,t) at vertex with object coordinates (x,y,z,w)
  - Use interpolated (s,t) for texel lookup at each pixel
  - Use the retrieved colour value to modify a polygon's color (or other surface properties)
  - Can be done manually or automatically



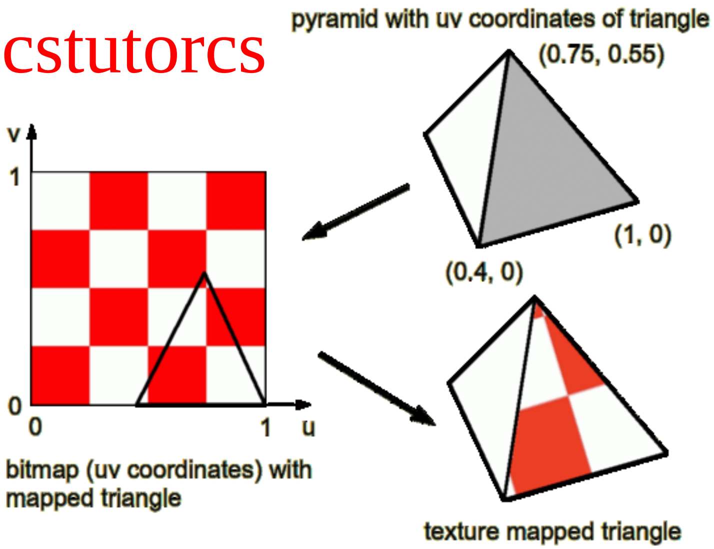
Fractional Texture Coordinates



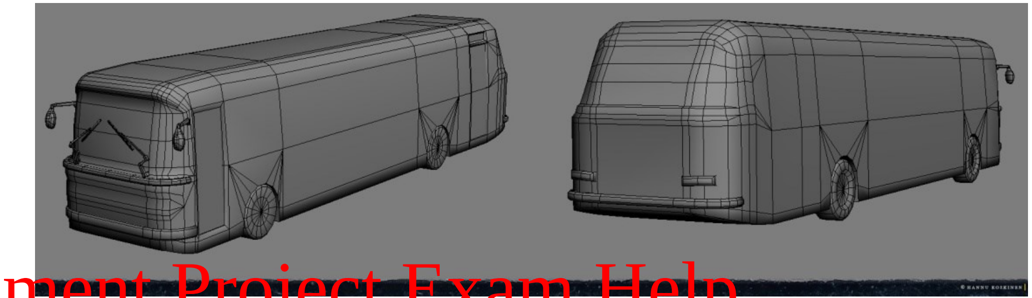
Texture image

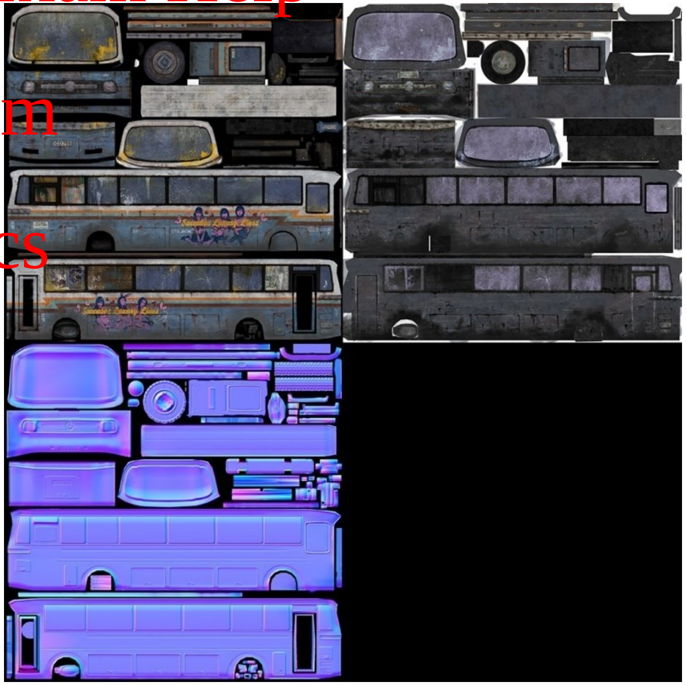


Only involve part of a texture map in the mapping process


pyramid with uv coordinates of triangle
(0.75, 0.55)
(1, 0)
(0.4, 0)
v
1
0
0
1
u
bitmap (uv coordinates) with mapped triangle
texture mapped triangle

Application: Texture atlas





A large image containing a collection, or "atlas", of sub-images, each of which is a texture for some part of an object.

Types of Texture Mapping Methods

Mapping: identify the correspondence between a texel (texture image element) and a screen pixel (fragment).

There are two types of mapping methods:

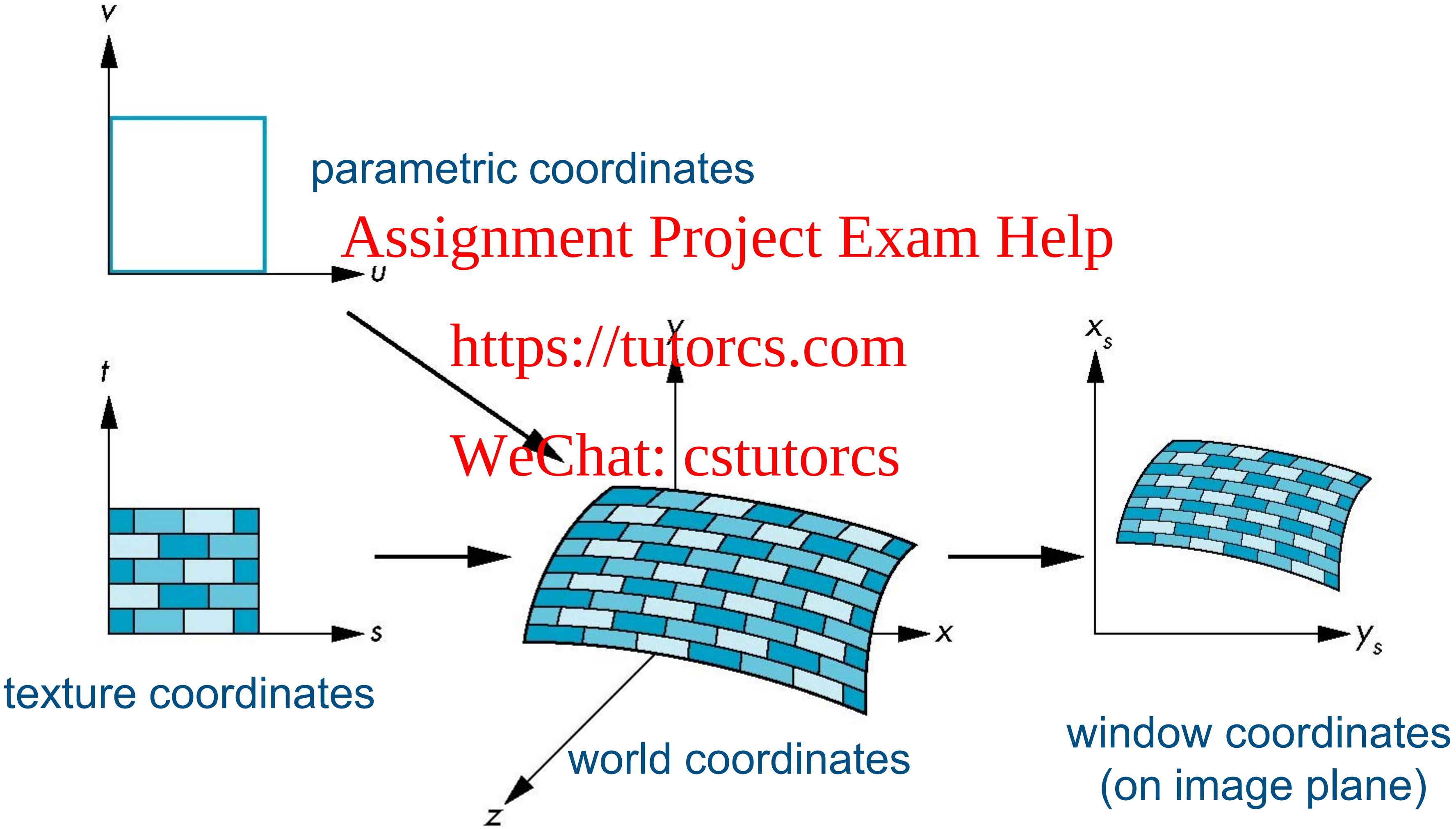
1. **Forward texture mapping**

   Compute 3D positions of the texture points and then project them onto the image plane.

2. **Inverse texture mapping**

   Select every pixel in the image plane (projected plane at the screen space) and identify which point of the texture image is projected there.

Coordinate Systems Involved



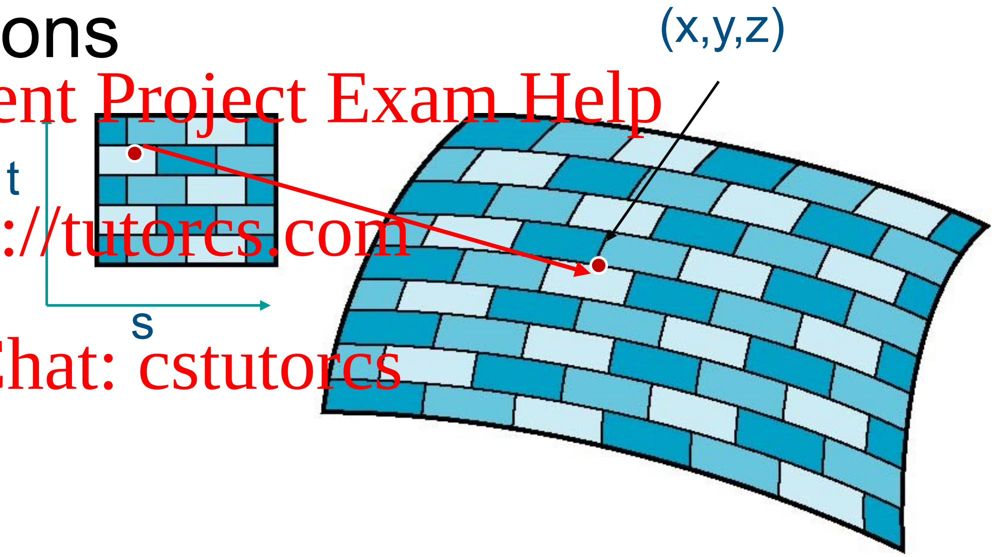
Coordinate Systems in CG application

- Parametric coordinates
  - A logical coordinate system for processing the surface and the internal space of a 3D object
- Texture coordinates
  - Used to identify points in the image to be mapped
- Local or World Coordinates
  - Used to position 3D objects
- Window Coordinates
  - Where the final output image is really produced

Forward Texture Mapping

- Consider mapping from texture coordinates to points on a surface
- Need three functions

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



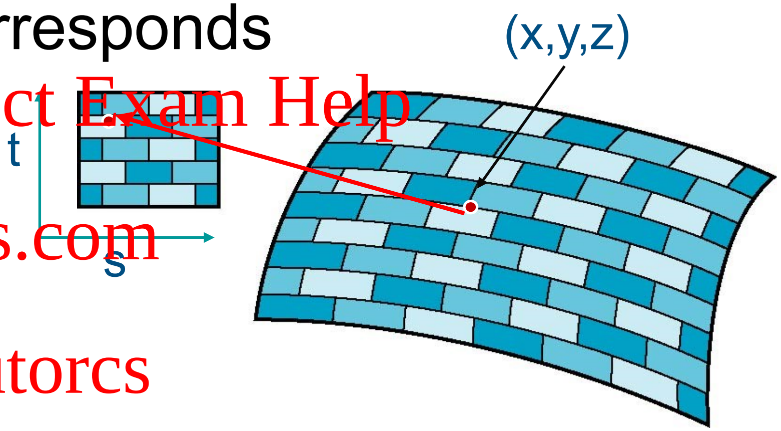
- **Main problem:** Adjacent texture points may project onto non-adjacent image points, thus creating a non-colored area.

Backward Texture Mapping

- We can consider a backward approach:
  - ☑ Given a point on an object, we identify to which point in the texture image it corresponds
- Need a map of the form:

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

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



- **Good:** Make sure every object point has a corresponding texel, particularly visibility of an object point is considered
- **Bad:** Such functions are difficult to find in general

Two-part Mapping Process

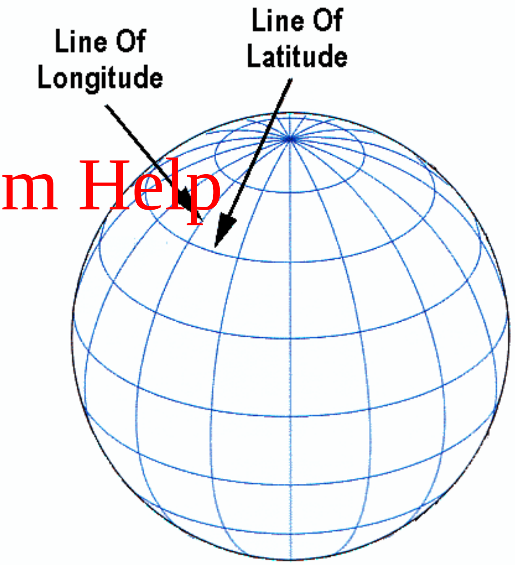
Map an texture image to a complicated shape is difficult

- Break the texture mapping process into two parts:
  1. map the texture to a simple *intermediate surface*,
  2. the textured intermediate surface is then mapped to the object

Spherical Map

We can use a parametric sphere

$x = r \cos 2\pi u$
$y = r \sin 2\pi u \cos 2\pi v$
$z = r \sin 2\pi u \sin 2\pi v$

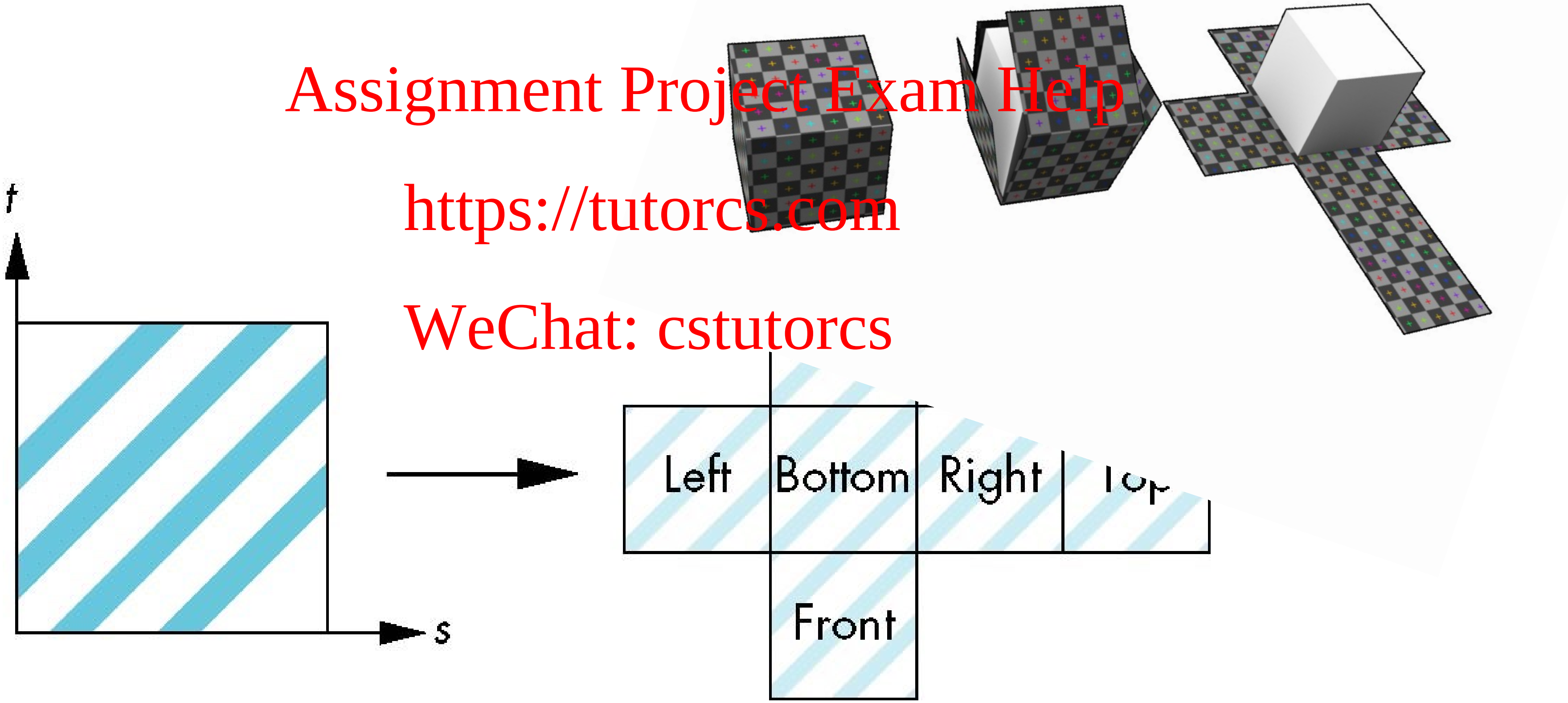


in a similar manner to the cylinder but have to decide where to put the distortion

Spheres are used in environmental maps

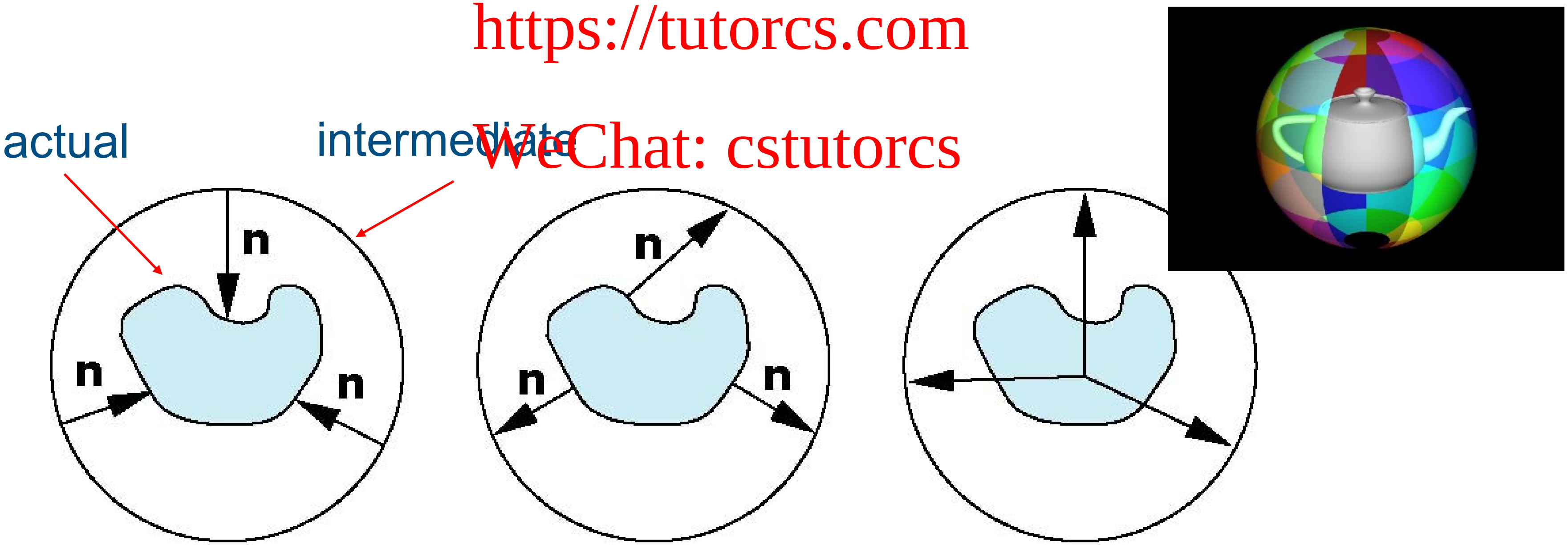
Box Mapping

- Easy to use with simple … ojection
- Also used in environme



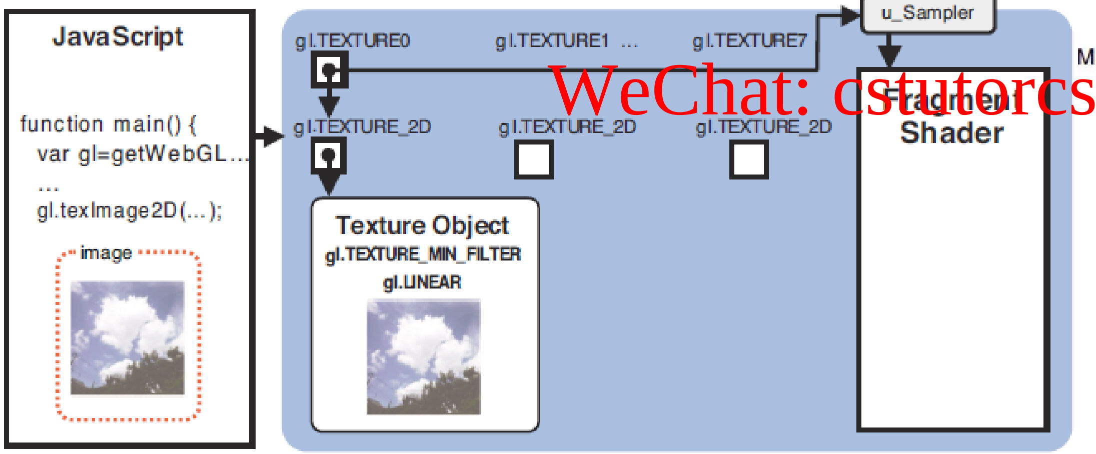
Second Mapping

- Map from intermediate object to actual object
  - Normals from intermediate to actual
  - Normals from actual to intermediate
  - Vectors from center of intermediate



Texture Mapping with WebGL

- Read texture map from an image file
- Assign texture units (gl.TEXTURE0, gl.TEXTURE1, etc.) and proper buffer spaces
- Align texture coordinates with vertices in vertex shader
- Apply colours from texture map in fragment shader



```
function main() {
  var gl=getWebGL...
  ...
  gl.texImage2D(...);
```

Figure 5.31 Set texture unit to uniform variable

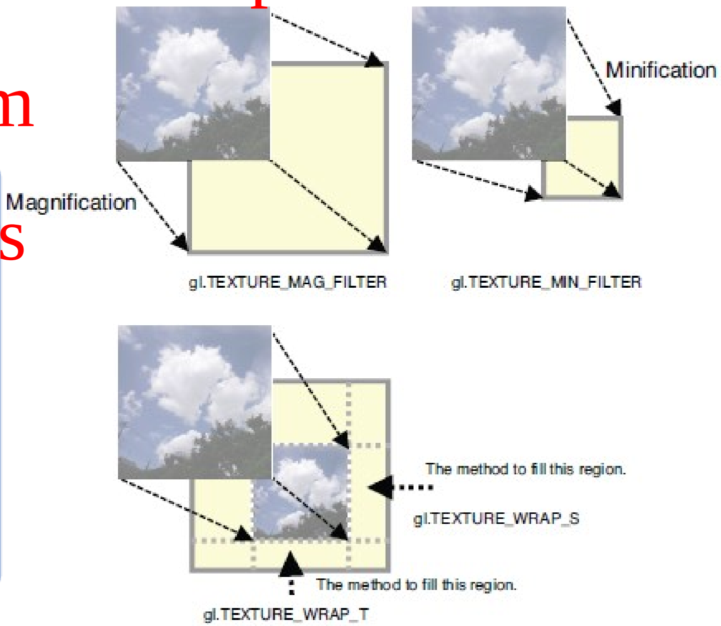
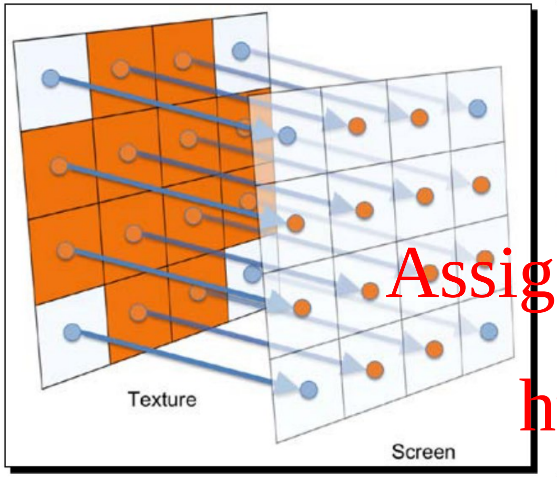
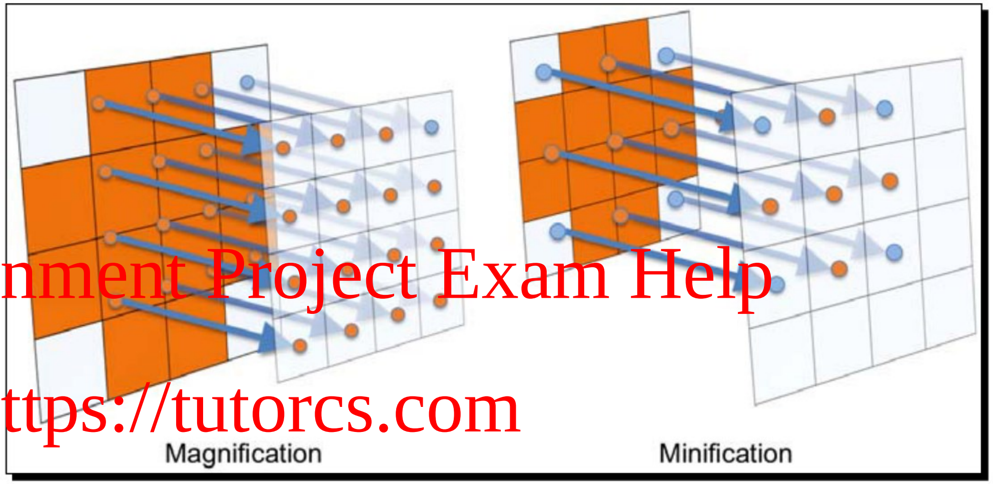
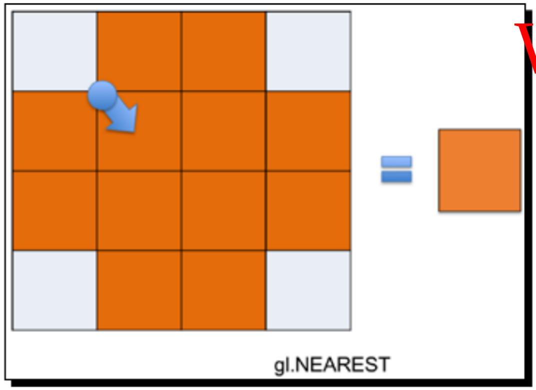


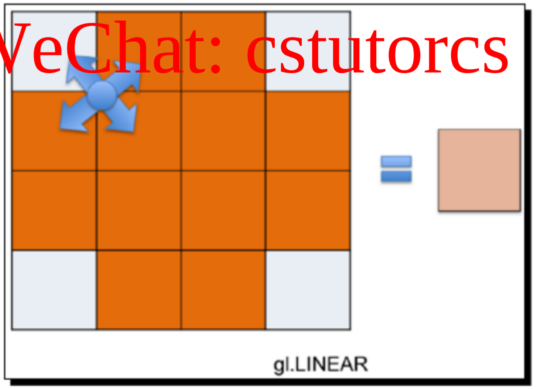
Figure 5.28 Four texture parameters and their effects

Texture Image Filtering





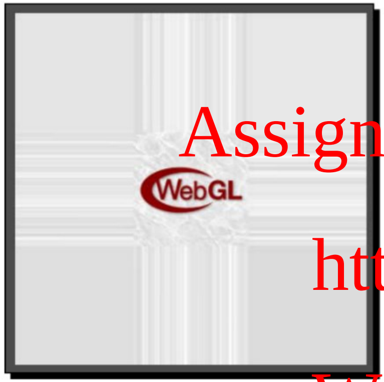




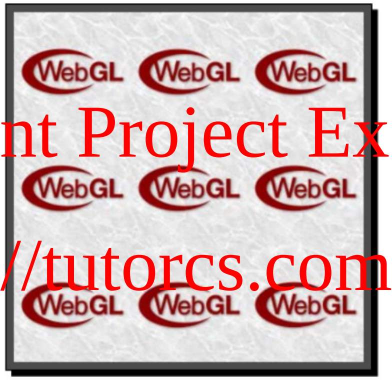
```
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
```

Texture Wrapping

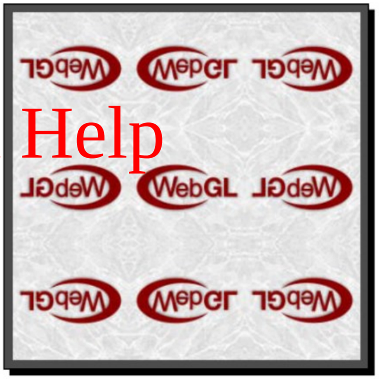
- Texture wrapping describes the behaviour of the sampler when the texture coordinates fall outside the range of 0-1.



CLAMP_TO_EDGE



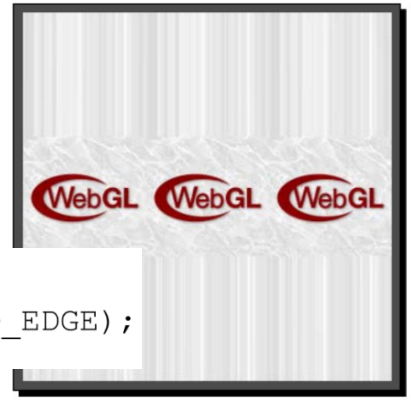
REPEAT



MIRRORED_REPEAT

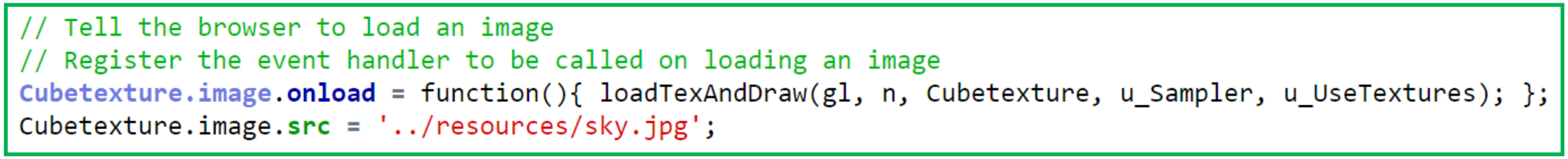
Example:

```
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.REPEAT);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
```

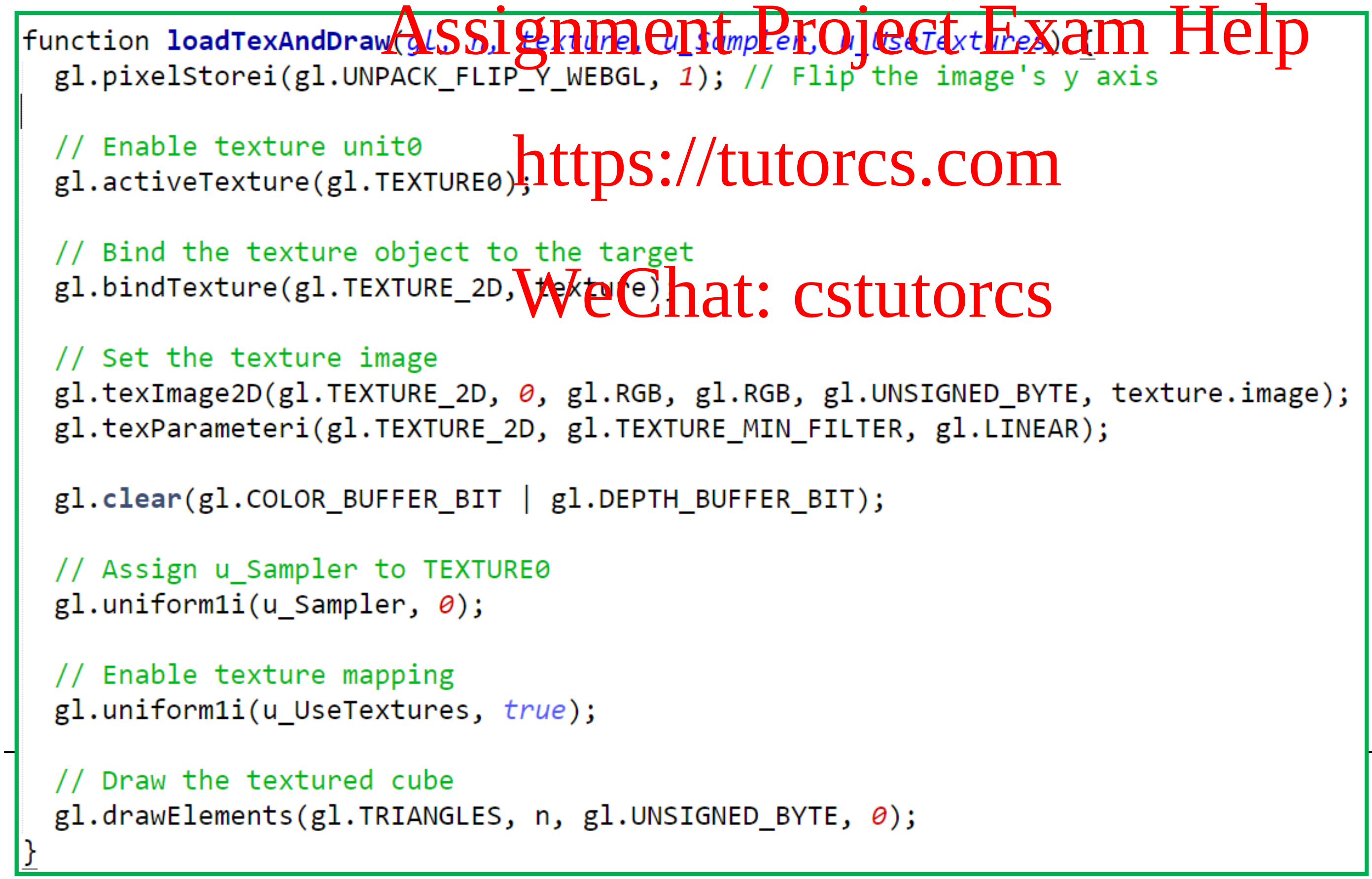


Load Texture Image

- Load texture image from a file



```
// Tell the browser to load an image
// Register the event handler to be called on loading an image
Cubetexture.image.onload = function(){ loadTexAndDraw(gl, n, Cubetexture, u_Sampler, u_UseTextures); };
Cubetexture.image.src = '../resources/sky.jpg';
```



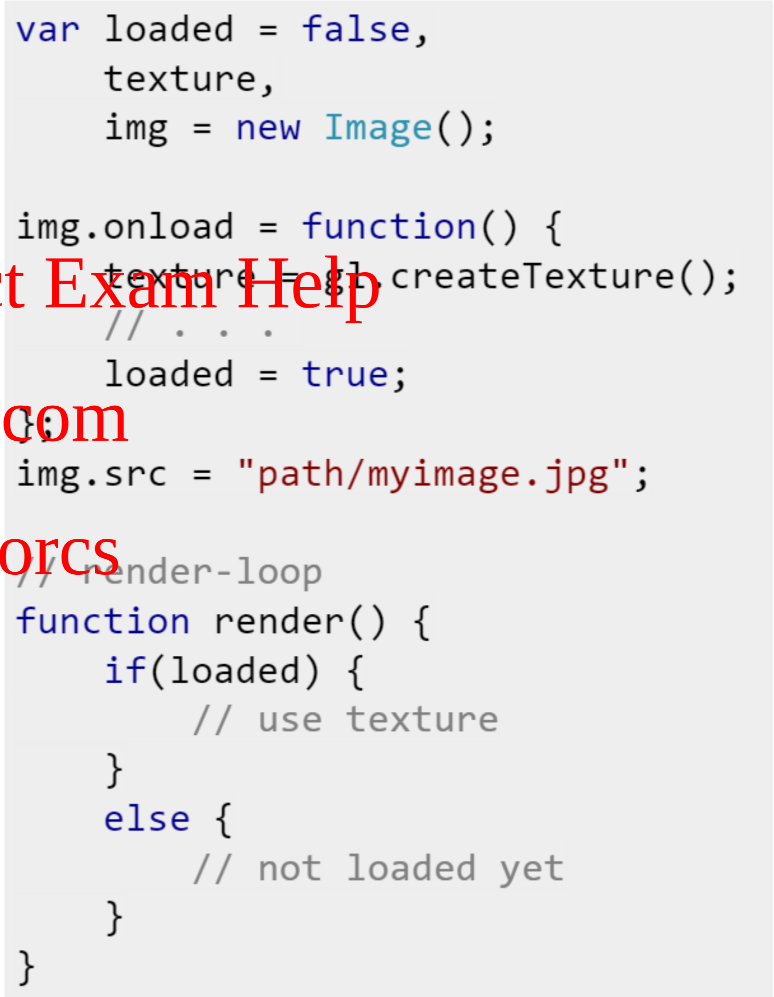
```
function loadTexAndDraw(gl, n, texture, u_Sampler, u_UseTextures) {
  gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1); // Flip the image's y axis

  // Enable texture unit0
  gl.activeTexture(gl.TEXTURE0);

  // Bind the texture object to the target
  gl.bindTexture(gl.TEXTURE_2D, texture);

  // Set the texture image
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, texture.image);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);

  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  // Assign u_Sampler to TEXTURE0
  gl.uniform1i(u_Sampler, 0);

  // Enable texture mapping
  gl.uniform1i(u_UseTextures, true);

  // Draw the textured cube
  gl.drawElements(gl.TRIANGLES, n, gl.UNSIGNED_BYTE, 0);
}
```

Load Texture Image Properly

When support user interaction, extra care is required for loading texture image.

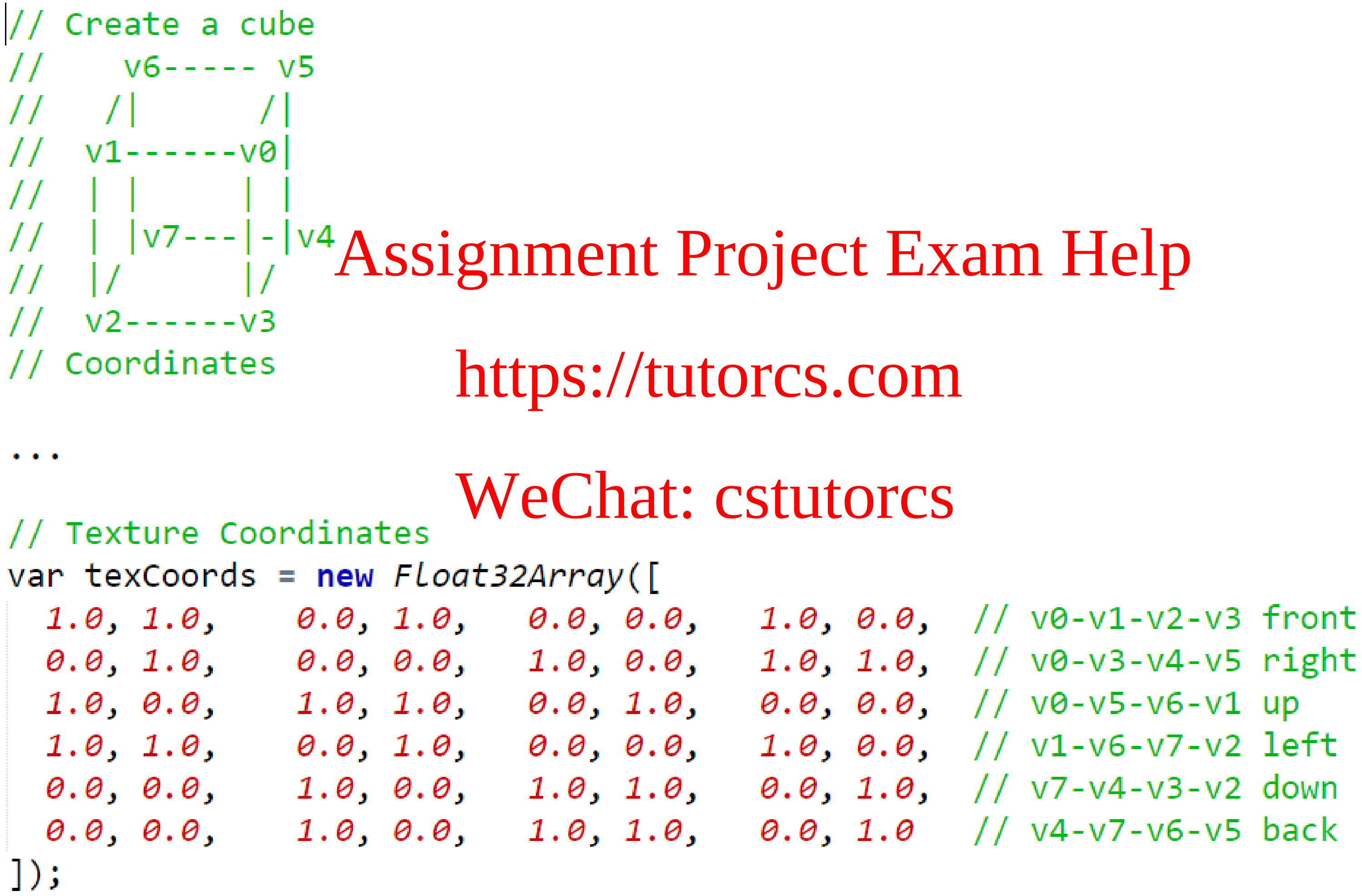
Chrome browser parameter to enable loading local texture image:

-allow-file-access-from-files



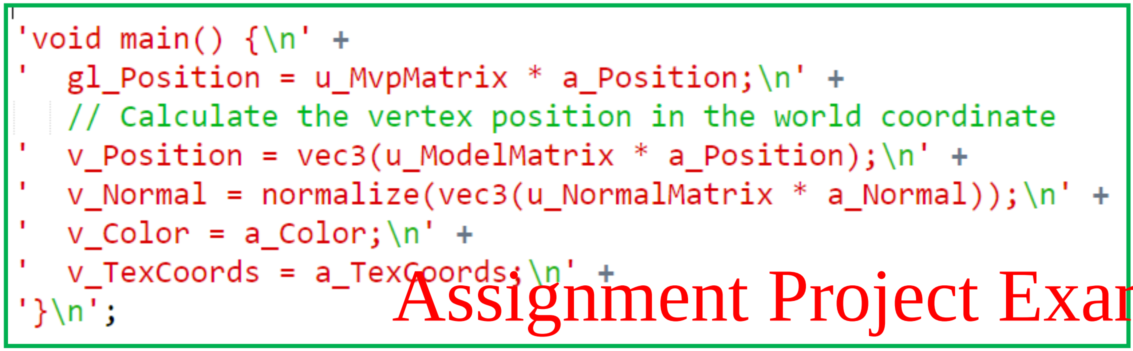
```
var loaded = false,
    texture,
    img = new Image();

img.onload = function() {
    texture = gl.createTexture();
    // . . .
    loaded = true;
};
img.src = "path/myimage.jpg";

// render-loop
function render() {
    if(loaded) {
        // use texture
    }
    else {
        // not loaded yet
    }
}
```

Texture Coordinates



```
// Create a cube
//    v6----- v5
//   /|      /|
//  v1------v0|
//  | |     | |
//  | |v7---|-|v4
//  |/      |/
//  v2------v3
// Coordinates

...

// Texture Coordinates
var texCoords = new Float32Array([
  1.0, 1.0,    0.0, 1.0,    0.0, 0.0,    1.0, 0.0,   // v0-v1-v2-v3 front
  0.0, 1.0,    0.0, 0.0,    1.0, 0.0,    1.0, 1.0,   // v0-v3-v4-v5 right
  1.0, 0.0,    1.0, 1.0,    0.0, 1.0,    0.0, 0.0,   // v0-v5-v6-v1 up
  1.0, 1.0,    0.0, 1.0,    0.0, 0.0,    1.0, 0.0,   // v1-v6-v7-v2 left
  0.0, 0.0,    1.0, 0.0,    1.0, 1.0,    0.0, 1.0,   // v7-v4-v3-v2 down
  0.0, 0.0,    1.0, 0.0,    1.0, 1.0,    0.0, 1.0    // v4-v7-v6-v5 back
]);
```

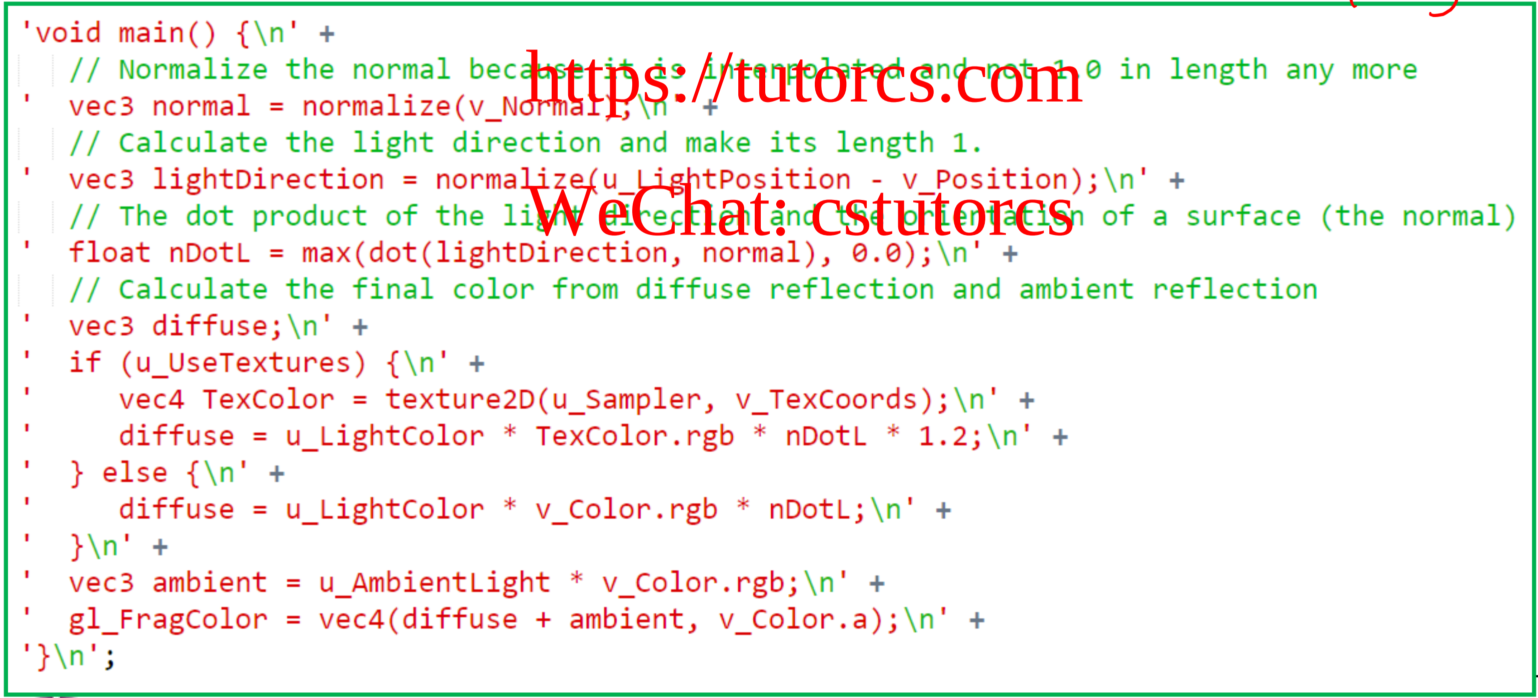
Vertex and Fragment Shaders



```
'void main() {\n' +
'  gl_Position = u_MvpMatrix * a_Position;\n' +
   // Calculate the vertex position in the world coordinate
'  v_Position = vec3(u_ModelMatrix * a_Position);\n' +
'  v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
'  v_Color = a_Color;\n' +
'  v_TexCoords = a_TexCoords;\n' +
'}\n';
```

Vertex Shader

Fragment Shader



```
'void main() {\n' +
   // Normalize the normal because it is interpolated and not 1.0 in length any more
'  vec3 normal = normalize(v_Normal);\n' +
   // Calculate the light direction and make its length 1.
'  vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
   // The dot product of the light direction and the orientation of a surface (the normal)
'  float nDotL = max(dot(lightDirection, normal), 0.0);\n' +
   // Calculate the final color from diffuse reflection and ambient reflection
'  vec3 diffuse;\n' +
'  if (u_UseTextures) {\n' +
'     vec4 TexColor = texture2D(u_Sampler, v_TexCoords);\n' +
'     diffuse = u_LightColor * TexColor.rgb * nDotL * 1.2;\n' +
'  } else {\n' +
'     diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
'  }\n' +
'  vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
'  gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
'}\n';
```

Summary

- Texture mapping definition
- Mapping methods
- WebGL Implementation

References:

- **Computer Graphics with Open GL [Chapter 16]**
- **WebGL Programming Guide [Ch. 8]**