OpenGL Textures & WebGL Shaders

> **Covers Section 11 + Lab 4 (WebGL Textures)**
> **Focus**: Texture Mapping, Sampling, Filtering, Procedural Textures, Bump Maps
> **Exam Ready**: Lab 4.1–4.3 full solutions + common exam patterns
> **Tip**: Exam may ask to **sample textures**, **implement procedural textures**, or **add bump mapping**

1. Texture Basics (Section 11)

Term	Meaning
Texture	2D image applied to 3D surface
UV Coordinates	`(u,v)` in [0,1] → maps texture to geometry
Texel	Pixel in texture
Sampler	`sampler2D` in shader → reads texture
Mipmap	Pre-scaled versions of texture for minification

2. WebGL Texture Setup (JS)

```
// 1. Create & bind texture
const tex = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, tex);

// 2. Load image
const img = new Image();
img.onload = () => {
    gl.bindTexture(gl.TEXTURE_2D, tex);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, img);

    // 3. Generate mipmaps
    gl.generateMipmap(gl.TEXTURE_2D);

    // 4. Set filters
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER,
gl.LINEAR_MIPMAP_LINEAR);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.REPEAT);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.REPEAT);

    render();
};
img.src = 'brick.jpg';
```

LAB 4.1: Sample 2D Texture (Albedo)

Vertex Shader

```
attribute vec2 a_coords;
attribute vec2 a_texCoords;
varying vec2 v_texCoords;
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main() {
    v_texCoords = a_texCoords;
    gl_Position = projection * view * model * vec4(a_coords, 0.0, 1.0);
}
```

Fragment Shader

```
precision mediump float;
varying vec2 v_texCoords;
uniform sampler2D u_texture1;

void main() {
    gl_FragColor = texture2D(u_texture1, v_texCoords);
}
```

JS: Bind Texture to Uniform

```
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, texture);
gl.uniform1i(gl.getUniformLocation(program, 'u_texture1'), 0);
```

LAB 4.2: Procedural Textures (4 Types)

Fragment Shader (Complete)

```
#ifdef GL_FRAGMENT_PRECISION_HIGH
   precision highp float;
#else
   precision mediump float;
#endif

uniform mat3 normalMatrix;
uniform int textureNum;
```

```
uniform float scale;
varying vec3 v_normal;
varying vec3 v_eyeCoords;
varying vec2 v_texCoords;

vec3 N = normalize(normalMatrix * v_normal);
vec3 L = normalize(-v_eyeCoords);
float diffuseFactor = max(dot(N, L), 0.0);
vec3 color = vec3(0.0);
vec2 uv = v_texCoords * max(scale, 0.0001);

// Hash function for noise
float hash(vec2 p) {
    return fract(sin(dot(p, vec2(12.9898, 78.233))) * 43758.5453);
}

// Smoothstep interpolation
float smoothstep(float edge0, float edge1, float x) {
    float t = clamp((x - edge0) / (edge1 - edge0), 0.0, 1.0);
    return t * t * (3.0 - 2.0 * t);
}

if (textureNum == 0) {
    // 0) Checkerboard 2D
    float x = floor(uv.x * 10.0);
    float y = floor(uv.y * 10.0);
    color = mod(x + y, 2.0) < 1.0 ? vec3(1.0, 0.8, 0.6) : vec3(0.3, 0.2, 0.1);

} else if (textureNum == 1) {
    // 1) Sine Function (wavy bands)
    float wave = sin(uv.x * 20.0) * sin(uv.y * 20.0);
    wave = wave * 0.5 + 0.5;  // [-1,1] → [0,1]
    color = vec3(wave, wave * 0.7, wave * 0.4);

} else if (textureNum == 2) {
    // 2) Value Noise 2D
    vec2 i = floor(uv * 10.0);
    vec2 f = fract(uv * 10.0);
    f = smoothstep(0.0, 1.0, f);

    float a = hash(i);
    float b = hash(i + vec2(1.0, 0.0));
    float c = hash(i + vec2(0.0, 1.0));
    float d = hash(i + vec2(1.0, 1.0));

    float noise = mix(mix(a, b, f.x), mix(c, d, f.x), f.y);
    color = vec3(noise);

} else if (textureNum == 3) {
    // 3) Radial Rings
    vec2 center = uv - 0.5;
    float r = length(center) * 20.0;
    float rings = sin(r) * 0.5 + 0.5;
    color = vec3(rings, rings * 0.8, rings * 0.5);
```

```
}

gl_FragColor = vec4(diffuseFactor * color, 1.0);
```

LAB 4.3: Creative Texture Effect (Bump Mapping)

Add Normal Map Texture

```
// Load normal map
const normalImg = new Image();
normalImg.onload = () => {
    const normalTex = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, normalTex);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE,
normalImg);
    gl.generateMipmap(gl.TEXTURE_2D);
    gl.uniform1i(gl.getUniformLocation(program, 'u_normalMap'), 1);
};
normalImg.src = 'normal.jpg';
```

Vertex Shader (Pass TBN)

```
attribute vec3 a_position;
attribute vec3 a_normal;
attribute vec3 a_tangent;
attribute vec2 a_texCoords;
uniform mat4 u_modelview;
uniform mat4 u_projection;
uniform mat3 u_normalMatrix;

varying vec3 v_tangent;
varying vec3 v_bitangent;
varying vec3 v_normal;
varying vec2 v_texCoords;
varying vec3 v_position;

void main() {
    vec4 pos = u_modelview * vec4(a_position, 1.0);
    v_position = pos.xyz;
    v_normal = normalize(u_normalMatrix * a_normal);
    v_tangent = normalize(u_normalMatrix * a_tangent);
    v_bitangent = cross(v_normal, v_tangent);
    v_texCoords = a_texCoords;
    gl_Position = u_projection * pos;
}
```

Fragment Shader (Tangent-Space Normal Mapping)

```
precision mediump float;
varying vec3 v_tangent;
varying vec3 v_bitangent;
varying vec3 v_normal;
varying vec2 v_texCoords;
varying vec3 v_position;

uniform sampler2D u_texture;
uniform sampler2D u_normalMap;
uniform vec3 u_lightPos;

void main() {
    // Sample normal map (in [0,1] → [-1,1])
    vec3 normalMap = texture2D(u_normalMap, v_texCoords).rgb;
    normalMap = normalMap * 2.0 - 1.0;

    // TBN matrix
    mat3 TBN = mat3(v_tangent, v_bitangent, v_normal);
    vec3 N = normalize(TBN * normalMap);
    vec3 L = normalize(u_lightPos - v_position);

    float diff = max(dot(N, L), 0.0);
    vec3 texColor = texture2D(u_texture, v_texCoords).rgb;
    vec3 final = texColor * diff;

    gl_FragColor = vec4(final, 1.0);
}
```

4. Other Creative Ideas (Exam Flex)

Effect	Code Snippet
MatCap	`vec3 viewDir = normalize(v_position); vec2 uv = viewDir.xy * 0.5 + 0.5; gl_FragColor = texture2D(matcap, uv);`
UV Animation (Water)	`v_texCoords += vec2(u_time * 0.1, 0);`
Scanlines	`float scan = sin(v_texCoords.y * 800.0) * 0.05; gl_FragColor.rgb -= scan;`
Blend with Mask	`float mask = texture2D(maskTex, uv).r; gl_FragColor = mix(tex1, tex2, mask);`

5. Exam Question Patterns

Type 1: Sample Texture

```
gl_FragColor = texture2D(u_tex, v_texCoords);
```

Type 2: Procedural Checkerboard

```
float checker = mod(floor(uv.x*10.0) + floor(uv.y*10.0), 2.0);
color = checker < 1.0 ? vec3(1.0) : vec3(0.0);
```

Type 3: Fix Mipmap

```
gl.generateMipmap(gl.TEXTURE_2D);  // Missing!
```

Type 4: Bump Mapping

```
vec3 normalMap = texture2D(u_normal, uv).rgb * 2.0 - 1.0;
vec3 N = normalize(TBN * normalMap);
```