OpenGL Programmable Pipeline

> **Covers Section 10 + Lab 3 (WebGL 2.0)**
> **Focus**: Programmable Pipeline vs Fixed-Function, Shader Basics, Lab 3 Extensions
> **Exam Ready**: Detailed Lab 3.1–3.3 solutions + exam-style questions
> **Tip**: Exam may test shader conversions, lighting models, or creative effects

---

1. Programmable Pipeline Overview (Section 10)

- **Fixed-Function Pipeline**: Hard-coded stages (e.g., OpenGL 1.1 with `glLight`, `glMaterial`). Limited flexibility.
- **Programmable Pipeline**: Custom shaders (GLSL) control vertex and fragment processing. Used in modern WebGL/GLES.
- **Key Stages**:
  - **Vertex Shader**: Transforms vertices (position, normal, color).
  - **Fragment Shader**: Colors pixels (lighting, texturing).

---

2. Shader Basics

2.1 Vertex Shader

- Processes each vertex.
- Outputs: Transformed position (`gl_Position`), varying data (e.g., color, normal).
- Example:

```
attribute vec3 a_position;
attribute vec3 a_normal;
uniform mat4 u_modelview;
uniform mat4 u_projection;
varying vec3 v_normal;

void main() {
    gl_Position = u_projection * u_modelview * vec4(a_position, 1.0);
    v_normal = a_normal;  // Pass to fragment
}
```

2.2 Fragment Shader

Colors each pixel. Inputs: Varying data from vertex shader. Example:

```
precision mediump float;
varying vec3 v_normal;
void main() {
```

```
    gl_FragColor = vec4(abs(v_normal), 1.0);  // Normal-based color
}
```

2.3 Variable Types

- **Attribute**: Per-vertex data (e.g., `a_position`, `a_color`).
- **Uniform**: Global data (e.g., `u_modelview`, `u_lightPos`).
- **Varying**: Interpolated data between vertex and fragment (e.g., `v_normal`).

---

3. LAB 3.1: Solid Color Cube + Rotation

Vertex Shader

```
attribute vec3 a_position;
uniform mat4 u_modelview;
uniform mat4 u_projection;

void main() {
    gl_Position = u_projection * u_modelview * vec4(a_position, 1.0);
}
```

Fragment Shader

```
precision mediump float;
void main() {
    gl_FragColor = vec4(0.4, 0.7, 1.0, 1.0);  // Solid sky blue
}
```

JS Implementation

```
const canvas = document.getElementById('webgl-canvas');
const gl = canvas.getContext('webgl');

gl.enable(gl.DEPTH_TEST);

// Buffers
const vertBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vertBuffer);
gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

const indexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, indices, gl.STATIC_DRAW);

// Shader Program
const program = createProgram(gl, vertexShaderSource, fragmentShaderSource);
```

```
gl.useProgram(program);

// Attributes
const posLoc = gl.getAttribLocation(program, 'a_position');
gl.enableVertexAttribArray(posLoc);
gl.bindBuffer(gl.ARRAY_BUFFER, vertBuffer);
gl.vertexAttribPointer(posLoc, 3, gl.FLOAT, false, 0, 0);

// Matrices
let modelview = mat4.create();
let projection = mat4.create();
mat4.perspective(projection, Math.PI/4, 1, 0.1, 100);
gl.uniformMatrix4fv(gl.getUniformLocation(program, 'u_projection'), false,
projection);

// Animation
let angle = 0;
function render() {
    angle += 0.01;
    mat4.identity(modelview);
    mat4.translate(modelview, modelview, [0, 0, -5]);
    mat4.rotateY(modelview, modelview, angle);

    gl.uniformMatrix4fv(gl.getUniformLocation(program, 'u_modelview'), false,
modelview);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    gl.drawElements(gl.TRIANGLES, indices.length, gl.UNSIGNED_SHORT, 0);
    requestAnimationFrame(render);
}
render();
```

---

4. LAB 3.2: Lambert + Cell Shading

Add Color Buffer

```
const colorBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
gl.bufferData(gl.ARRAY_BUFFER, colors, gl.STATIC_DRAW);

const colorLoc = gl.getAttribLocation(program, 'a_color');
gl.enableVertexAttribArray(colorLoc);
gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
gl.vertexAttribPointer(colorLoc, 3, gl.FLOAT, false, 0, 0);
```

Vertex Shader

```
attribute vec3 a_position;
attribute vec3 a_normal;
```

```
attribute vec3 a_color;
uniform mat4 u_modelview;
uniform mat4 u_projection;
uniform mat4 u_normalMatrix;
varying vec3 v_normal;
varying vec3 v_position;
varying vec3 v_color;

void main() {
    vec4 pos = u_modelview * vec4(a_position, 1.0);
    v_position = pos.xyz;
    v_normal = normalize((u_normalMatrix * vec4(a_normal, 0.0)).xyz);
    v_color = a_color;
    gl_Position = u_projection * pos;
}
```

Fragment Shader (Lambert + Cell)

```
precision mediump float;
varying vec3 v_normal;
varying vec3 v_position;
varying vec3 v_color;
uniform vec3 u_lightPos;

void main() {
    vec3 L = normalize(u_lightPos - v_position);
    float NdotL = max(0.0, dot(v_normal, L));

    // Cell shading: 4 levels
    float levels = 4.0;
    float shade = floor(NdotL * levels) / (levels - 1.0);  // Adjusted for range

    vec3 finalColor = v_color * shade;
    gl_FragColor = vec4(finalColor, 1.0);
}
```

JS: Normal Matrix & Light

```
let normalMatrix = mat3.create();
mat3.normalFromMat4(normalMatrix, modelview);
gl.uniformMatrix3fv(gl.getUniformLocation(program, 'u_normalMatrix'), false,
normalMatrix);

const lightPos = [5, 5, 5];
gl.uniform3fv(gl.getUniformLocation(program, 'u_lightPos'), lightPos);
```

---

5. LAB 3.3: Creative Shader Ideas

Vertex Shader (Wobble Example)

```
attribute vec3 a_position;
uniform mat4 u_modelview;
uniform mat4 u_projection;
uniform float u_time;
varying vec3 v_pos;

void main() {
    vec3 pos = a_position;
    pos.y += sin(u_time + pos.x * 10.0) * 0.1;  // Wobble
    gl_Position = u_projection * u_modelview * vec4(pos, 1.0);
    v_pos = pos;
}
```

Fragment Shader (Hologram Effect)

```
precision mediump float;
varying vec3 v_pos;
uniform float u_time;

void main() {
    float flicker = sin(u_time * 10.0) * 0.1 + 0.9;
    float scan = sin(v_pos.y * 20.0 + u_time * 5.0) * 0.1 + 0.5;
    gl_FragColor = vec4(0.1, 0.8, 1.0, 0.3 * flicker * scan);  // Blue hologram
}
```

JS: Add Time Uniform

```
let time = 0;
function render() {
    time += 0.01;
    gl.uniform1f(gl.getUniformLocation(program, 'u_time'), time);
    // ... rest of render loop
}
```

---

6. Lighting Models in Shaders

6.1 Lambert (Per-Vertex)

```
varying vec3 v_normal;
varying vec3 v_position;
uniform vec3 u_lightPos;

void main() {
```

```
    vec3 L = normalize(u_lightPos - v_position);
    float diff = max(0.0, dot(v_normal, L));
    gl_FragColor = vec4(vec3(1.0, 0.5, 0.2) * diff, 1.0);  // Orange diffuse
}
```

6.2 Phong (Per-Pixel)

```
varying vec3 v_normal;
varying vec3 v_position;
uniform vec3 u_lightPos;
uniform float u_shininess;

void main() {
    vec3 N = normalize(v_normal);
    vec3 L = normalize(u_lightPos - v_position);
    vec3 V = normalize(-v_position);  // To viewer
    vec3 R = reflect(-L, N);
    float diff = max(0.0, dot(N, L));
    float spec = pow(max(0.0, dot(R, V)), u_shininess);
    vec3 color = vec3(0.8, 0.4, 0.2) * diff + vec3(1.0) * spec * 0.5;
    gl_FragColor = vec4(color, 1.0);
}
```

---

7. Exam Question Patterns

Type 1: Shader Conversion

"Convert this OpenGL lighting to WebGL"

OpenGL:

```
glLightfv(GL_LIGHT0, GL_POSITION, {5,5,5,1});
glMaterialfv(GL_FRONT, GL_DIFFUSE, {1,0,0,1});
```

WebGL:

```
// Vertex
varying vec3 v_pos, v_normal;
void main() {
    v_pos = (u_modelview * vec4(a_position, 1.0)).xyz;
    v_normal = normalize((u_normalMatrix * vec4(a_normal, 0.0)).xyz);
    gl_Position = u_projection * vec4(v_pos, 1.0);
}

// Fragment
uniform vec3 u_lightPos;
void main() {
```

```
    vec3 L = normalize(u_lightPos - v_pos);
    float diff = max(0.0, dot(v_normal, L));
    gl_FragColor = vec4(vec3(1,0,0) * diff, 1.0);
}
```

Type 2: Add Effect

"Add cell shading to this shader"

```
float levels = 4.0;
float shade = floor(diff * levels) / (levels - 1.0);
gl_FragColor = vec4(v_color * shade, 1.0);
```

Type 3: Debug Shader

"Fix this code"

Wrong:

```
gl.vertexAttribPointer(posLoc, 3, gl.FLOAT, false, 0, 0);  // No buffer bind
```

Correct:

```
gl.bindBuffer(gl.ARRAY_BUFFER, vertBuffer);
gl.vertexAttribPointer(posLoc, 3, gl.FLOAT, false, 0, 0);
```