ChatGPT

QED-Driven Structural Color Visualizer

Inspired by light–matter interactions in physics, this single-page WebGL demo generates dynamic interference patterns that produce vivid **structural colors** (colors from light interference, not pigments [1] [2]). We use a fragment shader on a full-screen canvas to simulate sinusoidal wave interference: for example, thin-film effects like soap bubbles create iridescent colors by constructive/destructive interference [1]. The shader also overlays rainbow spectrum shifts and "particle-wave" motion, evoking **quantum electrodynamics (QED)** concepts (photons and electrons interacting) [3]. The UI provides controls to toggle modes and adjust wave frequency/speed, exploring different color spaces and harmonic effects.

Implementation Details

- **HTML Structure:** A full-window `<canvas>` element is used for WebGL rendering, with an overlaid `<div>` containing UI controls (mode selector, frequency slider, speed slider).
- **WebGL Setup:** We acquire a raw WebGL context from the canvas, compile simple vertex and fragment shaders, and draw a full-screen quad each frame. The vertex shader passes normalized UV coordinates to the fragment shader.
- **Shader Logic (Fragment Shader):** In each pixel, we compute a 2D position `(x,y)` relative to center. We then generate **wave patterns** using sine functions (`sin` waves along radial distance and along axes) and animate them with time (`uTime`). These waves **interfere** to form complex patterns. In *Interference* mode, the intensity (`0.5+0.5*sin(...)`) modulates brightness and we map the result to a hue varying with distance (`structuralHue = fract(dist+t)`), mimicking thin-film interference [1] [2]. In *Rainbow* mode, we map the phase pattern into hue shifts (`rainbowHue = fract(angle + pattern + t)`) for a flowing spectrum effect. We convert hue/saturation/value to RGB in the shader for color output.
- **UI Controls:** A dropdown selects **Mode** (Interference vs. Rainbow), and sliders control **Frequency** (spatial frequency of waves) and **Speed** (animation rate). Event listeners update corresponding shader uniforms (`uMode`, `uFreq`, `uSpeed`) in real time. This lets users switch between color domains and modulate harmonics dynamically.
- **Animation Loop:** Using `requestAnimationFrame`, we continuously update the uniform `uTime` (millisecond timestamp) and redraw the quad. The result is a reactive, evolving pattern that nods to QED's wave-particle duality – a playful visualization of light interacting with matter [3].

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>QED-inspired Structural Color Visualizer</title>
  <style>
    body { margin: 0; overflow: hidden; background: #000; color: #fff; font-
family: sans-serif; }
    #canvas { display: block; width: 100vw; height: 100vh; }
```

```
    #ui { position: absolute; top: 10px; left: 10px; background: rgba(0,0,0,
0.5); padding: 10px; border-radius: 5px; }
    #ui label { margin: 4px; font-size: 14px; }
    #ui select, #ui input { vertical-align: middle; }
  </style>
</head>
<body>
  <canvas id="canvas"></canvas>
  <div id="ui">
    <label>Mode:
      <select id="mode">
        <option value="0">Interference</option>
        <option value="1">Rainbow</option>
      </select>
    </label><br>
    <label>Frequency: <span id="freqVal">10</span>
      <input type="range" id="freq" min="1" max="30" value="10">
    </label><br>
    <label>Speed: <span id="speedVal">1.0</span>
      <input type="range" id="speed" min="0" max="10" step="0.1" value="1.0">
    </label>
  </div>
  <script>
    // Get WebGL context and setup
    const canvas = document.getElementById('canvas');
    const gl = canvas.getContext('webgl');
    if (!gl) alert('WebGL not supported.');
    function resize() {
      canvas.width = window.innerWidth;
      canvas.height = window.innerHeight;
      gl.viewport(0, 0, gl.drawingBufferWidth, gl.drawingBufferHeight);
      gl.uniform2f(uResolutionLoc, canvas.width, canvas.height);
    }
    window.addEventListener('resize', () => resize());
    resize();

    // Vertex shader for full-screen quad
    const vertSrc = `
      attribute vec2 aPosition;
      varying vec2 vUV;
      void main() {
        vUV = aPosition * 0.5 + 0.5;
        gl_Position = vec4(aPosition, 0.0, 1.0);
      }
    `;
    // Fragment shader: interference + color mapping
    const fragSrc = `
      precision mediump float;
```

```
  uniform float uTime;
  uniform vec2 uResolution;
  uniform int uMode;
  uniform float uFreq;
  uniform float uSpeed;
  varying vec2 vUV;
  // HSV to RGB conversion
  vec3 hsv2rgb(vec3 c) {
    vec4 K = vec4(1.0, 2.0/3.0, 1.0/3.0, 3.0);
    vec3 p = abs(fract(c.xxx + K.xyz) * 6.0 - K.www);
    return c.z * mix(K.xxx, clamp(p - K.xxx, 0.0, 1.0), c.y);
  }
  void main() {
    vec2 uv = vUV;
    vec2 pos = uv - 0.5;
    float dist = length(pos) * 2.0;
    float angle = atan(pos.y, pos.x);
    float hue = (angle / (2.0 * 3.14159)) + 0.5;
    float t = uTime * uSpeed * 0.001;
    float wave1 = sin((dist * uFreq) + t);
    float wave2 = sin((pos.x * uFreq * 1.5) + t * 0.7);
    float pattern = wave1 + wave2;
    vec3 color;
    if (uMode == 0) {
      // Interference mode: modulate brightness by pattern
      float intensity = 0.5 + 0.5 * pattern;
      float structuralHue = fract(dist + t);  // shifting hue with distance
      color = hsv2rgb(vec3(structuralHue, 1.0, intensity));
    } else {
      // Rainbow mode: map pattern into hue + fixed brightness
      float rainbowHue = fract(hue + pattern * 0.1 + t);
      color = hsv2rgb(vec3(rainbowHue, 1.0, 0.8));
    }
    gl_FragColor = vec4(color, 1.0);
  }
`;
// Compile and link shaders
function compileShader(src, type) {
  const s = gl.createShader(type);
  gl.shaderSource(s, src);
  gl.compileShader(s);
  if (!gl.getShaderParameter(s, gl.COMPILE_STATUS))
    console.error('Shader compile error:', gl.getShaderInfoLog(s));
  return s;
}
const vertShader = compileShader(vertSrc, gl.VERTEX_SHADER);
const fragShader = compileShader(fragSrc, gl.FRAGMENT_SHADER);
const program = gl.createProgram();
```

```
gl.attachShader(program, vertShader);
gl.attachShader(program, fragShader);
gl.linkProgram(program);
if (!gl.getProgramParameter(program, gl.LINK_STATUS))
  console.error('Program link error:', gl.getProgramInfoLog(program));
gl.useProgram(program);

// Full-screen quad vertices
const quadVerts = new Float32Array([
  -1, -1,  1, -1,  -1, 1,
   1, -1,  1,  1,  -1, 1
]);
const buffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
gl.bufferData(gl.ARRAY_BUFFER, quadVerts, gl.STATIC_DRAW);
const aPos = gl.getAttribLocation(program, 'aPosition');
gl.enableVertexAttribArray(aPos);
gl.vertexAttribPointer(aPos, 2, gl.FLOAT, false, 0, 0);

// Uniform locations
const uTimeLoc = gl.getUniformLocation(program, 'uTime');
const uResolutionLoc = gl.getUniformLocation(program, 'uResolution');
const uModeLoc = gl.getUniformLocation(program, 'uMode');
const uFreqLoc = gl.getUniformLocation(program, 'uFreq');
const uSpeedLoc = gl.getUniformLocation(program, 'uSpeed');
gl.uniform2f(uResolutionLoc, canvas.width, canvas.height);
let startTime = Date.now();

// UI event handlers
const modeSelect = document.getElementById('mode');
const freqSlider = document.getElementById('freq');
const speedSlider = document.getElementById('speed');
const freqVal = document.getElementById('freqVal');
const speedVal = document.getElementById('speedVal');
modeSelect.addEventListener('change', () => {
  gl.uniform1i(uModeLoc, parseInt(modeSelect.value));
});
freqSlider.addEventListener('input', () => {
  const v = parseFloat(freqSlider.value);
  freqVal.textContent = v;
  gl.uniform1f(uFreqLoc, v);
});
speedSlider.addEventListener('input', () => {
  const v = parseFloat(speedSlider.value);
  speedVal.textContent = v;
  gl.uniform1f(uSpeedLoc, v);
});
// Initialize uniform values from UI
```

```
    gl.uniform1i(uModeLoc, parseInt(modeSelect.value));
    gl.uniform1f(uFreqLoc, parseFloat(freqSlider.value));
    gl.uniform1f(uSpeedLoc, parseFloat(speedSlider.value));

    // Render loop
    function render() {
      const now = Date.now();
      gl.uniform1f(uTimeLoc, now - startTime);
      gl.clear(gl.COLOR_BUFFER_BIT);
      gl.drawArrays(gl.TRIANGLES, 0, 6);
      requestAnimationFrame(render);
    }
    render();
  </script>
</body>
</html>
```

This code uses WebGL directly (no frameworks) on a full-screen canvas. The fragment shader generates animated, wave-based color patterns. Structural color arises from wave interference: the code synthesizes this by combining sine waves and mapping them to hue/brightness [1] [2]. The UI lets you switch modes and adjust frequencies in real time. In *Interference* mode, the shader's color varies with distance and time like a multi-layer thin-film, evoking phenomena discussed by Feynman [1]; in *Rainbow* mode it cycles through hues to overlay a spectrum. Together, these capture the "playful interplay" of physics and color – echoing QED's light–matter interactions [3] in a dynamic visual form.

Sources: We draw on principles of structural coloration (light interference on microscopic layers) [2] [1] and on descriptions of QED's particle-photon dynamics [3] to guide the shader design. All code is custom, but the colors and behavior are informed by these physical concepts.

---

[1] The Feynman Lectures on Physics Vol. I Ch. 35: Color Vision
https://www.feynmanlectures.caltech.edu/I_35.html

[2] Structural coloration - Wikipedia
https://en.wikipedia.org/wiki/Structural_coloration

[3] Quantum electrodynamics - Wikipedia
https://en.wikipedia.org/wiki/Quantum_electrodynamics