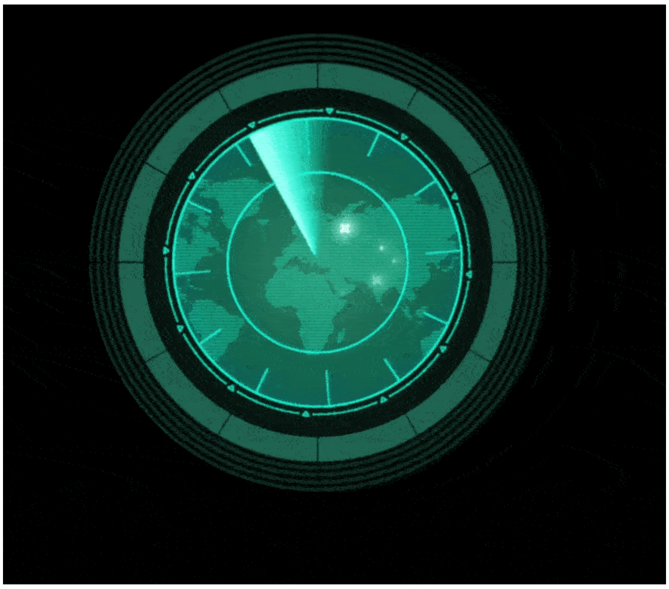
HUD Project

A portfolio project that showcases proficiency in shaders, UI, and digital humans.

Reference done in DCC: https://www.youtube.com/watch?v=r-l74gVYgM8

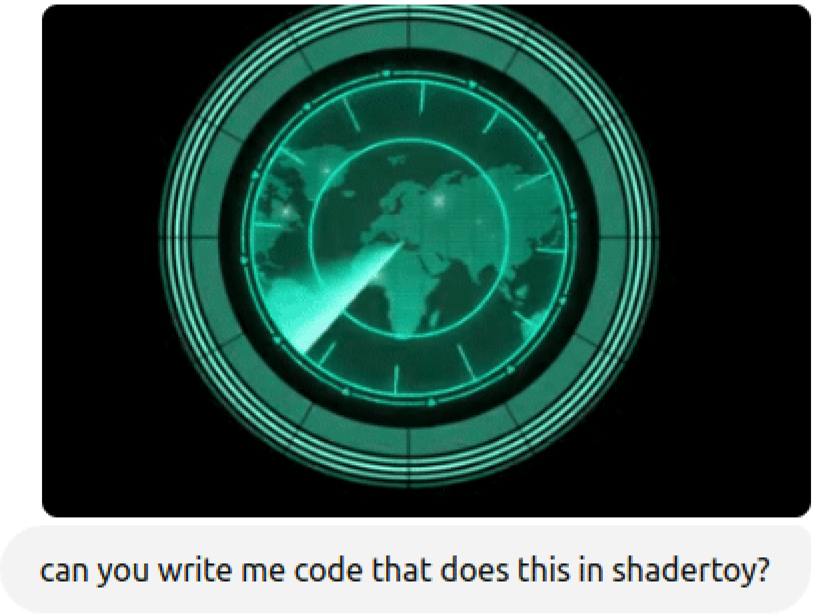
Getting Started

Lets ask Chatgpt by uploading an animated GIF of the radar done in a DCC to see if it can come up with the shader code itself.



Reference: Kinemaster Premium 3D Interface Radar 2 - Hologram Black Screen Ironman Hud Max Quality (4K) +Sound (youtube.com)

The following is the response on first try:



can you write me code that does this in shadertoy?

RESPONSE:

To create a radar-like effect as seen in the GIF, we need to simulate several elements in ShaderToy:

1. **Rotating Radar Sweep**: This is a rotating arc or line that moves around the radar.
2. **Circular Grid**: We can use concentric circles and radial lines to build the structure of the radar.
3. **World Map Projection**: We can omit the map for simplicity at first but simulate dots or blips on the radar.

Here's a basic approach to replicating this effect using ShaderToy:

Step-by-Step Breakdown:

1. **Circular Radar Grid**: We will use mathematical functions to create circles and radial lines.
2. **Radar Sweep**: A rotating arc that moves around the radar in sync with the `iTime` value in ShaderToy.
3. **Radar Dots/Blips**: Simulated points that blink or pulse as the radar sweep passes over them.

```
// ShaderToy - Radar Simulation

void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Convert fragCoord to normalized UV coordinates
    vec2 uv = fragCoord / iResolution.xy;
    uv = uv * 2.0 - 1.0;  // Center the UV coordinates (-1 to 1 range)
    uv.x *= iResolution.x / iResolution.y; // Correct for aspect ratio

    vec2 center = vec2(0.0, 0.0); // Radar center at (0, 0)
    float radius = length(uv); // Distance from center
    float angle = atan(uv.y, uv.x); // Angle in radians

    // Radar grid: Concentric circles
    float grid = smoothstep(0.02, 0.03, mod(radius * 10.0, 1.0));

    // Radar grid: Radial lines (create radial divisions)
    float radialLines = smoothstep(0.01, 0.015, abs(mod(angle + 3.14159, 3.14159 / 8.0) - (3.14159 / 16.0)));

    // Radar sweep (rotating arc)
    float sweepAngle = mod(iTime * 1.5, 6.28318); // Rotates with time
    float sweep = smoothstep(0.03, 0.0, abs(angle - sweepAngle)) * smoothstep(0.5, 0.3, radius); // Controls width

    // Radar blips: Simulated dots
    float blip = smoothstep(0.02, 0.0, length(uv - vec2(sin(iTime), cos(iTime)) * 0.5));

    // Combine the elements
    vec3 color = vec3(0.0);
    color += vec3(0.2, 0.8, 0.4) * (grid + radialLines); // Radar grid color
    color += vec3(0.0, 1.0, 0.5) * sweep; // Radar sweep color
    color += vec3(1.0, 0.2, 0.2) * blip; // Blip color

    // Output final color
    fragColor = vec4(color, 1.0);
}
```

Explanation:

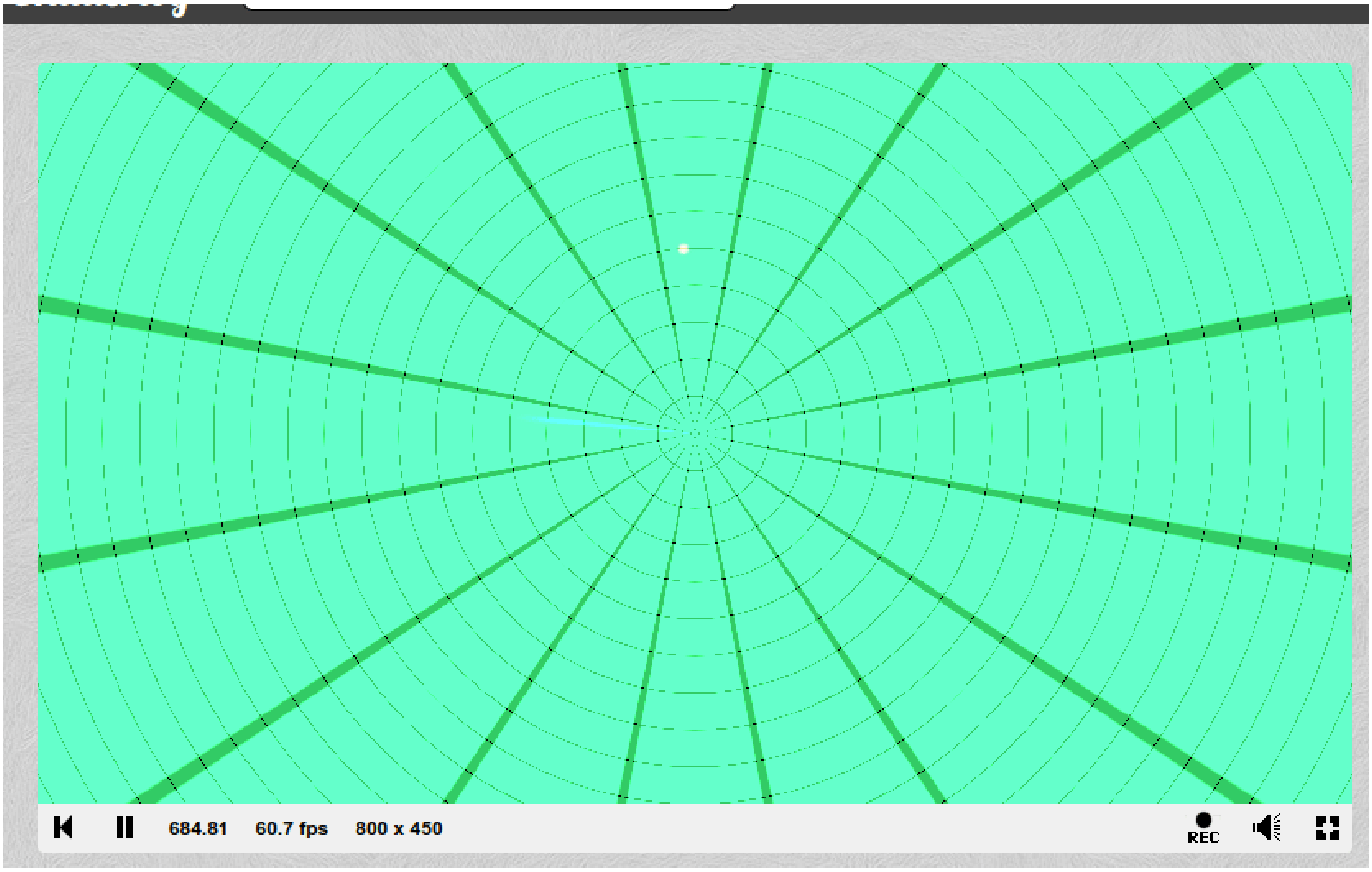
- **Radar Grid**: We use concentric circles and radial lines to build the background grid. The circles are created using the `radius` and `smoothstep()` functions, while the radial lines are

based on the `angle` and `mod()` function.
- **Radar Sweep**: A rotating arc is created by calculating the difference between the fragment’s angle and the sweep angle (`sweepAngle`). The arc is controlled by adjusting the smoothstep function to define its width and behavior.
- **Blips**: A simple radar blip is simulated by placing a dot at a certain location and making it pulse or blink when the radar sweep passes over it. The location is time-dependent here (sine and cosine functions), but you can add more blips by introducing random or fixed positions.

Next Steps:

- You can further enhance the radar by adding multiple blips or implementing different dot patterns.
- To simulate the map, you could map a texture or create procedural details like continent outlines. However, this would require more complexity.

You can paste this code directly into ShaderToy to test and see the rotating radar sweep with blips and a grid.

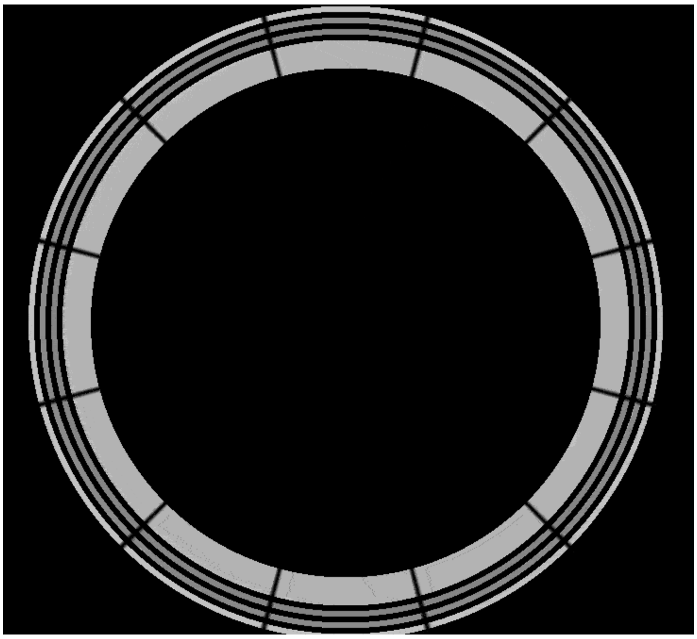


Fi

CURRENT WIP

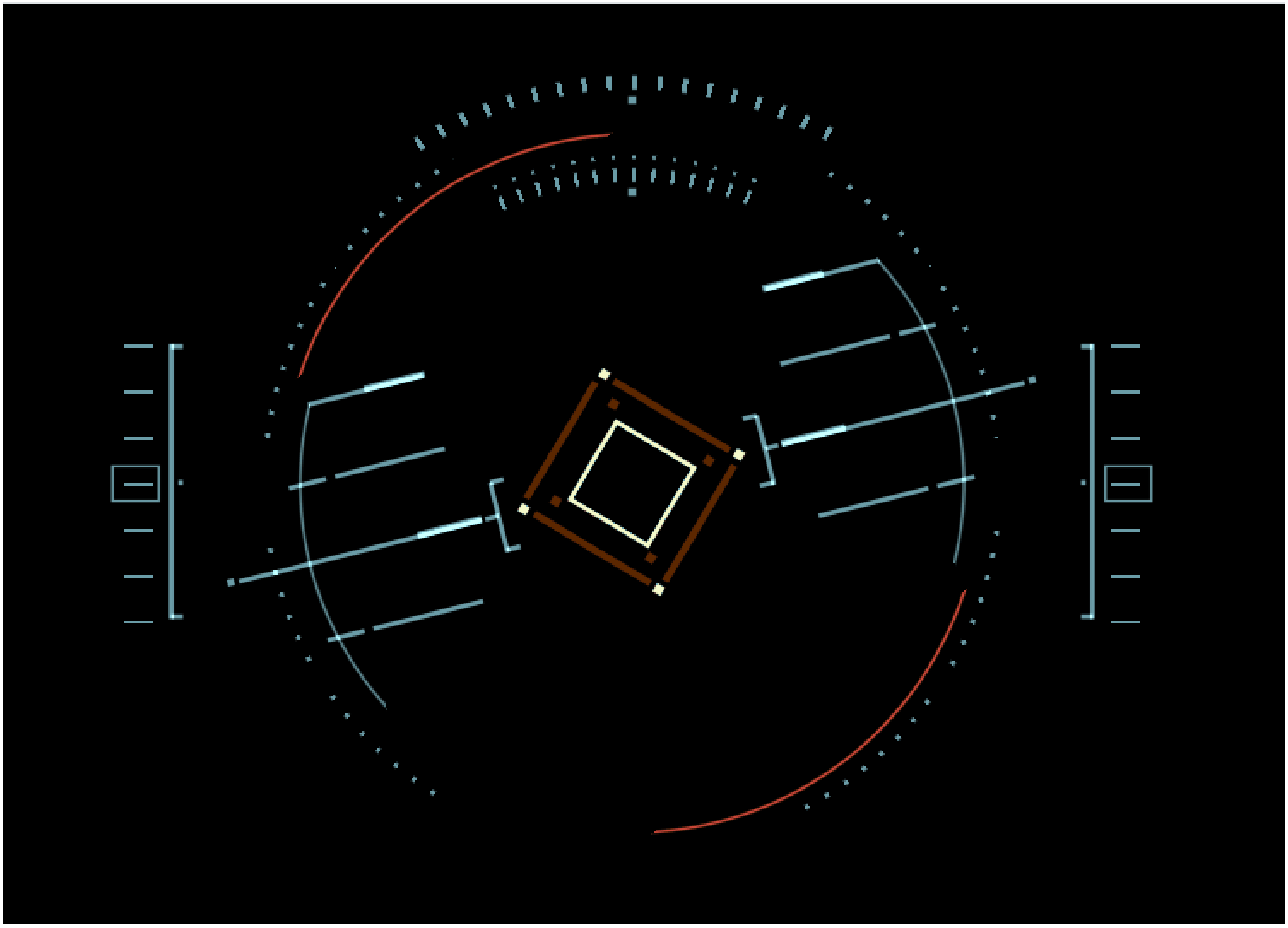
https://www.shadertoy.com/view/43lyD8

```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Set the center of the rings
    vec2 center = iResolution.xy * 0.5;

    // Define base radius and width for the big ring
    float bigRadius = 0.25 * iResolution.x;
    float bigRingWidth = 0.025 * iResolution.x;

    // Parameters for smaller rings
    float smallRadius = 0.26 * iResolution.x;
    float smallRingWidth = 0.005 * iResolution.x;

    // Parameters for the small white ring inside all rings
    float innerRadius = 0.12 * iResolution.x;
    float innerRingWidth = 0.005 * iResolution.x;

    // Calculate the distance from the current pixel to the center
    vec2 pos = fragCoord - center;
    float dist = length(pos);
    float angle = atan(pos.y, pos.x);

    // Normalize angle to [0, 1] range and divide into 12 segments
    angle = mod(angle + 2.0 * 3.14159265359, 2.0 * 3.14159265359) / (2.0 * 3.14159265359); // Normalize to [0, 1]
    float segment = angle * 12.0;  // 12 segments

    // Create transparent lines between the 12 segments
    float lineWidth = 0.03; // Width of the dividing lines
    float lines = smoothstep(0.0, lineWidth, abs(fract(segment) - 0.5));

    // Create the sequential blinking effect
    float blinkSpeed = 7.5; // Adjust speed of blinking
    float phaseShift = 1.0; // Time offset for sequential blinking
    // Blinking for the big ring (inner ring) starts first
    float bigRingBlink = 0.5 + 0.5 * sin(iTime * blinkSpeed);

    // Blinking for the smaller rings starts after the inner ring, with a phase shift
    float smallRing1Blink = 0.5 + 0.5 * sin(iTime * blinkSpeed - phaseShift);
    float smallRing2Blink = 0.5 + 0.5 * sin(iTime * blinkSpeed - 2.0 * phaseShift);
    float smallRing3Blink = 0.5 + 0.5 * sin(iTime * blinkSpeed - 3.0 * phaseShift);

    // Define the ring color (blinking between gray and white) for each ring
    vec3 bigRingColor = mix(vec3(0.5), vec3(1.0), bigRingBlink);
    vec3 smallRing1Color = mix(vec3(0.5), vec3(1.0), smallRing1Blink);
    vec3 smallRing2Color = mix(vec3(0.5), vec3(1.0), smallRing2Blink);
    vec3 smallRing3Color = mix(vec3(0.5), vec3(1.0), smallRing3Blink);

    // Define a static white color for the small inner ring
    vec3 innerRingColor = vec3(1.0, 1.0, 1.0);

    // Create the big central ring with hard edges and transparent lines
    float bigRing = step(bigRadius - bigRingWidth, dist) - step(bigRadius, dist);
    bigRing *= lines;

    // Create the three smaller rings with hard edges and transparent lines
    float smallRings = 0.0;

    // Small ring 1
    float smallDist = length(fragCoord - center);
    float smallRing1 = step(smallRadius - smallRingWidth, smallDist) - step(smallRadius, smallDist);
    smallRing1 *= lines;

    // Small ring 2
    float smallRing2 = step(0.27 * iResolution.x - smallRingWidth, smallDist) - step(0.27 * iResolution.x, smallDist);
    smallRing2 *= lines;

    // Small ring 3
    float smallRing3 = step(0.28 * iResolution.x - smallRingWidth, smallDist) - step(0.28 * iResolution.x, smallDist);
    smallRing3 *= lines;

    // Combine all small rings
    smallRings = smallRing1 + smallRing2 + smallRing3;

    // Create the small inner white ring
    float innerRing = step(innerRadius - innerRingWidth, dist) - step(innerRadius, dist);

    // Set the fragment color using the combined rings, with individual blinking for each ring
    vec3 finalColor = bigRingColor * bigRing + smallRing1Color * smallRing1 + smallRing2Color * smallRing2 + smallRing3Color * smallRing3 + innerRingColor * innerRing;
    fragColor = vec4(finalColor, 1.0);
}
```



Additional Reference

Checked to see other HUDS on Shadertoy, this is the only other good one.



```
float getBox(vec2 st, float left, float bottom, float width, float height) {
    float sm = 0.002;
    float x_range = smoothstep(left - sm, left, st.x) - smoothstep(left + width, left + width + sm, st.x);
    float y_range = smoothstep(bottom - sm, bottom, st.y) - smoothstep(bottom + height,bottom + height + sm, st.y);

    return x_range * y_range;
}

float getCircle(vec2 st, vec2 center, float radius, float thickness, float sm) {
    float distance = distance(st, center);
    return smoothstep(radius, radius + sm, distance) - smoothstep(radius + thickness, radius + thickness + sm, distance);
}

float getDottedCircle(vec2 st, vec2 center, float radius, float thickness, float sm) {
    float distance = distance(st, center);
    float circle = smoothstep(radius, radius + sm, distance) - smoothstep(radius + thickness, radius + thickness
+ sm, distance);

    vec2 vector = center - st;
    float angle = atan(vector.y, vector.x);
    angle = ((angle * 0.5) + (3.14 * 0.5)) / 3.14;
    circle *= step(8., mod(floor(angle / 0.001), 10.0));

    return circle;
}

mat2 rotate(in float angle) {
    return mat2(
                cos(angle), -sin(angle),
                sin(angle), cos(angle)
    □□);
}

vec2 getRotation(vec2 st, vec2 origin, float angle) {
    vec2 rotatedCoord = st - vec2(origin);
    rotatedCoord = rotate(angle) * rotatedCoord;
    rotatedCoord += vec2(origin);

    return rotatedCoord;
}

mat2 scale(in float x, in float y) {
    return mat2(
        □□x, 0.0,
                0.0, y
        □);
}

vec2 getScaling(vec2 st, vec2 origin, float x, float y) {
    vec2 scaledCoord = st - vec2(origin);
    scaledCoord = scale(x, y) * scaledCoord;
    scaledCoord += vec2(origin);

    return scaledCoord;
}
```

```
float getInnetDial(vec2 st, float center, float radius) {
	float lineThickness = 0.002;
    float sm = 0.003;

    float circle2 = getCircle(st, vec2(center), radius - 0.015, lineThickness, sm);
    circle2 *= step(st.x, -0.254);

    float circle2dash1 = getBox(st, -0.254, 0.134, 0.100, lineThickness);
    circle2 += circle2dash1;
    float circle2dash1_2 = getBox(st, -0.204, 0.134, 0.052, lineThickness * 2.);
    circle2 += circle2dash1_2;
    float circle2dash2 = getBox(st, -0.288, 0.068, 0.031, lineThickness);
    circle2 += circle2dash2;
    float circle2dash2_2 = getBox(st, -0.247, 0.068, 0.096, lineThickness);
    circle2 += circle2dash2_2;
    float circle2dash3 = getBox(st, -0.350, 0.0, 0.215, lineThickness);
    circle2 += circle2dash3;
    float circle2dash3_2 = getBox(st, -0.19, 0.0, 0.055, lineThickness * 2.);
    circle2 += circle2dash3_2;
    float circle2dash3_3 = getBox(st, -0.36, 0.0, lineThickness * 2., lineThickness * 2.);
    circle2 += circle2dash3_3;
    float circle2dash4 = getBox(st, -0.288, -0.068, 0.031, lineThickness);
    circle2 += circle2dash4;
    float circle2dash4_2 = getBox(st, -0.247, -0.068, 0.096, lineThickness);
    circle2 += circle2dash4_2;

    float circle2dash5 = getBox(st, -0.13, 0.0, 0.01, lineThickness);
    circle2 += circle2dash5;
    float circle2dash5_1 = getBox(st, -0.12, -0.03, lineThickness, 0.06);
    circle2 += circle2dash5_1;
    float circle2dash5_2 = getBox(st, -0.118, 0.03, 0.01, lineThickness);
    circle2 += circle2dash5_2;
    float circle2dash5_3 = getBox(st, -0.118, -0.03, 0.01, lineThickness);
    circle2 += circle2dash5_3;

    return circle2;
}

float getSideLine(vec2 st, float left, float top, float lineThickness) {
    float lineEdgeLength = 0.008;
    float sideLine = getBox(st, left, top, lineThickness, 0.236);
    sideLine += getBox(st, left + lineThickness, top + 0.234, lineEdgeLength, lineThickness);
    sideLine += getBox(st, left + lineThickness, top, lineEdgeLength, lineThickness);
    sideLine += getBox(st, left + lineThickness + 0.006, 0.0, lineThickness, lineThickness);

    return sideLine;
}

void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
	vec2 st = fragCoord.xy / iResolution.xy;
    st -= 0.5;
    st.x *= iResolution.x/iResolution.y;
	fragColor = vec4(st,0.5+0.5*sin(iTime),1.0);

    float center = 0.0;
    float radius = 0.3;
    float sm = 0.003;
    float lineThickness = 0.002;
    float angle = atan(st.y, st.x);
    angle = (angle + 3.1416) / (2.0 * 3.1416);

    vec3 color = vec3(0.);

    st = getRotation(st, vec2(center), -iTime);
    float circle1 = getCircle(st, vec2(0.), radius, lineThickness, sm);
    circle1 *= (step(st.x, -0.25) + step(0.25, st.x));
    st = getRotation(st, vec2(center), iTime);



    float dottedCircle = getDottedCircle(st, vec2(center), radius + 0.015, lineThickness + 0.002, sm);
    dottedCircle *= (
        (step(0.02, angle) - step(0.08, angle)) +
        (step(0.09, angle) - step(0.16, angle)) +
        (step(0.32, angle) - step(0.4, angle)) +
        (step(0.41, angle) - step(0.48, angle)) +

        (step(0.52, angle) - step(0.59, angle)) +
        (step(0.6, angle) - step(0.66, angle)) +
    (step(0.83, angle) - step(0.89, angle)) +
    (step(0.9, angle) - step(0.98, angle))
);

float timeFactor = 3.14 * 0.2 * cos(iTime * 0.5);
st = getRotation(st, vec2(center), timeFactor);
float dial1 = getDottedCircle(st, vec2(center), radius + 0.04, lineThickness + 0.01, sm);

float dial2 = getDottedCircle(st, vec2(center), radius + -0.04, lineThickness + 0.01, sm);
dial2 += getDottedCircle(st, vec2(center), radius + -0.02, lineThickness + 0.001, sm);
st = getRotation(st, vec2(center), -timeFactor);


dial1 *= (step(0.662, angle) - step(0.84, angle));
dial2 *= (step(0.682, angle) - step(0.82, angle));


float sideLine = getSideLine(st, -0.4, -0.116, 0.002);
st = getScaling(st, vec2(0.0), -1.0, 1.0);
sideLine += getSideLine(st, -0.4, -0.116, 0.002);
st = getScaling(st, vec2(0.0), -1.0, 1.0);

st = getRotation(st, vec2(center), 3.14 * 0.1 * sin(iTime));

float innerDial = getInnetDial(st, center, radius);
st = getScaling(st, vec2(0.0), -1.0, 1.0);
innerDial += getInnetDial(st, center, radius);
st = getScaling(st, vec2(0.0), -1.0, 1.0);


st = getRotation(st, vec2(center), 3.14 / 4.);
float box = getBox(st, -0.068, -0.057, 0.004, 0.115);
box += getBox(st, 0.066, -0.057, 0.004, 0.115);
box += getBox(st, -0.057, 0.070, 0.115, 0.004);
box += getBox(st, -0.057, -0.070, 0.115, 0.004);
box -= getBox(st, -0.036, -0.036, 0.072, 0.072);

box += getBox(st, -0.05, -0.05, 0.006, 0.006);
box += getBox(st, 0.048, -0.05, 0.006, 0.006);
box += getBox(st, 0.048, 0.046, 0.006, 0.006);
box += getBox(st, -0.05, 0.046, 0.006, 0.006);

float innerBox = getBox(st, -0.04, -0.04, 0.08, 0.08);
innerBox -= getBox(st, -0.036, -0.036, 0.072, 0.072);
innerBox += getBox(st, -0.07, -0.07, 0.006, 0.006);
innerBox += getBox(st, 0.066, -0.07, 0.006, 0.006);
innerBox += getBox(st, 0.066, 0.066, 0.006, 0.006);
innerBox += getBox(st, -0.07, 0.066, 0.006, 0.006);
st = getRotation(st, vec2(center), -3.14 / 4.);

st = getRotation(st, vec2(center), -3.14 * 0.1 * sin(iTime));

float sideMarks = step(18., mod(floor((st.y + 0.1 * sin(iTime)) / 0.002), 20.0));
sideMarks *= (
    (step(-0.44, st.x) - step(-0.415, st.x)) +
    (step(0.415, st.x) - step(0.44, st.x))
);
sideMarks *= (step(-0.12, st.y) - step(0.12, st.y));

float sideMarksBox = getBox(st, -0.45, -0.015, 0.04, 0.03);
sideMarksBox -= getBox(st, -0.448, -0.013, 0.036, 0.026);
st = getScaling(st, vec2(0.0), -1.0, 1.0);
sideMarksBox += getBox(st, -0.45, -0.015, 0.04, 0.03);
sideMarksBox -= getBox(st, -0.448, -0.013, 0.036, 0.026);
st = getScaling(st, vec2(0.0), -1.0, 1.0);

sideMarksBox += getBox(st, -0.0025, 0.33, 0.005, 0.005);
sideMarksBox += getBox(st, -0.0025, 0.25, 0.005, 0.005);

box *= (0.2 + 0.8 * pow(abs(sin(iTime * 4.)), 2.));

color += vec3(1.000,0.345,0.287) * circle1;
color += vec3(0.39,0.61,0.65) * dottedCircle;
color += vec3(0.39,0.61,0.65) * dial1;
color += vec3(0.39,0.61,0.65) * dial2;
color += vec3(0.39,0.61,0.65) * innerDial;
color += vec3(0.39,0.61,0.65) * sideLine;
color += vec3(0.39,0.61,0.65) * sideMarks;
color += vec3(0.39,0.61,0.65) * sideMarksBox;
color += vec3(0.995,0.425,0.003) * box;
```

```
    color += vec3(0.96, 0.98, 0.8) * innerBox;


    fragColor = vec4(color,1.0);
}
```

Tools

Shader Playground (timjones.io) - Decompiles shader code to Assembly

SHADERed - Free and open source shader editor

Revision #23
Created 9 September 2024 03:18:17 by victor
Updated 12 September 2024 23:03:32 by victor