```
<!DOCTYPE html>
<!-- saved from url=(0062)https://flippant.eu/wp-content/uploads/2025/05/shuffleBox.html
-->
<html lang="nl"><head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Canvas Vierkanten Animatie</title>
    <style>
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
            background-color: lightblue;
        }

        .container {
            position: relative;
            width: 890px;
            height: 890px;
            display: flex;
            justify-content: center;
            align-items: center;
        }

        .black-box {
            position: absolute;
            width: 890px;
            height: 890px;
            background-color: black;
        }

        canvas {
            position: absolute;
            width: 890px;
            height: 890px;
        }
    </style>
</head>

<body>
    <div class="container">
        <div class="black-box"></div>
        <canvas id="myCanvas" width="890" height="890"></canvas>
    </div>

    <script>
        const canvas = document.getElementById("myCanvas");
        const ctx = canvas.getContext("2d");

        let baseSize = 800;
        let maxSize = 890;
        let originalSizes = [];
        let sizes = [];
        let time = 0;
        let speed = 0.0036;
        let scaleFactor = maxSize / baseSize;
        let rotationTime = 1.33;
        let rotationDuration = 0.26;
        let rotationAngle = 0;

        // Set up the sizes for all squares
        for (let i = 0; i < 9; i++) {
            originalSizes.push(baseSize);
            sizes.push(baseSize);
            baseSize *= 0.63;
```

```
67          }
68
69          function waveFunction(t) {
70              return (Math.sin(t * Math.PI * 2) * 0.5) + 1;
71          }
72
73          function transitionEffect(factor, duration, offset) {
74              return factor + Math.sin((time + offset) * Math.PI * 2 / duration) * 0.1;
75          }
76
77          // Function to calculate the scale factor to keep the largest square inside the
            canvas
78          function calculateMaxScaleForRotation(size) {
79              let halfDiagonal = size / Math.sqrt(2);
80              let maxAllowedScale = Math.min(canvas.width / (halfDiagonal * 2),
                canvas.height / (halfDiagonal * 2));
81              return maxAllowedScale;
82          }
83
84          // Function to generate continuous random transparency based on time
85          function getRandomTransparency(t) {
86              return Math.abs(Math.sin(t * Math.PI * 2 * (17 / 1.43))) * 0.8 + 0.2;
87          }
88
89          // Function to check if the square is large enough for lines to connect
90          function isLargeEnough(size) {
91              return size > 55;  // Check if the size is larger than the margin (55px)
92          }
93
94          function animateSquares() {
95              time += speed;
96              let waveValue = waveFunction(time);
97              let rotationProgress = (time % rotationTime) / rotationDuration;
98
99              if (rotationProgress < Math.PI) {
100                 rotationAngle += Math.sin(rotationProgress * Math.PI) / Math.tan(Math.PI
                    * rotationProgress);
101             } else {
102                 rotationAngle += Math.sin(waveValue / 0.000037) * -Math.cos(waveValue /
                    0.000037) / Math.PI;
103             }
104
105             // Kies een willekeurige waarde voor i tussen 2 en 6
106             let randomIndex = Math.floor(Math.random() * (6 - 2 + 1)) + 2;  //
                Willekeurige index tussen 2 en 6
107
108             // Itereer over de vierkanten en pas de rotatie toe op de geselecteerde
                vierkant
109             for (let i = 0; i < sizes.length; i++) {
110                 let scaleModifier = 1;
111                 if (i > 0 && i % 2 === 0) {
112                     scaleModifier = transitionEffect(-0.83, 0.057, i) + Math.sin(Math.PI
                        / (waveValue * (time / (Math.PI + waveValue / (speed - waveValue *
                        time)) * i / time + speed))) * Math.PI / (i + i);
113                 } else if (i > 0) {
114                     scaleModifier = -transitionEffect(1.53, -0.084, i) / Math.cos(
                        Math.sin(Math.PI / (waveValue * (time / (Math.PI - waveValue / (speed
                         - waveValue * time)) * i / time + speed))));
115                 }
116
117                 scaleModifier += waveValue / (i + 1);
118
119                 if (i === 0) {
120                     sizes[i] = originalSizes[i] * (1 + (waveValue - 1) * (scaleFactor -
                        1.3));
121                 } else if (i % 2 === 0) {
122                     sizes[i] = originalSizes[i] * (1 - (waveValue - 1) * 1.5 *
```

```
                    scaleModifier);
123                 } else {
124                     sizes[i] = originalSizes[i] * (1 + (waveValue - 1) * 1.3 *
                        scaleModifier);
125                 }
126             }
127
128             // Wis het canvas voordat we tekenen
129             ctx.clearRect(0, 0, canvas.width, canvas.height);
130
131             // Teken zowel het eerste vierkant als het willekeurige vierkant
132             drawSquares(0);
133             drawSquares(randomIndex);  // Geef de willekeurige index door aan de
                drawSquares functie
134
135             requestAnimationFrame(animateSquares);
136         }
137
138         function drawLine(x1, y1, x2, y2, color) {
139             ctx.beginPath();
140             ctx.moveTo(x1, y1);
141             ctx.lineTo(x2, y2);
142             ctx.strokeStyle = color;
143             ctx.stroke();
144         }
145
146         function drawSquares(index) {
147             let centerX = canvas.width / 2;
148             let centerY = canvas.height / 2;
149
150             ctx.lineWidth = 1;
151
152
153             let transparencyValue = getRandomTransparency(time);
154
155             let maxScale = calculateMaxScaleForRotation(sizes[0]);
156             sizes[0] *= maxScale;
157
158             // Kleur voor de lijnen op basis van index
159             let colors = [
160                 "rgb(255, 255, 255)",    // Blauw
161                 "rgb(0, 0, 0)",    // Donkerblauw
162                 "rgb(255, 255, 255)",    // Lichtblauw
163                 "rgb(0, 0, 0)",    // Blauw
164                 "rgb(255, 255, 255)",  // Blauw met een kleine rode tint
165                 "rgb(0, 0, 0)",   // Blauw met een beetje groen
166                 "rgb(255, 255, 255)",   // Blauw met een ander groen-blauw
167                 "rgb(0, 0, 0)",    // Zeer lichte blauwe tint
168                 "rgb(255, 255, 255)"    // Blauw met een grotere groene tint
169             ];
170
171             for (let i = 0; i < sizes.length; i++) {
172                 let size = sizes[i];
173                 let halfSize = size / 2;
174                 let x1 = centerX - halfSize;
175                 let y1 = centerY - halfSize;
176                 let x2 = centerX + halfSize;
177                 let y2 = centerY + halfSize;
178
179
180
181                 // Als we het vierkant willen roteren, doe dit voor het geselecteerde
                    vierkant
182
183
184                 if (i === 0) {
185                     ctx.save();
```

```
                    ctx.translate(centerX, centerY);
                    ctx.rotate(Math.sin(rotationTime / 500) * Math.PI);  // Pas de
                    rotatie toe over de tijd
                    ctx.translate(-centerX, -centerY);

                    ctx.globalAlpha = transparencyValue;
                }


                if (i === index) {
                    ctx.save();
                    ctx.translate(centerX, centerY);
                    ctx.rotate(rotationAngle);  // Pas de rotatie toe over de tijd
                    ctx.translate(-centerX, -centerY);

                    ctx.globalAlpha = transparencyValue;
                }

                // Teken het vierkant met verschillende kleuren voor elke lijn
                drawLine(x1, y1, x2, y1, colors[i % colors.length]); // Top
                drawLine(x2, y1, x2, y2, colors[(i + 1) % colors.length]); // Right
                drawLine(x2, y2, x1, y2, colors[(i + 2) % colors.length]); // Bottom
                drawLine(x1, y2, x1, y1, colors[(i + 3) % colors.length]); // Left

                // Herstel de context voor de vierkanten die geroteerd zijn
                if (i === index) {  // Herstel de context na het roteren
                    ctx.restore();
                }
            }
        }

        drawSquares(0);
        requestAnimationFrame(animateSquares);
    </script>


</body></html>
```