```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
canvas {
    background-color: white;
}
</style>
</head>
<body id="b">
<noscript>Your browser does not have JavaScript enabled. Please enable or use
Firefox.</noscript>
<canvas id="c" width="500" height="500">Your browser does not support canvas. Please
upgrade or use Firefox.</canvas>
<script type="text/javascript">
var c = document.getElementById('c');
var ctx = c.getContext('2d');
ctx.translate(c.width/2,c.height/2);
ctx.scale(c.width/4,c.height/4);
function Point(x, y) {
    this.x = x;
    this.y = y;
}

// calculate product of two matrices
function matrixProduct(a, b) {
    var cw = b[0].length,
        ch = a.length,
        cn = a[0].length;
    if (cn != b.length) throw new Error("dimension mismatch");
    c = [];
    c.length = ch;
    for (var cr = 0; cr < ch; ++cr) {
        c[cr] = [];
        c[cr].length = cw;
        for (var cc = 0; cc < cw; ++cc) {
            c[cr][cc] = 0;
            for (var cj = 0; cj < cn; ++cj) {
                c[cr][cc] += a[cr][cj] * b[cj][cc];
            }
        }
    }
    return c;
}
// matrix product for the special case where p is a column vector expressed as an array
function transformProduct(m, p) {
    if (m.length !== p.length) throw new Error("dimension mismatch");
    var outp = []; outp.length = p.length;
    for (var r = 0; r < m.length; ++r) {
        outp[r] = 0;
        for (var j = 0; j < p.length; ++j) {
            outp[r] += m[r][j] * p[j];
        }
    }
    return outp;
}
// vector product axb
function vectorProduct(a, b) {
  return [a[1]*b[2]-a[2]*b[1], a[2]*b[0]-a[0]*b[2], a[0]*b[1]-a[1]*b[0]];
}
// scalar product a.b
function scalarProduct(a, b) {
  return a[0]*b[0] + a[1]*b[1] + a[2]*b[2];
}
// modulus of vector |a|
function modulus(a) {
  return Math.sqrt(a[0]*a[0] + a[1]*a[1] + a[2]*a[2]);
}

// current rotation
var rotx = 0, roty = 0;
// where the light is directed
var lightray = [-1, -1, 0];

// points on a cube
var points = [
[[-1],[-1],[-1]],
[[-1],[-1],[+1]],
[[-1],[+1],[-1]],
[[-1],[+1],[+1]],
[[+1],[-1],[-1]],
[[+1],[-1],[+1]],
[[+1],[+1],[-1]],
[[+1],[+1],[+1]]
];
// cube adjacency
var adjs = [[1,2], [4,1], [2,4]];
var faces = [];
var curi;
curi = 0;
// calculate cube faces
for (var a = 0; a < adjs.length; ++a)
    faces.push([curi, curi^adjs[a][0], curi^adjs[a][0]^adjs[a][1], curi^adjs[a][1]]);
curi = 7;
for (var a = 0; a < adjs.length; ++a)
    faces.push([curi, curi^adjs[a][1], curi^adjs[a][1]^adjs[a][0], curi^adjs[a][0]]);

var normals = []; normals.length = faces.length;

// calculate cube face normals
for (var f = 0; f < faces.length; ++f) {
    var facePoints = []; facePoints.length = 0;
    for (var p = 0; p < 4; ++p)
      facePoints[p] = points[faces[f][p]];
    var xp1 = [
      facePoints[1][0] - facePoints[0][0],
      facePoints[1][1] - facePoints[0][1],
      facePoints[1][2] - facePoints[0][2],
    ];
    var xp2 = [
      facePoints[3][0] - facePoints[0][0],
      facePoints[3][1] - facePoints[0][1],
      facePoints[3][2] - facePoints[0][2],
    ];
    normals[f] = vectorProduct(xp1, xp2);
}

setInterval(function () {
    // clear the background
    ctx.fillStyle = "rgba(255,255,255,1)";
    ctx.fillRect(-2,-2,4,4);

    // set up the rotation matrix
    var mx =
        [[1,0,0],
         [0, Math.cos(rotx), -Math.sin(rotx)],
         [0, Math.sin(rotx), Math.cos(rotx)]];
    var my =
        [[Math.cos(roty),0,Math.sin(roty)],
         [0, 1, 0],
         [-Math.sin(roty),0 , Math.cos(roty)]];
    var rotm = matrixProduct(mx, my);

    // make eight transformed cube vertices
    var pointsout = [];
    pointsout.length = 8;
    for (var i = 0; i < points.length; ++i) {
        pointsout[i] = transformProduct(rotm, points[i]);
        ctx.fillRect(pointsout[i][0][0], pointsout[i][1][0], 0.01, 0.01);
    }
    // draw the edges
    ctx.lineWidth = 0.01;
    for (var i = 0; i < points.length; ++i) {
        for (var j = 1; j <= 4; j *= 2) {
            ctx.beginPath();
            ctx.moveTo(pointsout[i][0], pointsout[i][1]);
            ctx.lineTo(pointsout[i^j][0], pointsout[i^j][1]);
            ctx.stroke();
        }
    }
    // find the closest point to the camera
    var besti, minz = 9999;
    for (var i = 0; i < points.length; ++i) {
        if (pointsout[i][2] < minz) {
            minz = pointsout[i][2];
            besti = i;
        }
    }
    // draw faces
    for (var f = 0; f < faces.length; ++f) {
        if (faces[f].indexOf(besti) === -1) continue;
        var facePoints = []; facePoints.length = 0;
        for (var p = 0; p < 4; ++p)
            facePoints[p] = pointsout[faces[f][p]];
        var xpnormal = transformProduct(rotm, normals[f]);

        // calculate lighting
        var angle = Math.acos(scalarProduct(xpnormal, lightray) /
(modulus(xpnormal)*modulus(lightray)));
        ctx.fillStyle = "rgb(" + Math.floor(angle*64 - 1) + "," + Math.floor(angle*64 - 1)
+ "," + Math.floor(angle*64-1) + ")";
        ctx.beginPath();
        ctx.moveTo(facePoints[0][0], facePoints[0][1]);
        ctx.lineTo(facePoints[1][0], facePoints[1][1]);
        ctx.lineTo(facePoints[2][0], facePoints[2][1]);
        ctx.lineTo(facePoints[3][0], facePoints[3][1]);
        ctx.fill();

        // draw on face ID and normal vector
        ctx.font = "0.2px monospace";
        ctx.fillStyle = "#000";
        ctx.fillText(f, (facePoints[0][0] + facePoints[2][0])/2, (facePoints[0][1] +
facePoints[2][1])/2);
        ctx.beginPath();
        ctx.moveTo((facePoints[0][0] + facePoints[2][0])/2, (facePoints[0][1] +
facePoints[2][1])/2);
        ctx.lineTo((facePoints[0][0] + facePoints[2][0])/2 + xpnormal[0], (facePoints[0][1]
+ facePoints[2][1])/2 + xpnormal[1]);
        ctx.stroke();
    }
    // change the rotation
    rotx += 0.09;
    roty += 0.1;
}, 50);

</script>
</body>
</html>
```