

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Maske – Farbwechsel + Volle Buttonfläche + Korrekte Rotation</title>
    <script src="https://cdn.jsdelivr.net/npm/p5@1.6.0/lib/p5.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/dat-gui/0.7.7/dat.gui.min.js"></script>
    <style>
      body { margin: 0; overflow: hidden; background: black; }
      canvas { display: block; position: absolute; z-index: -1; }
      #gui-container { position: absolute; left: 50px; bottom: 50px; z-index: 10;
pointer-events: auto; }
      #help-container {
        position: absolute; left: 50px; top: 50px;
        background: rgba(0,0,0,0.7); color: white; padding: 10px 20px;
        border: 1px solid red; border-radius: 10px; font-family: sans-serif; font-
size: 14px;
        z-index: 20; display: none;
      }
      .dg.main { background-color: #222 !important; color: red !important; }
      .dg .close-button { display: none !important; }
      .dg .slider-fg { background: red !important; }
      .dg .c .slider { background: #444 !important; }
      .dg .cr.number { border-left: 3px solid red !important; }
      .dg input[type="text"] { color: red !important; }
    </style>
  </head>
  <body>
    <div id="gui-container"></div>
    <div id="help-container">
      <strong>Shortcuts:</strong><br>
      [h] Show/Hide Handles and GUI<br>
      [r] Reset Workspace<br>
      [a] Fill Circle Red<br>
      [b] Fill Circle Green<br>
      [c] Fill Circle Blue<br>
      [d] Fill Circle Yellow<br>
      [0] Remove Circle Fill<br><br>
      <em>Click outside the area → Fullscreen toggle</em><br>
      <em>Click on Load A/B/C/D → Change color</em>
    </div>

    <script>
      let corners = [], circleCenter, interfaceElements = [], draggingIndex = -1,
draggingCircleCenter = false, draggingInterfaceIndex = -1;
      let showHandles = false, circleFillColor = null;
      const cornerRadius = 25, interfaceSize = { w: 240, h: 80 };
      let params = { Diameter: 400 }, gui;

      function setup() {
        createCanvas(windowWidth, windowHeight);
        resetCorners(); resetCircleCenter(); setupInterfaceElements();
        gui = new dat.GUI({ autoPlace: false });
      }
    </script>
  </body>
</html>
```

```
document.getElementById('gui-container').appendChild(gui.domElement);
gui.add(params, 'Diameter', 100, 1000).step(1).name('Circle Diameter');
document.getElementById('gui-container').style.display = 'none';
}

function draw() {
  background(0);
  fill(50); noStroke();
  beginShape(); for (let pt of corners) vertex(pt.x, pt.y); endShape(CLOSE);
  if (circleFillColor) fill(circleFillColor); else noFill();
  stroke(255); strokeWeight(1);
  ellipse(circleCenter.x, circleCenter.y, params.Diameter);
  noStroke();
  rectMode(CENTER); textAlign(CENTER, CENTER); textSize(16);
  for (let el of interfaceElements) {
    push();
    translate(el.position.x, el.position.y);
    rotate(radians(el.rotation));
    fill(100); rect(0, 0, interfaceSize.w, interfaceSize.h);
    fill(255); text(el.label, 0, 0);
    if (showHandles) { fill(255, 0, 0); ellipse(0, 0, cornerRadius); }
    pop();
  }
  if (showHandles) {
    fill(255, 0, 0);
    for (let pt of corners) ellipse(pt.x, pt.y, cornerRadius);
    ellipse(circleCenter.x, circleCenter.y, cornerRadius);
  }
}

function mousePressed() {
  const guiBounds = document.getElementById('gui-
container').getBoundingClientRect();
  if (mouseX >= guiBounds.left && mouseX <= guiBounds.left + guiBounds.width
&&
  mouseY >= guiBounds.top && mouseY <= guiBounds.top + guiBounds.height)
return;

  for (let i = 0; i < interfaceElements.length; i++) {
    let el = interfaceElements[i];
    let local = screenToLocal(mouseX, mouseY, el);
    if (local.x > -interfaceSize.w/2 && local.x < interfaceSize.w/2 &&
    local.y > -interfaceSize.h/2 && local.y < interfaceSize.h/2) {
      if (showHandles) {
        draggingInterfaceIndex = i;
      } else {
        if (el.label.includes("A")) circleFillColor = color(255, 0, 0);
        if (el.label.includes("B")) circleFillColor = color(0, 255, 0);
        if (el.label.includes("C")) circleFillColor = color(0, 0, 255);
        if (el.label.includes("D")) circleFillColor = color(255, 255, 0);
      }
    }
    return;
  }
}
```

```

    if (showHandles) {
      if (dist(mouseX, mouseY, circleCenter.x, circleCenter.y) < cornerRadius /
2) {
        draggingCircleCenter = true; return;
      }
      for (let i = 0; i < corners.length; i++) {
        if (dist(mouseX, mouseY, corners[i].x, corners[i].y) < cornerRadius /
2) {
          draggingIndex = i; return;
        }
      }
    }
    if (!insideWorkArea(mouseX, mouseY)) fullscreen(!fullscreen());
  }

function screenToLocal(mx, my, el) {
  let dx = mx - el.position.x;
  let dy = my - el.position.y;
  let angle = radians(-el.rotation);
  let localX = dx * cos(angle) - dy * sin(angle);
  let localY = dx * sin(angle) + dy * cos(angle);
  return {x: localX, y: localY};
}

function mouseDragged() {
  if (draggingIndex !== -1) {
    corners[draggingIndex].x = mouseX;
    corners[draggingIndex].y = mouseY;
  } else if (draggingCircleCenter) {
    circleCenter.x = mouseX;
    circleCenter.y = mouseY;
  } else if (draggingInterfaceIndex !== -1) {
    if (insideWorkArea(mouseX, mouseY)) {
      interfaceElements[draggingInterfaceIndex].position.x = mouseX;
      interfaceElements[draggingInterfaceIndex].position.y = mouseY;
    }
  }
}

function mouseReleased() {
  draggingIndex = -1; draggingCircleCenter = false; draggingInterfaceIndex =
-1;
}

function keyPressed() {
  if (key === 'h' || key === 'H') {
    showHandles = !showHandles;
    document.getElementById('gui-container').style.display = showHandles ?
'block' : 'none';
    document.getElementById('help-container').style.display = showHandles ?
'block' : 'none';
  }
  if (key === 'r' || key === 'R') {
    resetCorners(); resetCircleCenter(); setupInterfaceElements();
    params.Diameter = 400; gui.updateDisplay(); circleFillColor = null;
  }
}

```

```
    if (key === 'a' || key === 'A') circleFillColor = color(255, 0, 0);
    if (key === 'b' || key === 'B') circleFillColor = color(0, 255, 0);
    if (key === 'c' || key === 'C') circleFillColor = color(0, 0, 255);
    if (key === 'd' || key === 'D') circleFillColor = color(255, 255, 0);
    if (key === '0') circleFillColor = null;
}

function windowResized() { resizeCanvas(windowWidth, windowHeight); }

function resetCorners() {
  const margin = 20;
  const s = min(540, (min(windowWidth, windowHeight) - 2 * margin) / 2);
  const centerX = windowWidth / 2;
  const centerY = windowHeight / 2;
  corners = [
    createVector(centerX - s, centerY - s),
    createVector(centerX + s, centerY - s),
    createVector(centerX + s, centerY + s),
    createVector(centerX - s, centerY + s)
  ];
}

function resetCircleCenter() { circleCenter = createVector(windowWidth/2,
windowHeight/2); }

function setupInterfaceElements() {
  const cx = windowWidth / 2, cy = windowHeight / 2;
  interfaceElements = [
    { id: "topLeft", label: "Load A", rotation: 180, position:
createVector(cx-200, cy-200) },
    { id: "topRight", label: "Load B", rotation: 270, position:
createVector(cx+200, cy-200) },
    { id: "bottomRight", label: "Load C", rotation: 0, position:
createVector(cx+200, cy+200) },
    { id: "bottomLeft", label: "Load D", rotation: 90, position:
createVector(cx-200, cy+200) }
  ];
}

function insideWorkArea(x, y) { return collidePointPoly(x, y, corners); }

function collidePointPoly(px, py, vertices) {
  let collision = false;
  for (let current = 0; current < vertices.length; current++) {
    let next = (current + 1) % vertices.length;
    let vc = vertices[current];
    let vn = vertices[next];
    if (((vc.y > py && vn.y < py) || (vc.y < py && vn.y > py)) &&
        (px < (vn.x - vc.x) * (py - vc.y) / (vn.y - vc.y) + vc.x)) {
      collision = !collision;
    }
  }
  return collision;
}
```

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

From:

<https://wiki.ct-lab.info/> - Creative Technologies Lab | dokuWiki

Permanent link:

https://wiki.ct-lab.info/doku.php/extras:codikon:sonstiges:dome_projection_v01?rev=1751788124

Last update: **2025/07/06 07:48**

