Work Section (Interactive Laboratory)

Last Updated: January 13, 2026 **Related Docs:** ACADEMIC-JOURNEY-SECTION.md | ARCHITECTURE.md | MASTER-OVERVIEW.md

---

Table of Contents

1. Overview
2. Component Architecture
3. Cellular Metaphor System
4. Physics Simulation
5. OrganicBlob Membrane
6. Organelle Cards
7. Breathing Animation
8. Collision Detection
9. Expansion System
10. Mobile Responsiveness
11. Performance Optimizations

Overview

The Work Section transforms project showcases into a **living biological system**, where each project is an "organelle" floating inside a breathing cell membrane with realistic physics.

Design Philosophy

Cell Biology as Interface: Projects are organelles inside a cell, moving with physics-based motion, colliding naturally, and responding to user interaction.

Element	Biological Equivalent	Interactive Behavior
Organelles	Mitochondria, Ribosomes	Floating project cards with physics motion
Cell Membrane	Phospholipid Bilayer	Breathing SVG path with wave animation
Cytoplasm	Cell Interior	Physics simulation space
Collision	Organelle Interactions	Realistic bouncing and separation

Element	Biological Equivalent	Interactive Behavior
Expansion	Cell Signaling	Click organelle → Expands, pushes others away

Key Features

- **4 Work Items**: Research, education, automation, clinical projects
- **Real Physics**: Velocity, damping, collision detection, boundary checking
- **Breathing Membrane**: 10-second breath cycle (4s inhale, 6s exhale)
- **Traveling Wave**: Sinusoidal ripple around membrane (8s per rotation)
- **Interactive Expansion**: Click organelle → 2.5× larger, others pushed away
- **Theme-Aware Colors**: Each organelle type has unique color
- **Mobile Optimized**: Rounded square boundary instead of circle

---

Component Architecture

File Structure

```
src/
├── components/
│   └── work/
│       ├── WorkSection.jsx          # Main orchestrator (246 lines)
│       ├── OrganelleNew.jsx        # Individual project cards (171 lines)
│       ├── OrganicBlob.jsx          # Breathing membrane SVG (404 lines)
│       └── OrganellePhysics.js     # Physics simulation engine (520 lines)
└── Styles/
    └── Work.css                     # Section styles
```

Component Hierarchy

```
<WorkSection>
├── <div className="work-content-grid">
│   ├── <div ref={containerRef} className="work-cell">
│   │   └── <OrganicBlob>
│   │       ├── SVG Mask/Clip Path
│   │       │   ├── Cell Interior Path
│   │       │   ├── Outer Membrane Layer
│   │       │   ├── Inner Membrane Layer
│   │       │   ├── Specular Glow Overlay
```

```
|   |       |   └── Radial Flow Gradients
|   |       └── <OrganelleNew> × 4
|   |           ├── Title
|   |           ├── Icon (emoji or SVG)
|   |           ├── Close Button (when active)
|   |           └── View Project Link (when active)
|   └── <motion.div className="work-sidebar">
|       ├── Title: "Interactive Laboratory"
|       ├── Subtitle
|       └── Description
```

State Management

11 State Variables (in WorkSection):

```jsx
const [isMobile, setIsMobile] = useState(false);                    // Mobile detection (768px)
const [activeIndex, setActiveIndex] = useState(null);               // Which organelle is expanded
const [hoveredId, setHoveredId] = useState(null);                   // Which organelle is hovered
const [organellePositions, setOrganellePositions] = useState({});   // Physics-driven positions

// Refs for physics system
const containerRef = useRef(null);                                   // Container DOM element
const physicsManagerRef = useRef(null);                              // Physics manager instance
const resizeObserverRef = useRef(null);                              // Window resize observer
const animationFrameRef = useRef(null);                              // RAF loop ID
```

Physics State (in OrganellePhysicsManager):

```jsx
// Per organelle:
{
  id: "organelle-0",
  x: 0,               // X position (relative to center)
  y: 0,               // Y position (relative to center)
  velX: 1.5,          // X velocity
  velY: 1.5,          // Y velocity
  radius: 90,         // Collision radius (90px desktop, 35px mobile)
  isHovered: false,   // Hover state (slows organelle)
  isExpanded: false,  // Expansion state (stops movement, increases radius)
  originalRadius: 90  // Stored original radius (when expanded)
}
```

Membrane State (in OrganicBlob):

```
const [breathScale, setBreathScale] = useState(1.0);       // Breathing scale (0.986-1.147)
const [wavePhase, setWavePhase] = useState(0);             // Traveling wave phase (0-2π)
const [breathProgress, setBreathProgress] = useState(0);   // Breath progress (0-1)
```

Cellular Metaphor System

Work Items Data

4 Projects with biological classification:

```
const workItems = [
  {
    title: "Melanoma Genetics Workshop",
    description: "Interactive evaluation analysing dermatologists' genetics training outcomes.",
    technologies: "React, R, Statistical Analysis",
    role: "Research Data Scientist",
    link: "/projects/melanoma-workshop",
    icon: " ",
    organelleType: "education",     // Maps to CSS var(--organelle-education)
    fontStyle: "creative"
  },
  {
    title: "Hearing Loss & Gestational Diabetes Study",
    description: "Neonatal screening insights from 328,751 births...",
    technologies: "React, D3.js, Statistical Analysis",
    role: "Research Data Scientist",
    link: "/projects/hearing-loss-diabetes",
    icon: " ",
    organelleType: "research",      // Maps to CSS var(--organelle-research)
    fontStyle: "technical"
  },
  {
    title: "Automated PRS Booklet Tool",
    description: "Generates personalised melanoma PRS dossiers...",
    technologies: "Python, LaTeX",
    role: "Automation Engineer",
    link: "/work/prs-generator",
    icon: " ",
    organelleType: "automation",    // Maps to CSS var(--organelle-automation)
    fontStyle: "technical"
  },
  {
```

```
    title: "Clinical Calculator for DermClinic",
    description: "Point-of-care dermato-oncology dose calculator...",
    technologies: "React, Node",
    role: "Frontend Developer",
    link: "/work/dermcalc",
    icon: "  ",
    organelleType: "clinical",      // Maps to CSS var(--organelle-clinical)
    fontStyle: "technical"
  }
];
```

Organelle Type Colors

CSS Custom Properties (expected in CSS):

```
:root {
  --organelle-education: #9333ea;    /* Purple */
  --organelle-research: #0ea5e9;     /* Sky Blue */
  --organelle-automation: #10b981;   /* Emerald Green */
  --organelle-clinical: #f59e0b;     /* Amber */
  --organelle-bg: rgba(100, 100, 100, 0.3);  /* Fallback gray */
}
```

Visual Coding: - **Education** (Purple): Teaching and training projects - **Research** (Blue): Data analysis and scientific studies - **Automation** (Green): Tools and workflow systems - **Clinical** (Amber): Healthcare applications

---

Physics Simulation

OrganellePhysicsManager Class

Purpose: Manages real-time physics for all organelles using velocity-based Newtonian mechanics.

File: `/src/components/work/OrganellePhysics.js` (520 lines)

Initialization

```
const manager = new OrganellePhysicsManager(
  clientWidth,        // Container width in pixels
  clientHeight,       // Container height in pixels
```

```
  isMobile,           // Boolean: mobile vs desktop
  organelleRadius     // Organelle radius (90px desktop, 35px mobile)
);

// Add organelles
workItems.forEach((_, index) => manager.addOrganelle(`organelle-${index}`));

// Start physics loop
manager.start();
```

Physics Constants

```
this.damping = 0.995;               // Velocity decay per frame (0.5% loss)
this.bounceStrength = 0.8;          // Energy retention on collision (80%)
this.minSpeed = 2.0;                // Minimum pixels/frame (prevents stopping)
this.maxSpeed = 3.5;                // Maximum pixels/frame (prevents chaos)
this.separationForce = 0.3;         // Repulsion strength between organelles
this.breathingForceStrength = 0.15; // Membrane breathing pull strength (desktop)
```

Physics Update Loop

60 FPS Animation Loop:

```
start() {
  const loop = () => {
    if (!this.isRunning) return;

    this.update();  // Update all organelle positions
    this.animationFrame = requestAnimationFrame(loop);
  };

  this.animationFrame = requestAnimationFrame(loop);
}

update() {
  // 1. Apply breathing force (desktop only)
  if (!isMobile && currentBreathingVelocity !== 0) {
    for (const organelle of organelles) {
      applyBreathingForce(organelle);
    }
  }

  // 2. Update positions and velocities
  for (const organelle of organelles) {
```

```
    if (organelle.isExpanded) continue; // Expanded organelles don't move

    // Apply movement with hover slowdown
    const speedMultiplier = organelle.isHovered ? 0.1 : 1.0;
    organelle.x += organelle.velX * speedMultiplier;
    organelle.y += organelle.velY * speedMultiplier;

    // Apply damping
    organelle.velX *= damping;
    organelle.velY *= damping;

    // Random perturbations (2% chance per frame)
    if (Math.random() < 0.02) {
      const perturbStrength = 0.3;
      const perturbAngle = Math.random() * Math.PI * 2;
      organelle.velX += Math.cos(perturbAngle) * perturbStrength;
      organelle.velY += Math.sin(perturbAngle) * perturbStrength;
    }

    // Enforce speed limits
    enforceSpeedLimits(organelle);
  }

  // 3. Handle collisions
  handleOrganelleCollisions();

  // 4. Handle boundary collisions
  for (const organelle of organelles) {
    if (isMobile) {
      handleRoundedSquareBoundary(organelle);
    } else {
      handleEllipseBoundary(organelle);
    }
  }
}
```

Velocity Management

Minimum Speed Enforcement:

```
const speed = Math.sqrt(velX * velX + velY * velY);

if (speed < 0.0001) {
  // Generate fresh direction when stopped
  const angle = Math.random() * Math.PI * 2;
```

```
  organelle.velX = Math.cos(angle) * minSpeed;
  organelle.velY = Math.sin(angle) * minSpeed;
} else if (speed < minSpeed) {
  // Scale up to minimum speed
  const factor = minSpeed / speed;
  organelle.velX *= factor;
  organelle.velY *= factor;
}
```

Maximum Speed Clamping:

```
if (speed > maxSpeed) {
  const factor = maxSpeed / speed;
  organelle.velX *= factor;
  organelle.velY *= factor;
}
```

Why Speed Limits? - **Minimum**: Prevents organelles from appearing stationary (boring) - **Maximum**: Prevents chaotic motion (hard to track visually)

Random Perturbations

```
if (Math.random() < 0.02) { // 2% chance each frame (~1-2 times/second at 60fps)
  const perturbStrength = 0.3;
  const perturbAngle = Math.random() * Math.PI * 2;
  organelle.velX += Math.cos(perturbAngle) * perturbStrength;
  organelle.velY += Math.sin(perturbAngle) * perturbStrength;
}
```

Purpose: Prevents stable orbital patterns (organelles would otherwise settle into equilibrium).

OrganicBlob Membrane

Breathing Animation System

Purpose: SVG path that breathes like a living cell, expanding/contracting over 10-second cycle.

File: `/src/components/work/OrganicBlob.jsx` (404 lines)

Breath Cycle

10-Second Asymmetric Cycle:

```
const breathCycleDuration = 10000;  // 10 seconds total
const inhaleRatio = 0.4;            // 40% of cycle = 4 seconds inhale
const exhaleRatio = 0.6;            // 60% of cycle = 6 seconds exhale

const cyclePosition = (elapsed % breathCycleDuration) / breathCycleDuration; // 0 to 1

// Easing function for smooth transitions
const easeInOutSine = (t) => -(Math.cos(Math.PI * t) - 1) / 2;

let normalizedBreath;
if (cyclePosition < inhaleRatio) {
  // Inhale phase: 0 to 1 (faster - 4 seconds)
  const inhaleProgress = cyclePosition / inhaleRatio;
  normalizedBreath = easeInOutSine(inhaleProgress);
} else {
  // Exhale phase: 1 to 0 (slower - 6 seconds)
  const exhaleProgress = (cyclePosition - inhaleRatio) / exhaleRatio;
  normalizedBreath = 1 - easeInOutSine(exhaleProgress);
}

// Convert normalized breath (0-1) to radius
const minRadius = 37;  // Inner red circle (viewBox units)
const maxRadius = 43;  // Outer blue circle (viewBox units)
const currentRadius = minRadius + normalizedBreath * (maxRadius - minRadius);
const breathScale = currentRadius / 37.5; // Scale factor (0.986-1.147)
```

Why Asymmetric? - **Inhale (4s)**: Quick breath in (energetic) - **Exhale (6s)**: Slow breath out (calm) - **Effect**: More natural than symmetric sine wave

Traveling Wave

8-Second Rotation:

```
const waveCycleDuration = 8000; // 8 seconds per rotation
const currentWavePhase = (elapsed / waveCycleDuration) * Math.PI * 2; // 0 to 2π

// Wave amplitude modulation: stronger during inhale, subtler during exhale
const baseWaveAmplitude = 0.4; // ~1% of radius
const waveAmplitude = baseWaveAmplitude * (0.5 + breathProgress * 0.5);
```

```
// Apply wave to each point around circumference
for (let i = 0; i < numPoints; i++) {
  const angle = (i / numPoints) * Math.PI * 2;

  // Traveling wave: sinusoidal displacement
  const waveOffset = Math.sin(angle - wavePhase) * waveAmplitude;

  const finalRadius = radius + waveOffset;
  const finalX = centerX + Math.cos(angle) * finalRadius;
  const finalY = centerY + Math.sin(angle) * finalRadius;

  points.push({ x: finalX, y: finalY });
}
```

Effect: Ripple travels counterclockwise around membrane, like a wave in water.

Path Generation

Desktop: Organic Blob

```
function generateOrganicBlobPath(breathScale = 1.0, wavePhase = 0, breathProgress = 0) {
  const baseRadius = 37.5;
  const radius = baseRadius * breathScale;
  const numPoints = 10; // Fewer points for smooth curves

  const points = [];
  for (let i = 0; i < numPoints; i++) {
    const angle = (i / numPoints) * Math.PI * 2;
    const waveOffset = Math.sin(angle - wavePhase) * waveAmplitude;
    const finalRadius = radius + waveOffset;

    const finalX = centerX + Math.cos(angle) * finalRadius;
    const finalY = centerY + Math.sin(angle) * finalRadius;
    points.push({ x: finalX, y: finalY });
  }

  // Create smooth curved path with Bezier curves
  let path = `M ${points[0].x} ${points[0].y}`;

  for (let i = 0; i < points.length; i++) {
    const current = points[i];
    const next = points[(i + 1) % points.length];
    const prev = points[(i - 1 + points.length) % points.length];
    const after = points[(i + 2) % points.length];
```

```
    // Gentler curve tension for more fluid movement
    const tension = 0.18;
    const control1x = current.x + (next.x - prev.x) * tension;
    const control1y = current.y + (next.y - prev.y) * tension;
    const control2x = next.x - (after.x - current.x) * tension;
    const control2y = next.y - (after.y - current.y) * tension;

    path += ` C ${control1x} ${control1y}, ${control2x} ${control2y}, ${next.x} ${next.y}`;
  }

  path += ' Z';
  return path;
}
```

Mobile: Rounded Square

```
function generateRoundedSquarePath() {
  const borderRadius = 15; // Large radius for smooth corners
  const squareWidth = 85;  // 85% of viewBox
  const squareHeight = 85;

  const left = centerX - squareWidth / 2;
  const right = centerX + squareWidth / 2;
  const top = centerY - squareHeight / 2;
  const bottom = centerY + squareHeight / 2;

  // Static rounded rectangle path (no wobble)
  const path = `
    M ${left + borderRadius} ${top}
    L ${right - borderRadius} ${top}
    Q ${right} ${top} ${right} ${top + borderRadius}
    L ${right} ${bottom - borderRadius}
    Q ${right} ${bottom} ${right - borderRadius} ${bottom}
    L ${left + borderRadius} ${bottom}
    Q ${left} ${bottom} ${left} ${bottom - borderRadius}
    L ${left} ${top + borderRadius}
    Q ${left} ${top} ${left + borderRadius} ${top}
    Z
  `;

  return path;
}
```

Why Different on Mobile? - **Desktop**: Organic breathing circle (complex, engaging) - **Mobile**: Static rounded square (simpler, less CPU usage)

SVG Layers

7 Overlapping Paths (all use same `blobPath`):

```
<svg viewBox="0 0 100 100" preserveAspectRatio="none">
  <!-- 1. Cell Interior (solid fill) -->
  <path d={blobPath} fill="var(--cell-interior-bg)" />

  <!-- 2. Outer Membrane Layer (thick stroke) -->
  <path
    d={blobPath}
    fill="none"
    stroke="var(--cell-membrane-color)"
    strokeWidth={outerStrokeWidth}
  />

  <!-- 3. Inner Membrane Layer (lipid bilayer effect) -->
  <path
    d={blobPath}
    fill="none"
    stroke="var(--cell-membrane-inner)"
    strokeWidth={innerStrokeWidth}
    opacity="0.7"
    transform="scale(0.95)"
  />

  <!-- 4. Specular Glow (top-left light source) -->
  <path
    d={blobPath}
    fill="url(#membrane-glow)"
    style={{ mixBlendMode: 'soft-light' }}
  />

  <!-- 5. Primary Radial Flow Glow (breathing density gradient) -->
  <path
    d={blobPath}
    fill="url(#radial-flow-glow)"
    opacity={0.85 + breathProgress * 0.15}
    style={{ mixBlendMode: 'screen' }}
  />

  <!-- 6. Secondary Flow Layer (particle streaming effect) -->
  <path
    d={blobPath}
    fill="url(#radial-flow-secondary)"
    opacity={0.6}
```

```
    style={{ mixBlendMode: 'soft-light' }}
  />
</svg>
```

Radial Flow Density Effect

Inverts with Breathing:

```
// During contraction (breathProgress → 0): Center bright, edge dim (inward flow)
// During expansion (breathProgress → 1): Center dim, edge bright (outward flow)

const centerOpacity = 0.48 * (1 - breathProgress);  // Bright when contracted
const edgeOpacity = 0.48 * breathProgress;          // Bright when expanded
const midOpacity = (centerOpacity + edgeOpacity) / 2;

<radialGradient id="radial-flow-glow" cx="50%" cy="50%" r="65%">
  <stop offset="0%" stopColor="var(--cell-membrane-color)" stopOpacity={centerOpacity}/>
  <stop offset="40%" stopColor="var(--cell-membrane-color)" stopOpacity={midOpacity * 0.7}/>
  <stop offset="85%" stopColor="var(--cell-membrane-color)" stopOpacity={edgeOpacity * 1.1}/>
  <stop offset="100%" stopColor="transparent" stopOpacity="0"/>
</radialGradient>
```

Effect: Creates visual sensation of cytoplasm flowing inward/outward as cell breathes.

Organelle Cards

OrganelleNew Component

Purpose: Individual project card with physics-driven position and interactive expansion.

File: `/src/components/work/OrganelleNew.jsx` (171 lines)

Props Interface

```
<OrganelleNew
  id="organelle-0"                      // Unique ID
  title="Melanoma Genetics Workshop"   // Project title
  description="Interactive evaluation..." // Project description
  icon="□ "                             // Emoji or SVG path
```

```
  link="/projects/melanoma-workshop"   // Route link
  technologies="React, R, Statistical Analysis"
  role="Research Data Scientist"
  organelleType="education"             // Maps to CSS color
  fontStyle="creative"                  // "creative" or "technical"
  size={170}                            // Size in pixels (desktop: 170, mobile: 70)
  active={false}                        // Expanded state
  dimmed={false}                        // Dimmed when another is expanded
  onOpen={() => handleOpen(0)}          // Click handler
  onClose={handleClose}                 // Close handler
  position={{ x: 0, y: 0 }}             // Physics-driven position
  hovered={false}                       // Hover state
  onHoverStart={() => handleHoverChange("organelle-0", true)}
  onHoverEnd={() => handleHoverChange("organelle-0", false)}
  isMobile={false}
/>
```

States and Animations

Three States:

1. **Default** (dimmed: false, active: false)
   - Circular shape (`borderRadius: "50%"`)
   - Size: 170px × 170px (desktop) or 70px × 70px (mobile)
   - Opacity: 1.0
   - Scale: 1.0
   - Position: Physics-driven (x, y)
2. **Dimmed** (dimmed: true, active: false)
   - Same as default but:
   - Opacity: 0.3
   - Scale: 0.5
   - Pointer events: none (not clickable)
3. **Active** (dimmed: false, active: true)
   - Desktop: Rounded rectangle (`borderRadius: "20px"`)
   - Mobile: Stays circular (`borderRadius: "50%"`)
   - Size: 425px × 425px (desktop) or 175px × 175px (mobile) [2.5× larger]
   - Opacity: 1.0
   - Scale: 1.0
   - Position:
     - Desktop: Moves to center (x: 0, y: 0)
     - Mobile: Stays in place (x: position.x, y: position.y)

Expansion Animation

Desktop Expansion:

```
controls.start({
  width: size * 2.5,              // 170 → 425px
  height: size * 2.5,
  marginLeft: -(size * 2.5) / 2,  // Center alignment
  marginTop: -(size * 2.5) / 2,
  borderRadius: "20px",           // Rounded rectangle
  opacity: 1,
  scale: 1,
  x: 0,                           // Move to center
  y: 0,
  transition: {
    type: "spring",
    stiffness: 150,
    damping: 22,
    duration: 0.5                 // Max 500ms
  }
});
```

Mobile Expansion:

```
controls.start({
  width: size * 2.5,              // 70 → 175px
  height: size * 2.5,
  marginLeft: -(size * 2.5) / 2,
  marginTop: -(size * 2.5) / 2,
  borderRadius: "50%",            // Stay circular
  opacity: 1,
  scale: 1,
  x: position.x,                  // Stay in current position
  y: position.y,
  transition: {
    type: "spring",
    stiffness: 200,
    damping: 20,
    duration: 0.4                 // Max 400ms
  }
});
```

Why Different? - **Desktop**: Move to center for focus (like modal) - **Mobile**: Stay in place (no room to move in small viewport)

Collapse Animation

```
controls.start({
  width: size,
  height: size,
  marginLeft: -size / 2,
  marginTop: -size / 2,
  borderRadius: "50%",              // Back to circle
  scale: dimmed ? 0.5 : 1,
  opacity: dimmed ? 0.3 : 1,
  x: position.x,                    // Back to physics position
  y: position.y,
  transition: {
    type: "spring",
    stiffness: 180,
    damping: 25,
    duration: 0.4
  }
});
```

Content Layout

Default State: - Title (centered, visible) - Icon (emoji or image, centered) - No description (minimalist)

Active State: - Title (top) - Icon (center) - Close Button (× top-right) - "View Project →" Link (bottom)

Breathing Animation

Membrane-Physics Synchronization

OrganicBlob notifies Physics Manager of radius changes:

```
// In OrganicBlob.jsx
useEffect(() => {
  const animate = () => {
    // ... calculate currentRadius (37–43 viewBox units)

    // Notify parent of current membrane radius
    if (onMembraneRadiusChange) {
      onMembraneRadiusChange(currentRadius);
```

```
    }
  };

  animate();
}, [isMobile]);
```

Physics Manager updates boundary:

```
// In OrganellePhysics.js
updateMembraneBoundary(membraneRadiusInViewBox) {
  // Calculate breathing velocity (rate of change)
  const radiusChange = membraneRadiusInViewBox - this.currentMembraneRadius;
  this.currentBreathingVelocity = radiusChange;

  // Update ellipse boundary
  this.currentMembraneRadius = membraneRadiusInViewBox;
  const radiusRatio = membraneRadiusInViewBox / 100;
  this.ellipseRadiusX = this.containerWidth * radiusRatio;
  this.ellipseRadiusY = this.containerHeight * radiusRatio;
}
```

Breathing Force on Organelles

Radial Push/Pull:

```
if (!isMobile && this.currentBreathingVelocity !== 0) {
  for (const organelle of this.organelles) {
    if (organelle.isExpanded) continue;

    // Calculate radial direction from center to organelle
    const distanceFromCenter = Math.sqrt(organelle.x * organelle.x + organelle.y * organelle.y);

    if (distanceFromCenter > 1) {
      const radialDirectionX = organelle.x / distanceFromCenter;
      const radialDirectionY = organelle.y / distanceFromCenter;

      // Apply force proportional to breathing velocity
      // Positive velocity = expanding (push outward)
      // Negative velocity = contracting (pull inward)
      const force = this.currentBreathingVelocity * this.breathingForceStrength;

      organelle.velX += radialDirectionX * force;
      organelle.velY += radialDirectionY * force;
    }
```

```
  }
}
```

Effect: Organelles gently pushed outward during inhale, pulled inward during exhale (subtle, ~0.15 force strength).

Collision Detection

Organelle-to-Organelle Collisions

Elastic Collision with Separation:

```
handleOrganelleCollisions() {
  for (let i = 0; i < organelles.length; i++) {
    for (let j = i + 1; j < organelles.length; j++) {
      const org1 = organelles[i];
      const org2 = organelles[j];

      const dx = org1.x - org2.x;
      const dy = org1.y - org2.y;
      const distance = Math.sqrt(dx * dx + dy * dy);
      const minDistance = org1.radius + org2.radius;

      if (distance < minDistance && distance > 1) {
        // Normalize collision vector
        const nx = dx / distance;
        const ny = dy / distance;

        // Aggressive separation to prevent sticking
        const overlap = minDistance - distance;
        const separationDistance = overlap / 2 + 5; // Extra 5px separation

        org1.x += nx * separationDistance;
        org1.y += ny * separationDistance;
        org2.x -= nx * separationDistance;
        org2.y -= ny * separationDistance;

        // Calculate relative velocity along collision normal
        const relativeVelX = org1.velX - org2.velX;
        const relativeVelY = org1.velY - org2.velY;
        const velAlongNormal = relativeVelX * nx + relativeVelY * ny;
```

```
        // Don't resolve if velocities are separating
        if (velAlongNormal > 0) continue;

        // Elastic collision with restitution
        const restitution = 0.9; // 90% energy retention
        const impulse = -(1 + restitution) * velAlongNormal;
        const impulseStrength = impulse * 0.5; // Equal mass assumption

        org1.velX += impulseStrength * nx;
        org1.velY += impulseStrength * ny;
        org2.velX -= impulseStrength * nx;
        org2.velY -= impulseStrength * ny;

        // Add minimum separation velocity
        const minSeparationSpeed = 1.0;
        org1.velX += nx * minSeparationSpeed;
        org1.velY += ny * minSeparationSpeed;
        org2.velX -= nx * minSeparationSpeed;
        org2.velY -= ny * minSeparationSpeed;
      }
    }
  }
}
```

Key Steps: 1. **Detect Overlap**: `distance < minDistance` 2. **Separate Positions**: Push apart by `overlap / 2 + 5px` 3. **Calculate Impulse**: Based on relative velocity along collision normal 4. **Apply Impulse**: Update velocities with elastic collision 5. **Ensure Separation**: Add minimum outward velocity to prevent re-collision

Ellipse Boundary Collision (Desktop)

```
handleEllipseBoundary(organelle) {
  const margin = organelle.radius;
  let effectiveRadiusX = this.ellipseRadiusX - margin;
  let effectiveRadiusY = this.ellipseRadiusY - margin;

  // Quadrant-specific adjustment for Bezier overshoot
  const angle = Math.atan2(organelle.y, organelle.x);
  const isRightQuadrant = Math.abs(angle) < Math.PI / 2;

  if (isRightQuadrant) {
    const angleFromHorizontal = Math.abs(angle);
    const adjustmentFactor = 1 - (angleFromHorizontal / (Math.PI / 2)) * 0.02;
    effectiveRadiusX *= (0.97 + adjustmentFactor * 0.03); // 97-100% of original
```

```
  }

  const normalizedX = organelle.x / effectiveRadiusX;
  const normalizedY = organelle.y / effectiveRadiusY;
  const distance = Math.sqrt(normalizedX * normalizedX + normalizedY * normalizedY);

  if (distance >= 1) {
    // Push back inside
    const pushBackFactor = 0.95;
    organelle.x = (normalizedX / distance) * effectiveRadiusX * pushBackFactor;
    organelle.y = (normalizedY / distance) * effectiveRadiusY * pushBackFactor;

    // Calculate surface normal
    const normalX = (2 * organelle.x / (effectiveRadiusX * effectiveRadiusX));
    const normalY = (2 * organelle.y / (effectiveRadiusY * effectiveRadiusY));
    const normalLength = Math.sqrt(normalX * normalX + normalY * normalY);

    if (normalLength > 0) {
      const nx = normalX / normalLength;
      const ny = normalY / normalLength;

      // Simple reflection
      const dot = organelle.velX * nx + organelle.velY * ny;
      organelle.velX -= 2 * dot * nx * this.bounceStrength;
      organelle.velY -= 2 * dot * ny * this.bounceStrength;

      // Record collision for membrane reaction
      const impactForce = Math.abs(dot);
      this.recentCollisions.push({
        x: organelle.x,
        y: organelle.y,
        timestamp: Date.now(),
        force: impactForce * 2.5
      });

      // Clear old collisions (> 600ms)
      this.recentCollisions = this.recentCollisions.filter(
        c => Date.now() - c.timestamp < 600
      );
    }
  }
}
```

Quadrant-Specific Adjustment: - **Problem**: SVG Bezier curves bulge outward more on right side due to `preserveAspectRatio="none"` - **Solution**: Tighten boundary (97-100%) on right quadrants to compensate

Rounded Square Boundary (Mobile)

```
handleRoundedSquareBoundary(organelle) {
  const margin = organelle.radius;
  const left = this.rectLeft + margin;
  const right = this.rectRight - margin;
  const top = this.rectTop + margin;
  const bottom = this.rectBottom - margin;
  const cornerRadius = this.rectBorderRadius;

  // Check if in corner regions
  const inLeftCorner = organelle.x < left + cornerRadius;
  const inRightCorner = organelle.x > right - cornerRadius;
  const inTopCorner = organelle.y < top + cornerRadius;
  const inBottomCorner = organelle.y > bottom - cornerRadius;

  // Handle corners (circular collision with corner centers)
  if ((inLeftCorner && inTopCorner) || ...) {
    // Find corner center
    let cornerX, cornerY;
    // ... calculate corner center based on which corner

    const dx = organelle.x - cornerX;
    const dy = organelle.y - cornerY;
    const distFromCorner = Math.sqrt(dx * dx + dy * dy);

    if (distFromCorner > cornerRadius) {
      // Push back toward corner center
      const pushBackDist = distFromCorner - cornerRadius;
      const nx = dx / distFromCorner;
      const ny = dy / distFromCorner;

      organelle.x -= nx * pushBackDist;
      organelle.y -= ny * pushBackDist;

      // Reflect velocity
      const dot = organelle.velX * nx + organelle.velY * ny;
      organelle.velX -= 2 * dot * nx * this.bounceStrength;
      organelle.velY -= 2 * dot * ny * this.bounceStrength;
    }
  }

  // Handle straight edges
  if (organelle.x < left) {
    organelle.x = left;
    organelle.velX = Math.abs(organelle.velX) * this.bounceStrength;
```

```
  }
  if (organelle.x > right) {
    organelle.x = right;
    organelle.velX = -Math.abs(organelle.velX) * this.bounceStrength;
  }
  // ... same for top/bottom
}
```

Corner Collision: Treats rounded corners as circular arcs (accurate collision response).

Expansion System

Expansion Trigger

Click Handler:

```
const handleOpen = (index) => {
  const id = `organelle-${index}`;
  setActiveIndex(index);
  physicsManagerRef.current?.setExpandedState(
    id,
    true,
    isMobile ? 2.2 : 2.5   // Expansion scale
  );
};
```

Physics Manager Expansion

```
setExpandedState(id, isExpanded, expandedScale = 2.5) {
  const organelle = this.getOrganelle(id);
  if (!organelle) return;

  organelle.isExpanded = isExpanded;

  if (isExpanded) {
    // Store original radius and set expanded radius
    organelle.originalRadius = organelle.radius;
    organelle.radius = organelle.originalRadius * expandedScale; // 90 → 225px

    // Push other organelles away from this one
```

```
    this.pushOrganellesAway(id, organelle.radius);
  } else {
    // Restore original radius
    if (organelle.originalRadius) {
      organelle.radius = organelle.originalRadius;
      delete organelle.originalRadius;
    }
  }
}
```

Push Others Away

```
pushOrganellesAway(expandedId, expandedRadius) {
  const expandedOrg = this.getOrganelle(expandedId);
  if (!expandedOrg) return;

  for (const org of this.organelles) {
    if (org.id === expandedId) continue; // Skip the expanded one

    const dx = org.x - expandedOrg.x;
    const dy = org.y - expandedOrg.y;
    const distance = Math.sqrt(dx * dx + dy * dy);
    const minDistance = expandedRadius + org.radius;

    if (distance < minDistance) {
      // Push away from expanded organelle
      const pushDistance = minDistance - distance + 10; // Extra 10px separation
      const nx = distance > 0 ? dx / distance : 1;
      const ny = distance > 0 ? dy / distance : 0;

      org.x += nx * pushDistance;
      org.y += ny * pushDistance;

      // Add outward velocity to continue moving away
      const pushForce = 5.0;
      org.velX += nx * pushForce;
      org.velY += ny * pushForce;
    }
  }
}
```

Effect: Expanded organelle increases collision radius → Physics detects overlap with neighbors → Pushes them away with velocity.

Close Outside Click

Click Outside to Close:

```
useEffect(() => {
  if (activeOrganelleId == null) return;

  const handlePointerDown = (event) => {
    if (event.target.closest(".organelle")) return; // Clicked on organelle itself
    closeActiveOrganelle(); // Clicked outside
  };

  document.addEventListener("pointerdown", handlePointerDown);
  return () => document.removeEventListener("pointerdown", handlePointerDown);
}, [activeOrganelleId, closeActiveOrganelle]);
```

Mobile Responsiveness

Breakpoint: 768px

```
useEffect(() => {
  const updateBreakpoint = () => setIsMobile(window.innerWidth <= 768);
  updateBreakpoint();
  window.addEventListener("resize", updateBreakpoint);
  return () => window.removeEventListener("resize", updateBreakpoint);
}, []);
```

Layout Differences

Feature	Desktop ($>$ 768px)	Mobile (768px)
Membrane Shape	Breathing circle (organic)	Static rounded square
Organelle Size	170px diameter	70px diameter
Expansion Scale	2.5× (170 → 425px)	2.2× (70 → 154px)
Expanded Position	Move to center (x: 0, y: 0)	Stay in place (x: position.x, y: position.y)
Expanded Shape	Rounded rectangle (20px radius)	Stay circular (50%)

Feature	Desktop (> 768px)	Mobile (768px)
Boundary Type	Ellipse (dynamic with breathing)	Rounded square (85% of viewport)
Breathing Animation	Active (10s cycle)	Disabled (static)
Breathing Force	Active (subtle push/pull)	Disabled

Mobile Optimizations

No Breathing Animation:

```
useEffect(() => {
  if (isMobile) return; // Skip entire breathing animation loop

  const animate = () => {
    // ... breathing logic
  };

  animate();
}, [isMobile]);
```

Why? - **CPU Usage**: Continuous SVG path recalculation expensive on mobile - **Battery**: Constant animation drains battery - **UX**: Static rounded square sufficient on small screens

Smaller Organelles:

```
const ORGANELLE_DESKTOP_SIZE = 170;
const ORGANELLE_MOBILE_SIZE = 70;
```

Why? - **Space**: 4 organelles need more room to move in small viewport - **Touch Targets**: 70px still meets minimum touch target size (48×48px)

Performance Optimizations

RequestAnimationFrame Loop

Synchronized with Browser Rendering:

```
const syncState = () => {
  const currentManager = physicsManagerRef.current;
  if (!currentManager) return;

  setOrganellePositions(currentManager.getDisplayPositions());
  animationFrameRef.current = requestAnimationFrame(syncState);
};

animationFrameRef.current = requestAnimationFrame(syncState);
```

Why RAF? - **60 FPS**: Syncs with monitor refresh rate - **Throttled**: Pauses when tab inactive (saves CPU) - **Smooth**: No jank from `setInterval` timing issues

ResizeObserver for Container Updates

```
resizeObserverRef.current = new ResizeObserver(([entry]) => {
  const { width, height } = entry.contentRect;
  physicsManagerRef.current?.updateDimensions(width, height);
});
resizeObserverRef.current.observe(container);
```

Why ResizeObserver vs Window Resize? - **Precise**: Fires only when specific container resizes - **Debounced**: Browser handles throttling internally - **Efficient**: Doesn't fire for unrelated window resizes

useMemo for Blob Path

```
const blobPath = useMemo(() => {
  const result = isMobile
    ? generateRoundedSquarePath()
    : generateOrganicBlobPath(breathScale, wavePhase, breathProgress);
  return result.path;
}, [isMobile, breathScale, wavePhase, breathProgress]);
```

Why Memoize? - **Expensive**: Trigonometry + Bezier curve calculations - **Frequent**: Recalculates 60 times/second during breathing - **Memoization**: Only recalculates when dependencies change (not on every render)

Cleanup on Unmount

```
useEffect(() => {
  // ... setup physics, RAF, resize observer
```

```
  return () => {
    if (animationFrameRef.current) {
      cancelAnimationFrame(animationFrameRef.current);
      animationFrameRef.current = null;
    }
    resizeObserverRef.current?.disconnect();
    resizeObserverRef.current = null;
    physicsManagerRef.current?.stop();
    physicsManagerRef.current = null;
  };
}, [isMobile]);
```

Prevents: - **Memory Leaks**: RAF continues after unmount - **Multiple Loops**: New physics loop created without stopping old one - **Observer Leaks**: ResizeObserver continues observing

Will-Change Optimization

```
style={{
  willChange: active ? 'width, height, transform' : 'auto'
}}
```

How It Works: - **Browser Hint**: Tells browser to optimize these properties for animation - **GPU Layer**: Creates separate GPU layer for smooth animation - **Only When Active**: Disabled when inactive (saves GPU memory)

Future Enhancements

Potential Additions

1. **Mitosis Animation**
   - **Current**: 4 static organelles
   - **Enhancement**: Organelles divide when clicked ($1 \rightarrow 2$)
   - **Implementation**: Clone organelle with small offset, separate with velocity
2. **Nucleus Center**
   - **Current**: Empty cell interior
   - **Enhancement**: Large central nucleus (non-interactive)
   - **Implementation**: Fixed circular SVG in center, organelles avoid it

3. **Protein Floating Particles**
    - **Current**: Empty cytoplasm
    - **Enhancement**: Small particles floating around (like organelles but smaller)
    - **Implementation**: Additional physics objects with different collision properties
4. **Membrane Ripple on Collision**
    - **Current**: Collisions tracked but not visualized
    - **Enhancement**: Visible ripple effect at collision point
    - **Implementation**: SVG filter animation triggered by `recentCollisions` array
5. **Cilia/Flagella Movement**
    - **Current**: Static membrane
    - **Enhancement**: Hair-like projections along membrane edge
    - **Implementation**: SVG paths extending from membrane, animated with sine waves
6. **Organelle Trails**
    - **Current**: No motion history
    - **Enhancement**: Faint trailing lines showing recent path
    - **Implementation**: Store last N positions, render as SVG polyline with opacity gradient
7. **Zoom/Pan Controls**
    - **Current**: Fixed view
    - **Enhancement**: Mouse wheel zoom, drag to pan
    - **Implementation**: Transform container with CSS `transform: scale() translate()`
8. **Organelle Connections**
    - **Current**: No relationships visualized
    - **Enhancement**: Lines connecting related projects
    - **Implementation**: SVG lines between organelle centers, fade in on hover

---

Related Documentation

- ACADEMIC-JOURNEY-SECTION.md - Previous section (timeline)
- BLOG-SECTION.md *(coming soon)* - Next section (blog cards)
- HERO-SECTION.md - DNA helix physics patterns

- ARCHITECTURE.md - Component hierarchy
- MASTER-OVERVIEW.md - Full portfolio overview

Quick Reference

Key Files

File	Lines	Purpose
`WorkSection.jsx`	246	Main orchestrator, state management
`OrganelleNew.jsx`	171	Individual project cards
`OrganicBlob.jsx`	404	Breathing membrane SVG
`OrganellePhysics.js`	520	Physics simulation engine

Physics Constants

Constant	Value	Purpose
damping	0.995	Velocity decay (0.5% per frame)
bounceStrength	0.8	Energy retention on collision (80%)
minSpeed	2.0	Minimum pixels/frame (prevents stopping)
maxSpeed	3.5	Maximum pixels/frame (prevents chaos)
separationForce	0.3	Repulsion between organelles
breathingForceStrength	0.15	Membrane breathing pull (desktop)

Breathing Cycle

Phase	Duration	Progress	Radius
Inhale	4 seconds	$0 \rightarrow 1$	$37 \rightarrow 43$ viewBox units
Exhale	6 seconds	$1 \rightarrow 0$	$43 \rightarrow 37$ viewBox units
Total	10 seconds	Full cycle	$\pm 8\%$ of base radius

Organelle Sizes

State	Desktop	Mobile
Default	170px	70px
Expanded	425px (2.5×)	154px (2.2×)
Dimmed	85px (0.5×)	35px (0.5×)

Color Scheme

Type	Color	Hex
Education	Purple	#9333ea
Research	Sky Blue	#0ea5e9
Automation	Emerald	#10b981
Clinical	Amber	#f59e0b

This section demonstrates advanced frontend techniques: real-time physics simulation, SVG path animation, complex state orchestration, and biological metaphor as interaction design.