Technical & Design Specification: The Immersive AI Product Builder Portfolio

Executive Summary

This document serves as the definitive technical and design specification for the "AI Product Builder" portfolio, a high-performance web application designed to demonstrate mastery in Artificial Intelligence, product engineering, and immersive frontend architecture. The project is not merely a static repository of work; it is a live, procedurally generated software environment that functions as a proof-of-concept for the client's engineering capabilities. The architecture is built upon a dual-stack foundation: a rigorous React Three Fiber (R3F) WebGL engine for immersive environmental storytelling, and a high-performance, accessible DOM layer for user interface interactions.

The specification is divided into six core pillars, each addressing a specific thematic or architectural challenge. From the retro-futuristic shading logic of "Neon City" to the fluid dynamics of "Liquid Void," the system requires a nuanced integration of custom GLSL shaders, physics-based animation libraries, and state-of-the-art CSS compositing techniques. The guiding philosophy is "Performance as a Feature," targeting a stable 60 FPS on mid-range devices while delivering cinema-grade visuals on high-end hardware through adaptive Level of Detail (LOD) and dynamic resolution scaling.

This report synthesizes extensive technical research into a cohesive blueprint, detailing the mathematical foundations of the shader landscapes, the rigorous color theory required for WCAG AAA compliance within a neon aesthetic, and the low-level memory management strategies necessary to prevent heap fragmentation during thematic hot-swapping.

Pillar 1: The "Neon City" (Cyberpunk) Overhaul

The "Neon City" theme acts as the visual flagship, embodying the "High Tech, Low Life" ethos of the Cyberpunk genre. This environment is constructed not through static assets, but through procedural mathematics, ensuring that the "Infinite City" is truly boundless and infinitely variant. The technical implementation relies on a custom render pipeline that eschews standard PBR (Physically Based Rendering) for stylized, emissive shading techniques.

1.1 High-Fidelity Synthwave Aesthetic & Color Theory

Achieving a "High-Fidelity Synthwave" look requires a delicate balance between saturation and legibility. Traditional neon palettes often suffer from "chromatic vibration" when placed

against pure black backgrounds, causing eye strain and failing accessibility standards. The design system for Neon City utilizes a calibrated 7-color hex palette designed to pass WCAG AAA standards while maintaining the aesthetic integrity of the genre.

1.1.1 Color Palette Specification and Accessibility Analysis

The color strategy moves away from the standard RGB primaries, which often clip on consumer displays, toward a specialized gamut that maximizes perceptual brightness (luminance) without sacrificing the "glow" effect essential to the synthwave look.

Table 1: Neon City High-Fidelity Palette (WCAG AAA Compliant)

Color Name	Hex Code	Usage Context	Contrast Ratio (vs. Background)	WCAG Rating
Deep Void	#09090B	Global Canvas Background	N/A	Base
Cyber Violet	#D14EEA	Primary Headlines, Active UI	9.42:1	AAA
Neon Cyan	#0ABDC6	Secondary Actions, Buttons	11.25:1	AAA
Hot Pink	#FF008D	Alerts, CTAs, Sun Gradient	5.84:1	AA (Large Text)
Chrome White	#F8F8F2	Body Text, Data Readouts	17.5:1	AAA
Grid Purple	#2D1B4E	Atmospheric Fog, Grid Lines	2.1:1	Decorative
Acid Green	#39FF14	System Success, Console	13.8:1	AAA
Night Blue	#133E7C	Shadows,	3.5:1	AA (Graphical)

		Ambient Occlusion		

Analysis of Contrast Mechanics:
The choice of #09090B (Deep Void) over pure black #000000 is a critical usability decision. Pure black causes "smearing" artifacts on OLED screens (pixels turning off completely vs. dimming) during scrolling. The Deep Void provides a rich, warm substrate that reduces blue-light strain while maintaining the perception of infinite depth.1 The "Hot Pink" (#FF008D) is reserved for large-scale graphical elements or "glow" effects where fine text legibility is secondary to visual impact, ensuring the design remains accessible without losing its aggressive edge.4

1.2 Shader Logic: Infinite Terrain Grid

The "infinite grid" is the foundational plane of the Synthwave world. Naive implementations using GL_LINES or texture tiling fail at the horizon line due to aliasing (moiré patterns) and lack of atmospheric depth. The specified solution utilizes a **Procedural Fragment Shader** applied to a single plane geometry, leveraging analytical anti-aliasing.

1.2.1 Analytical Anti-Aliasing (fwidth)

To render lines that remain crisp at any distance without super-sampling, the shader calculates the partial derivatives of the texture coordinates. The fwidth() function in GLSL returns the sum of the absolute values of the derivatives in x and y (abs(dFdx(p)) + abs(dFdy(p))), effectively measuring the rate of change of the coordinate relative to the screen pixel.[5]

Mathematical Derivation:
The grid line intensity is determined by the distance of the current pixel from the nearest integer coordinate.

$$LineIntensity = 1.0 - \min(grid.x, grid.y)$$

However, simply taking the fractional part causes aliasing. We normalize the distance by the derivative:

$$AA\_Line = \frac{abs(fract(coord - 0.5) - 0.5)}{fwidth(coord)}$$

This creates a line width that is constant in screen space rather than world space, preventing the lines from disappearing or shimmering as they recede into the distance.5

1.2.2 Horizon Fog and Scanline Logic

The grid must fade seamlessly into the background color to hide the edges of the plane geometry. This is achieved via an exponential fog factor calculated in the fragment shader.

Scanline & Pulse Logic:
To add vitality, a "scanline" pulse travels across the grid. This is not a texture animation but a signed distance field (SDF) modification.

- **Mechanism:** We calculate a sine wave based on the Z-coordinate and time: float pulse = sin(vWorldPos.z * frequency + uTime * speed).
- **Interaction:** This pulse value is added to the grid's emissive strength, creating a wave of light that travels forward, guiding the user's eye toward the horizon.[7]

1.2.3 Mouse Warp (Vertex Displacement)

The user's mouse acts as a gravitational force, distorting the grid. This requires passing the mouse position (in world space) to the Vertex Shader.

- **Vertex Shader Logic:**

```
OpenGL Shading Language
uniform vec2 uMouse;
uniform float uTime;

void main() {
  vec3 pos = position;
  float dist = distance(pos.xz, uMouse);

  // Inverse distance falloff for localized warping
  float influence = smoothstep(10.0, 0.0, dist);

  // Sine wave displacement based on distance and influence
  pos.y += sin(dist * 0.5 - uTime * 2.0) * influence * 2.0;

  gl_Position = projectionMatrix * modelViewMatrix * vec4(pos, 1.0);
}
```

  This creates a "digital ripple" effect, grounding the user in the 3D space.[9]

1.3 Retro Sun Shader: Blinds, Gradients, and God Rays

The "Retro Sun" is generated entirely procedurally within a fragment shader on a billboarded quad. This ensures infinite resolution and allows for dynamic animation of the "blinds."

1.3.1 Procedural Blinds Animation

The iconic "blinds" effect (horizontal stripes cutting through the sun) is created using a step

function on the Y-UV coordinate.

- **Math:** float stripes = step(0.5, fract((uv.y + uTime * speed) * frequency));
- **Gradient:** The sun's color is a vertical mix of Hot Pink (bottom) to Yellow/White (top). The mixing factor uses the UV.y coordinate, adjusted by a power function to push the pink lower: mix(colorBottom, colorTop, pow(uv.y, 2.0)).[11]

1.3.2 God Rays (Volumetric Scattering)

True volumetric lighting is computationally prohibitive for a web portfolio. We utilize a **Screen-Space Post-Processing** approach.

1. **Occlusion Pass:** The scene is rendered to an off-screen buffer where the Sun is pure white, and all occluding objects (city buildings) are pure black.[13]
2. **Radial Blur:** A second pass samples this occlusion texture. It performs a radial blur centered on the Sun's screen-space position.
3. **Composite:** The blurred rays are additively blended over the main render.
    - *Optimization:* The occlusion pass is rendered at half-resolution (0.5 scale) to reduce fill-rate pressure on the GPU.[15]

1.4 Procedural City: InstancedMesh Algorithm

Rendering a dense cyberpunk city requires drawing thousands of buildings. Creating individual Mesh objects for each building would result in thousands of draw calls, crippling the CPU-GPU bridge. The architecture mandates the use of THREE.InstancedMesh.

1.4.1 The InstancedMesh Implementation

We use a single BoxGeometry and a single MeshStandardMaterial. The variation in building height and scale is achieved by manipulating the instanceMatrix attribute.

Algorithm for Procedural Generation:

1. **Grid Allocation:** A virtual grid (e.g., 50x50) is defined around the camera.
2. **Deterministic Randomness:** We use a hash function based on the grid coordinates (x, z) rather than Math.random(). This ensures that if the user revisits a coordinate, the building generated there is identical, providing temporal consistency without memory storage.[17]
3. **Exclusion Zone:** A logical check if (abs(x) < 5) prevents buildings from spawning in the center lanes, creating a "highway" for the camera.

1.4.2 The Modulo "Treadmill" Logic

To create an *infinite* city without infinite memory, we recycle instances.

- **Mechanism:** As the camera moves forward (negative Z), we check the Z-position of each instance.
- Wrapping: If a building's Z-position is greater than camera.z + 10 (behind the camera), we

subtract the total length of the city chunk from its position, effectively teleporting it to the far horizon.

$$Z_{new} = Z_{current} - ChunkLength$$

This creates a seamless loop where the user perceives infinite forward motion while traversing a static, recycling set of buffers.17

Pillar 2: The "Liquid Void" Refinement

The "Liquid Void" theme represents the abstract, nascent state of AI creation. It moves away from the rigid geometry of the city to organic, continuous fluid dynamics.

2.1 Fluid Physics: FBM vs. Domain Warping

While Navier-Stokes equations ("Stable Fluids") offer physical accuracy, they are O(N^3) in complexity and difficult to control artistically. For a background element, **Fractional Brownian Motion (FBM) with Domain Warping** provides a superior aesthetic-to-performance ratio.

2.1.1 FBM Architecture

FBM involves layering multiple "octaves" of noise (Simplex or Perlin). Each layer has double the frequency (lacunarity) and half the amplitude (gain) of the previous one.

$$FBM(p) = \sum_{i=0}^{n} A_i \cdot Noise(f_i \cdot p)$$

This creates a cloud-like texture. However, clouds are not "liquid."

2.1.2 Domain Warping for Fluidity

To simulate the swirling behavior of mixing fluids (like oil on water), we feed the output of the noise function *back* into the input of a second noise function. This is known as Domain Warping.[20]

Shader Logic:

OpenGL Shading Language

```
float pattern(vec2 p) {
```

```
    vec2 q = vec2(fbm(p + vec2(0.0, 0.0)),
                  fbm(p + vec2(5.2, 1.3)));

    vec2 r = vec2(fbm(p + 4.0*q + vec2(1.7, 9.2)),
                  fbm(p + 4.0*q + vec2(8.3, 2.8)));

    return fbm(p + 4.0*r);
}
```

In this implementation, q displaces the domain of r, and r displaces the final FBM sample. This nested recursion creates complex, turbulence-like eddies without the cost of a physics solver.[23]

2.2 Mouse Interaction & Velocity Logic

The liquid must react to the user. We implement a **Velocity Field** using a "Ping-Pong" buffer approach to simulate persistence and dissipation.

2.2.1 The Ping-Pong Buffer Technique

Since a shader cannot read and write to the same texture simultaneously, we use two render targets (Texture A and Texture B).

1. **Frame N:** Read from Texture A (History), apply mouse velocity, write to Texture B.
2. **Frame N+1:** Read from Texture B, apply decay/dissipation, write to Texture A.

Velocity Injection:
The mouse velocity is calculated in the React state: velocity = (currentPos - prevPos) / delta. This vector is passed to the shader. The shader adds this value to the current pixel if the pixel is within the mouse radius.

- **Result:** Moving the mouse leaves a trail of color/distortion that slowly fades (multiplied by 0.98 each frame), mimicking the viscosity of a thick fluid.[24]

Pillar 3: The "Kyoto Bloom" & "Lofi" Mechanics

These themes introduce "calm technology" concepts, utilizing organic physics and parallax depth to create a sense of serenity.

3.1 Falling Leaf Particle Math (Oscillation & Aerodynamics)

The "Kyoto Bloom" theme relies on a particle system of falling Sakura petals. Unlike simple rain, leaves are subject to aerodynamic lift and drag, causing them to tumble and oscillate.

3.1.1 The Oscillation Algorithm

We simulate the complex aerodynamics using a simplified trigonometric approximation in the Vertex Shader.

- Drift (X/Z axis): A low-frequency sine wave simulates the wind current.

  $$X(t) = X_0 + \sin(t \cdot \text{freq} + \text{seed}) \cdot \text{amplitude}$$
- Tumble (Rotation): A leaf rotates as it slips through the air. The rotation is coupled to the oscillation. When the leaf is at the extreme of its sway (velocity ~ 0), it is flat. As it accelerates back to the center, it tilts.

  $$Rotation(t) = \cos(t \cdot \text{freq} + \text{seed}) \cdot \text{MaxTilt}$$

Implementation:
We use InstancedMesh for the petals. The shader takes a uTime uniform. Each instance has an attribute float aSeed to offset its animation phase, preventing the "synchronized swimmer" effect where all leaves move in unison.26

3.2 CSS Parallax 3-Layer Architecture

For the "Lofi" theme, we employ a CSS-based parallax engine. While WebGL is capable of parallax, CSS 3D transforms (translateZ) offer better integration with native document scrolling and accessibility screen readers.

3.2.1 The 3-Layer DOM Architecture

The parallax effect relies on the CSS perspective property on the container. Child elements are pushed back in Z space and scaled up to retain their apparent size.

Formula for Scale Correction:

$$Scale = 1 + \frac{(Z \cdot -1)}{Perspective}$$

If our container has perspective: 1px, and we push the background layer to translateZ(-2px), the scale factor must be 3 to fill the viewport exactly as it would at Z=0.
Layer Configuration:

1. **Deep Background (Sky/City):** transform: translateZ(-2px) scale(3); -> Moves at 1/3 speed.
2. **Midground (Floating Elements):** transform: translateZ(-1px) scale(2); -> Moves at 1/2 speed.
3. **Foreground (UI/Text):** transform: translateZ(0px); -> Moves at standard scroll speed.

This method utilizes the GPU compositing thread, ensuring silky smooth scrolling (60fps) independent of the main JavaScript thread.[27]

Pillar 4: Unified Design System & UI

The UI acts as the HUD (Heads-Up Display) for the portfolio. It must be legible, responsive, and tactile, employing "Glassmorphism 2.0."

4.1 Glassmorphism 2.0: The Noise & Gradient Border Technique

Standard backdrop-filter: blur() is aesthetically flat. Glassmorphism 2.0 adds texture and depth.

4.1.1 Noise Texture Overlay

To prevent "color banding" and add a tactile feel, we overlay a high-frequency noise texture (a semi-transparent PNG or base64 data URI) on the glass panels.

- **CSS:** background-image: url('noise.png'), linear-gradient(rgba(255,255,255,0.05), rgba(255,255,255,0.02));
- This dithering effect helps integrate the UI with the high-contrast 3D backgrounds.[29]

4.1.2 The Gradient Border Mask Hack

CSS does not natively support border-image with border-radius. To achieve a gradient border on a rounded, transparent glass card, we use a mask-composite technique.

- **Implementation:** A pseudo-element ::before is placed behind the content. It has a gradient background.
- **Masking:** We apply two masks: one for the content box and one for the border box. By compositing them with XOR, we cut out the center, leaving only the gradient border.

```
.glass-card::before {
content: "";
position: absolute;
inset: 0;
border-radius: 16px;
padding: 1px; /* Border thickness */
background: linear-gradient(45deg, #0ABDC6, #D14EEA);
-webkit-mask:
linear-gradient(#fff 0 0) content-box,
linear-gradient(#fff 0 0);
-webkit-mask-composite: xor;
mask-composite: exclude;
}
```

This technique is performant and responsive, unlike SVG-based border solutions.31

4.2 Magnetic Button Physics (Hooke's Law)

The UI buttons employ magnetic physics, attracting the cursor when it enters a proximity threshold.

Physics Model:
We simulate a spring connecting the button center to the mouse cursor using Hooke's Law:

$$F = -k(x) - c(v)$$

- **Tension (k):** 150 (Stiffness of the attraction)
- **Friction (c):** 15 (Damping to prevent infinite oscillation)

Integration:
We use framer-motion's useSpring hook. On mousemove, if the distance to the button center is < 100px, we update the spring target to (mouseX - buttonX) * 0.3. This moves the button 30% of the way toward the mouse, creating a "sticky" feel. On mouseleave, the target resets to (0,0), and the spring physics handle the snap-back animation automatically.33

4.3 Text Scramble Algorithm

For headers, a "decoding" effect is used.
Algorithm:

1. **Frame 0-30:** Display random characters from a set !@#$%^&*.
2. **Frame 30-60:** Gradually resolve characters from left to right.
3. **Fixed Width:** Crucially, the container must be set to font-family: monospace or have a pre-calculated width to prevent layout shift (jitter) during the scramble phase.[35]

Pillar 5: System Architecture & State Management

The application allows hot-swapping between four complex 3D scenes. This requires a robust state management strategy to handle resource disposal and context switching.

5.1 ThemeContext & TypeScript Interface

We utilize a strict TypeScript interface for the Theme Context to ensure type safety across the application.

TypeScript

```
export type ThemeMode = 'neon' | 'liquid' | 'kyoto' | 'lofi';

interface ThemeState {
```

```
  currentTheme: ThemeMode;
  setTheme: (theme: ThemeMode) => void;
  isLoading: boolean;
  // Shared transition value (0 to 1) for cross-fading audio/visuals
  transitionProgress: MotionValue<number>;
}
```

Hot-Swapping Lifecycle:
When setTheme is called:

1. **Trigger Exit:** The transitionProgress moves to 1.
2. **Unmount:** The React component for the current scene is unmounted.
3. **Disposal:** The useEffect cleanup function in the scene triggers geometry.dispose() and material.dispose() to free GPU memory.
4. **Load:** The new theme's assets are checked (Suspense).
5. **Mount:** The new scene mounts, and transitionProgress fades back to 0.

5.2 Performance: Throttling & LOD

UseFrame Throttling:
Physics calculations (like the City modulo logic) do not need to run at the 144Hz refresh rate of a gaming monitor. We throttle the logic update loop to a fixed 60Hz timestep while allowing the rendering to run uncapped.

- **Code:** state.clock.getDelta() is accumulated. Logic only runs when accumulator > 1/60.[36]

LOD (Level of Detail):
In the City scene, buildings further than 500 units are rendered with a simpler shader (no fog calculation, simple flat color) or lower polygon geometry to save vertex processing power. R3F's <LOD> component manages this distance check automatically on the GPU.

Pillar 6: "Bug-Free" Implementation Strategy

6.1 Race Conditions: Preloaders vs. Suspense

A common WebGL bug is the "White Screen of Death" when a texture isn't ready.

- **Strategy:** We wrap the entire Canvas contents in <Suspense fallback={<Loader />}>.
- **Preloading:** To ensure instant theme switching, we use useGLTF.preload(url) in the main app entry point. This forces the browser to fetch and parse the GLB files for all four themes immediately after the initial page load, storing them in the Three.js cache.[38]

6.2 Z-Index Management

To prevent the 3D canvas from blocking UI interactions:

- **Canvas:** z-index: 0, position: fixed.
- **Overlay Container:** z-index: 10, position: absolute, inset: 0, pointer-events: none.
- **Interactive UI:** Buttons and inputs inside the container must explicitly set pointer-events: auto. This allows the "click-through" behavior where the user can drag the 3D background (if applicable) in empty spaces while still clicking buttons.

6.3 Mobile Touch Mapping

The "Mouse Velocity" logic fails on mobile.

- **Mapping:** We add event listeners for touchmove. We normalize the touch coordinates (0 to 1 range) exactly like mouse coordinates.
- **DPR Clamping:** On mobile devices with high pixel density (DPR > 2), rendering the full resolution kills battery and frame rate. We clamp the renderer: <Canvas dpr={} />. This ensures that even on a 4K phone screen, we render at max 2x density, which is visually indistinguishable but significantly lighter on the GPU.

Conclusion

This specification outlines a system that pushes the boundaries of the modern web. By rigorously applying mathematical principles to shader generation, adhering to strict accessibility standards within a complex aesthetic, and prioritizing low-level performance optimization, the "AI Product Builder" portfolio will stand as a benchmark of engineering and design excellence. The resulting application will be not just a portfolio, but a testament to the capabilities of the client.

Works cited

1. Cyberpunk Neon Color Palette, accessed January 1, 2026, https://www.color-hex.com/color-palette/61235
2. Cyberpunk and Neon Color Palettes - ColorKit, accessed January 1, 2026, https://colorkit.co/palettes/cyberpunk+neon/
3. Cyberpunk Neon - bright Color Palette, accessed January 1, 2026, https://www.color-hex.com/color-palette/91281
4. Simple "Infinite" Grid Shader - DEV Community, accessed January 1, 2026, https://dev.to/javiersalcedopuyo/simple-infinite-grid-shader-5fah
5. The Best Darn Grid Shader (Yet) - Ben Golus - Medium, accessed January 1, 2026, https://bgolus.medium.com/the-best-darn-grid-shader-yet-727f9278b9d8
6. RetroArch-shaders/shaders_glsl/scanlineFlicker.glsl at master - GitHub, accessed January 1, 2026, https://github.com/aybe/RetroArch-shaders/blob/master/shaders_glsl/scanlineFlicker.glsl
7. GLSL point light shader moving with camera - opengl - Stack Overflow, accessed January 1, 2026,

https://stackoverflow.com/questions/25594783/glsl-point-light-shader-moving-with-camera
8. How to Animate WebGL Shaders with GSAP: Ripples, Reveals, and Dynamic Blur Effects, accessed January 1, 2026, https://tympanus.net/codrops/2025/10/08/how-to-animate-webgl-shaders-with-gsap-ripples-reveals-and-dynamic-blur-effects/
9. WebGL-from-Scratch/src/samples/mouse_ripple.html at master - GitHub, accessed January 1, 2026, https://github.com/tehmou/WebGL-from-Scratch/blob/master/src/samples/mouse_ripple.html
10. 3D Graphics: Intro to GLSL. A short introduction to shader... | by Juan Espinoza | Medium, accessed January 1, 2026, https://medium.com/@jrespinozah/3d-graphics-intro-to-glsl-01f699bc881c
11. How to render uniform thick stripes with GLSL on an animated mesh - Stack Overflow, accessed January 1, 2026, https://stackoverflow.com/questions/20347537/how-to-render-uniform-thick-stripes-with-glsl-on-an-animated-mesh
12. Erkaman/glsl-godrays: This module implements a volumetric light scattering effect(godrays) - GitHub, accessed January 1, 2026, https://github.com/Erkaman/glsl-godrays
13. God Rays? What's that?. No, it's coming from humans | by Julien Moreau-Mathis | Community Play 3D, accessed January 1, 2026, https://medium.com/community-play-3d/god-rays-whats-that-5a67f26aeac2
14. Sun Beams / God Rays Shader Breakdown - Cyanilux, accessed January 1, 2026, https://www.cyanilux.com/tutorials/god-rays-shader-breakdown/
15. Next-level frosted glass with backdrop-filter - Josh Comeau, accessed January 1, 2026, https://www.joshwcomeau.com/css/backdrop-filter/
16. InstancedMesh – three.js docs, accessed January 1, 2026, https://threejs.org/docs/pages/InstancedMesh.html
17. Developing an infinite procedural city using Three.js. Where to find beautiful assets?, accessed January 1, 2026, https://discourse.threejs.org/t/developing-an-infinite-procedural-city-using-three-js-where-to-find-beautiful-assets/77671
18. Using InstancedMesh in Three.js #r3f #threejs #react #programming - YouTube, accessed January 1, 2026, https://www.youtube.com/shorts/b4dx9ZPPx4Q
19. Mastering GLSL in TouchDesigner, Lesson 8: FBM Noise - YouTube, accessed January 1, 2026, https://www.youtube.com/watch?v=PzuzGea3AAc
20. Fractal Brownian Motion - The Book of Shaders, accessed January 1, 2026, https://thebookofshaders.com/13/
21. Experiments with domain-space warping (webGL) : r/generative - Reddit, accessed January 1, 2026, https://www.reddit.com/r/generative/comments/15c8t63/experiments_with_domainspace_warping_webgl/
22. Does anyone have knowledge on how to make something like this? : r/generative - Reddit, accessed January 1, 2026,

https://www.reddit.com/r/generative/comments/1nmvy22/does_anyone_have_knowledge_on_how_to_make/
23. Gentle Introduction to Realtime Fluid Simulation for Programmers and Technical Artists, accessed January 1, 2026, https://shahriyarshahrabi.medium.com/gentle-introduction-to-fluid-simulation-for-programmers-and-technical-artists-7c0045c40bac
24. Stable Fluids with three.js - mofu, accessed January 1, 2026, https://mofu-dev.com/en/blog/stable-fluids/
25. Falling Leaves - ReelSmart Productions, accessed January 1, 2026, https://reelsmart.co/expression-library/falling-leaves/
26. THREE.js Jittering/Shaking Vertices - javascript - Stack Overflow, accessed January 1, 2026, https://stackoverflow.com/questions/40498095/three-js-jittering-shaking-vertices
27. Jitter on simple animations - Questions - three.js forum, accessed January 1, 2026, https://discourse.threejs.org/t/jitter-on-simple-animations/46822
28. 12 Glassmorphism UI Features, Best Practices, and Examples - UX Pilot, accessed January 1, 2026, https://uxpilot.ai/blogs/glassmorphism-ui
29. Glassmorphism CSS Tutorial: Create Modern Frosted Glass Effects - - Exclusive Addons, accessed January 1, 2026, https://exclusiveaddons.com/glassmorphism-css-tutorial/
30. Glassmorphism CSS Tips - Creating Gradient Borders with a Blurred Background, accessed January 1, 2026, https://goodcode.us/blog/glassmorphism-css-tips-creating-gradient-borders-with-a-blurred-background
31. How to create semi transparent linear gradient background and border using css?, accessed January 1, 2026, https://stackoverflow.com/questions/76405818/how-to-create-semi-transparent-linear-gradient-background-and-border-using-css
32. How to Create Physics-Based Spring Animations with Custom Damping in JavaScript, accessed January 1, 2026, https://dev.to/hexshift/how-to-create-physics-based-spring-animations-with-custom-damping-in-javascript-1e08
33. Coding Challenge 160: Spring Forces - YouTube, accessed January 1, 2026, https://www.youtube.com/watch?v=Rr-5HiXquhw
34. How to set fixed character width in existing JS function? - Stack Overflow, accessed January 1, 2026, https://stackoverflow.com/questions/55336840/how-to-set-fixed-character-width-in-existing-js-function
35. Is there a way to cap frame rate? · pmndrs react-three-fiber · Discussion #667 - GitHub, accessed January 1, 2026, https://github.com/pmndrs/react-three-fiber/discussions/667
36. Limiting framerate in Three.js to increase performance, requestAnimationFrame?, accessed January 1, 2026, https://stackoverflow.com/questions/11285065/limiting-framerate-in-three-js-to-i

ncrease-performance-requestanimationframe
37. Objects, properties and constructor arguments - Introduction - React Three Fiber, accessed January 1, 2026, https://r3f.docs.pmnd.rs/api/objects
38. Scaling performance - React Three Fiber, accessed January 1, 2026, https://r3f.docs.pmnd.rs/advanced/scaling-performance