CODROPS

Stas Bondar ’25: The Code & Techniques Behind a Next-Level Portfolio

A look behind stabondar.com — a creative portfolio featuring advanced web animations with GSAP, Three.js, and physics-based effects.

By Stas Bondar in Articles on March 25, 2025

Visit

*After Stas Bondar’s portfolio was named Site of the Week **and** Site of the Month by the GSAP team, we knew we had to take a closer look. In this exclusive deep dive, Stas walks us through the code, animation logic, and creative process that brought his stunning portfolio to life.*

Every finished portfolio has a story behind it, and mine is no exception. What visitors see today at stabondar.com is the result of more than a year of planning, experimentation, iteration, and refinement. It was quite a journey—juggling essential projects here and there, squeezing in time for my website late at night or on weekends. Playing around with different animation scenarios and transitions, only to redo everything when it didn’t turn out the way I expected… and then redoing it again. Yeah, it was a long ride.

I’m thrilled to share that two months ago, after dedicating countless hours to development, I finally launched my new portfolio website! My vision was bold: to create a vibrant showcase that

truly reflects the exciting possibilities of prioritizing animation and interaction design in web development.

This backstage tour will dive into how GSAP helped tackle key challenges, walking through the technical implementation of animations and sharing code examples you can adapt for your own projects.

Technical Overview

For my portfolio website, I needed a tech stack that offered both flexibility for creative animations and solid performance. Here's a breakdown of the key technologies and how they work together:

Astro Build

Not long ago, every project I worked on was built in Webflow. While it's a powerful platform, I realized I wasn't using Webflow's built-in interactions—I always preferred to handcraft every animation, interaction, and page transition myself. At some point, it hit me: I was essentially using Webflow as a static HTML generator, while all the dynamic elements were custom code.

That realization led me to explore alternatives, and Astro turned out to be the perfect solution. I wanted a framework that allowed me to quickly structure HTML and dive straight into animations without unnecessary overhead. Astro gave me exactly that—a streamlined development experience with:

- Fast page loads through partial hydration
- A simple component architecture that kept my code organized
- Minimal JavaScript by default, allowing me to add only what I needed
- Excellent handling of static assets—crucial for a portfolio site
- The flexibility to use modern JavaScript features while still delivering optimized output

Animations: GSAP

When I mentioned custom animations in Webflow, I was really talking about GSAP. The GreenSock Animation Platform has been the backbone of my projects for years, and I keep discovering new features that make me love it even more.

I still vividly remember working on Dmitry Kutsenko's portfolio four years ago. Back then, I wasn't particularly comfortable with JavaScript and relied heavily on Webflow's built-in interactions. For title animations, I had to manually split each character into individual spans, then painstakingly animate them one by one in Webflow's interface. I repeated this tedious process for navigation items and other elements throughout the site.

```
const title = document.querySelector('.title')

const split = new SplitText(title, { type: 'lines, chars' })
```

```
gsap.set(split.lines, { overflow: 'hidden' })

gsap.fromTo(split.chars,
  { yPercent: 100, opacity: 0 },
  { yPercent: 0, opacity: 1, stagger: 0.02, ease: 'power2.out' }
)
```

Just a few lines of code can replace days of manual work, making the system more flexible, easier to maintain, and faster. What once seemed like magic is now a key tool in my development toolkit.

3D and Visual Effects: Three.js

Recently, I discovered Three.js for myself. It all started with Bruno Simon's Three.js Journey. This incredible learning platform completely transformed my understanding of what's possible in web-based 3D and expanded my creative horizons. Writing custom shaders from scratch is still a significant challenge, but I've enjoyed the learning process immensely!

For my portfolio, Three.js provided the perfect toolset for creating immersive 3D effects that complement the GSAP animations. The WebGL-powered visuals add depth and interactivity that wouldn't be possible with standard DOM animations.

Seamless Page Transitions: Barba.js

I wanted my portfolio to feel like a single, fluid experience while still having regular website URLs for each section. Barba.js helped me create smooth transitions between pages instead of the usual abrupt page reloads.

Architecture Overview

The code architecture is designed around components that can be animated independently while still coordinating with each other:

```
src/
├── actions/              # Server-side form actions
├── components/           # UI components and layout elements
├── content/              # CMS-like content storage
├── js/                   # Client-side JavaScript
|   ├── App.js            # Main application entry point
|   ├── components/       # Reusable JS components
|   |   ├── 3D/           # Three.js components
|   |   ├── EventEmitter/ # Custom event system
|   |   ├── transition/   # Page transition components
|   |   └── ...
|   ├── pages/            # Page-specific logic
|   |   ├── home/         # Home page animations
|   |   |   ├── world/    # Three.js scene for home page
|   |   |   ├── Awards.js # Awards section animations
|   |   |   ├── Cube.js   # 3D cube animations
|   |   |   └── ...
|   |   ├── cases/        # Case studies animations
```

```
|   |   ├── contact/      # Contact page animations
|   |   └── error/        # 404 page animations
|   ├── transitions/      # Page transition definitions
|   └── utils/            # Helper functions
├── layouts/              # Astro layout templates
├── pages/                # Astro page routes
├── static/               # Static assets
└── styles/               # SCSS styles
```

Home Page

I guess this is where I spent most of my time 😅

#1: Hero Section

The first thing was the reel video. A plain rectangle felt monotonous, and I wasn’t really into that look. I also wanted to add some visual effects to make it more playful but not too overwhelming. So, I used the dithering effect (thanks, Maxime, for the fabulous tutorial on it) across almost the entire website for images and videos.

```
// The Bayer matrix defines the dithering pattern
const float bayerMatrix8x8[64] = float[64](
    0.0/64.0, 48.0/64.0, 12.0/64.0, 60.0/64.0, /* and so on... */
);

// Function to apply ordered dithering to a color
```

```
vec3 orderedDither(vec2 uv, vec3 color)
{
    // Find the corresponding threshold in the Bayer matrix
    int x = int(uv.x * uRes.x) % 8;
    int y = int(uv.y * uRes.y) % 8;
    float threshold = bayerMatrix8x8[y * 8 + x] - 0.88;

    // Add the threshold and quantize the color to create the dithering effect
    color.rgb += threshold;
    color.r = floor(color.r * (uColorNum - 1.0) + 0.5) / (uColorNum - 1.0);
    color.g = floor(color.g * (uColorNum - 1.0) + 0.5) / (uColorNum - 1.0);
    color.b = floor(color.b * (uColorNum - 1.0) + 0.5) / (uColorNum - 1.0);

    return color;
}

// Main shader function
void main()
{
    vec2 uv = vUv;

    // Apply pixelation effect
    float pixelSize = mix(uPixelSize, 1.0, uProgress);
    vec2 normalPixelSize = pixelSize / uRes;
    vec2 uvPixel = normalPixelSize * floor(uv / normalPixelSize);

    // Sample the video texture
    vec4 texture = texture2D(uTexture, uvPixel);

    // Calculate luminance for grayscale dithering
    float lum = dot(vec3(0.2126, 0.7152, 0.0722), texture.rgb);

    // Apply dithering to the luminance
    vec3 dither = orderedDither(uvPixel, vec3(lum));

    // Mix between dithered and original based on hover progress
    vec3 color = mix(dither, texture.rgb, uProgress);

    // Final output with alpha handling
    gl_FragColor = vec4(color, alpha);
}
```

Also, some GSAP magic to animate different states.

```
this.item.addEventListener('mouseenter', () =>
{
    gsap.to(this.material.uniforms.uProgress,
    {
        value: 1,
        duration: 1,
        ease: 'power3.inOut'
    })
})

// When the user moves away
this.item.addEventListener('mouseleave', () =>
{
    gsap.to(this.material.uniforms.uProgress,
    {
        value: this.videoEl.muted ? 0 : 1,
```

```
            duration: 1,
            ease: 'power3.inOut'
        })
    })
```

#2: Falling Text – Breaking Conventions with Physics

This is my favorite part of the website—a nontraditional approach to the ubiquitous "about me" section. Instead of a standard paragraph, I presented my story through a physics-driven text animation that breaks apart as you engage with it.

The text combines my professional background with personal insights—from my journey as an athlete to my love for gaming and travel. I deliberately highlighted "Creative Developer" in a different color to emphasize my professional identity, creating visual anchors throughout the lengthy text.

What makes this section technically interesting is its layered implementation:

First, I use GSAP’s SplitText plugin to break the text into individual elements:

```
// Split text into words, then specific words into characters
this.split = new SplitText(this.text,
{
    type: 'words',
    wordsClass: 'text-words',
    position: 'absolute'
})

this.spans = this.wrapper.querySelectorAll('span')
this.spanSplit = new SplitText(this.spans,
```

```
    {
        type: 'chars',
        charsClass: 'text-char',
        position: 'relative'
    })
```

Next, I create a physics world using `Matter.js` and add each character as a physics body.

```
// Create a physics body for each character
this.splitToBodies.forEach((char, index) =>
{
    const rect = char.getBoundingClientRect()
    const box = Matter.Bodies.rectangle(
        rect.left + rect.width / 2,
        rect.top + rect.height / 2,
        rect.width,
        rect.height,
        {
            isStatic: false,
            restitution: Math.random() * 1.2 + 0.1 // Random bounciness
        }
    );

    Matter.World.add(this.world, box)
    this.bodies.push({ box, char, top: rect.top, left: rect.left })
})
```

When triggered by scrolling, the `enterFalling()` method activates the physics simulation with random forces.

```
enterFalling()
{
    // Enable physics rendering
    this.allowRender = true

    // Add a class for CSS transitions
    this.fallingWrapper.classList.add('falling')

    // Apply slight random forces to each character
```

```
        this.bodies.forEach(body =>
        {
            const randomForceY = (Math.random() - 0.5) * 0.03
            Matter.Body.applyForce(
                body.box,
                { x: body.box.position.x, y: body.box.position.y },
                { x: 0, y: randomForceY }
            )
        })
    }
```

Finally, I use ScrollTrigger to activate the physics simulation at the correct scroll position.

```
this.fallingScroll = ScrollTrigger.create(
{
    trigger: this.wrapper,
    start: `top top-=${this.isDesktop ? 375 : 300}%`,
    onEnter: () => this.enterFalling(),
    onLeaveBack: () => this.backFalling()
})
```

When the physics is activated, each character responds independently to gravity and collisions, creating an organic falling effect that transforms the formal text into a playful, interactive element. As users continue to scroll, they can even "push" the text away with scroll momentum —a detail that rewards experimentation.

#3: The Interactive Cube

Following the physics-driven text animation, visitors encounter one of my portfolio's most technically complex elements: the 3D cube that showcases my awards and achievements. This section transforms what could have been a simple list into an interactive 3D experience that invites exploration.

The cube is built using a combination of pure CSS 3D transforms and GSAP for animation control. The basic structure consists of a container with six faces, each positioned in 3D space:

```
// Setting up the 3D environment
this.cube = this.main.querySelector('.cube')
this.parts = this.main.querySelectorAll('.cube_part')
this.tiles = this.main.querySelectorAll('.cube_tile')

// Creating a timeline for the cube transformation
this.tl = gsap.timeline({paused: true})
```

```
// Animating the cube parts with precise timing
this.tl.to(this.parts, {'--sideRotate': 1, stagger: 0.2}, 2)
```

The real magic happens when `ScrollTrigger` comes into play, controlling the cube's rotation based on the scroll position.

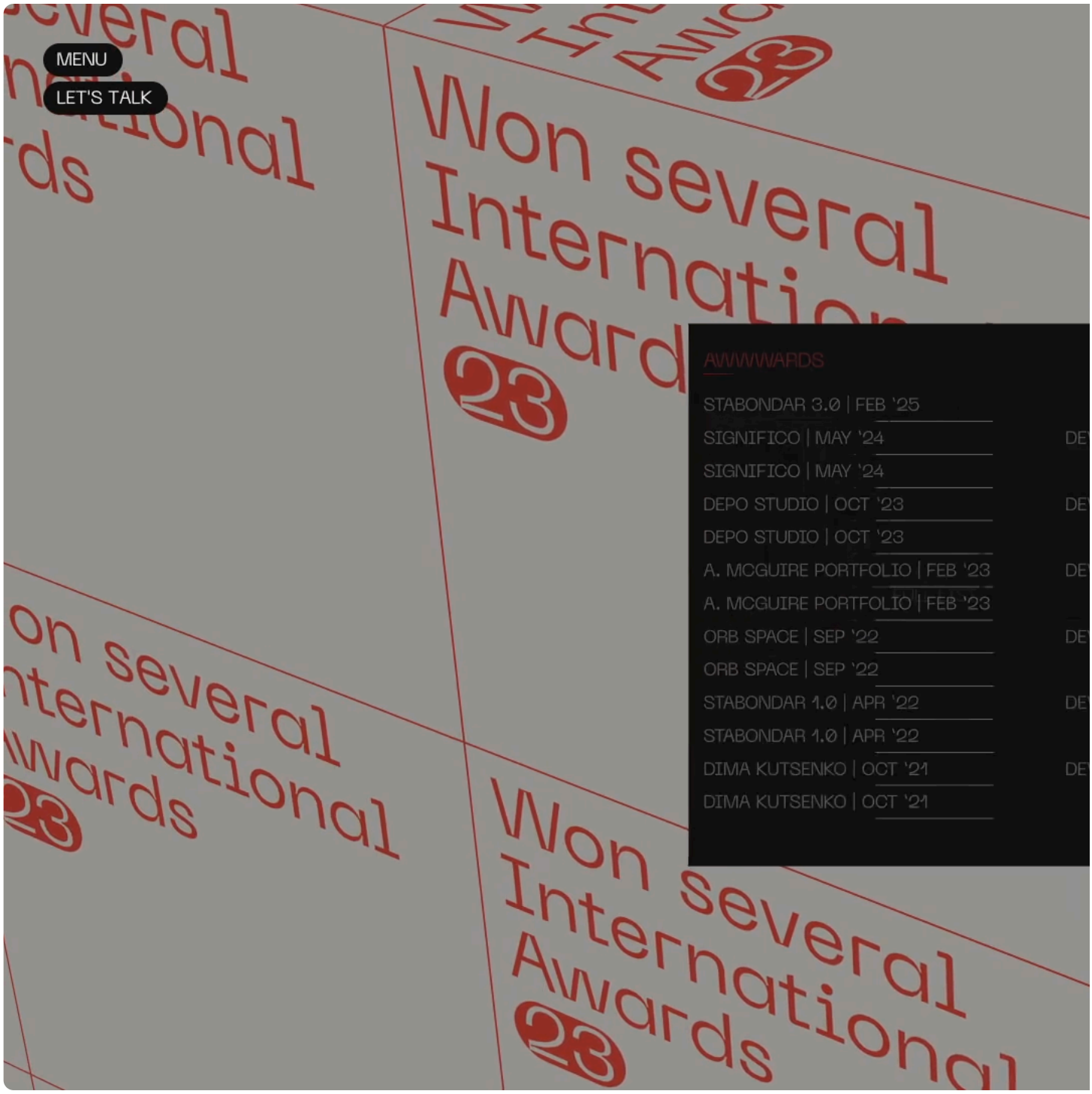
```
this.scrollRotate = ScrollTrigger.create(
{
    trigger: this.wrapper,
    start: 'top top',
    end: 'bottom bottom',
    scrub: true,
    onUpdate: (self) =>
    {
        const progress = self.progress;
        const x = progress * 10;
        const y = progress * 10;
        this.values.scrollX = x;
        this.values.scrollY = y;
    }
})
```

One of my favorite GSAP utilities that deserves more recognition is gsap.quickTo(). This function has been a game-changer for handling continuous animations, especially those driven by mouse movement or scroll position, like the cube's rotation.

Instead of creating and destroying hundreds of tweens for each frame update (which would cause performance issues), I use `quickTo` to create reusable animation functions:

```
// Create re-usable animation functions during initialization
this.cubeRotateX = gsap.quickTo(this.cube, '--rotateY', {duration: 0.4})
this.cubeRotateY = gsap.quickTo(this.cube, '--rotateX', {duration: 0.4})

// In the update function, simply call these functions with the new values
update()
{
    this.cubeRotateX(this.axis.y + this.values.mouseX + this.values.scrollX)
    this.cubeRotateY(this.axis.x - this.values.mouseY + this.values.scrollY)
}
```

The difference is profound. Instead of creating and destroying hundreds of tweens per second, I'm simply updating the target values of existing tweens. The animation remains smooth even during rapid mouse movements or complex scroll interactions.



This approach is used throughout my portfolio for performance-critical animations:

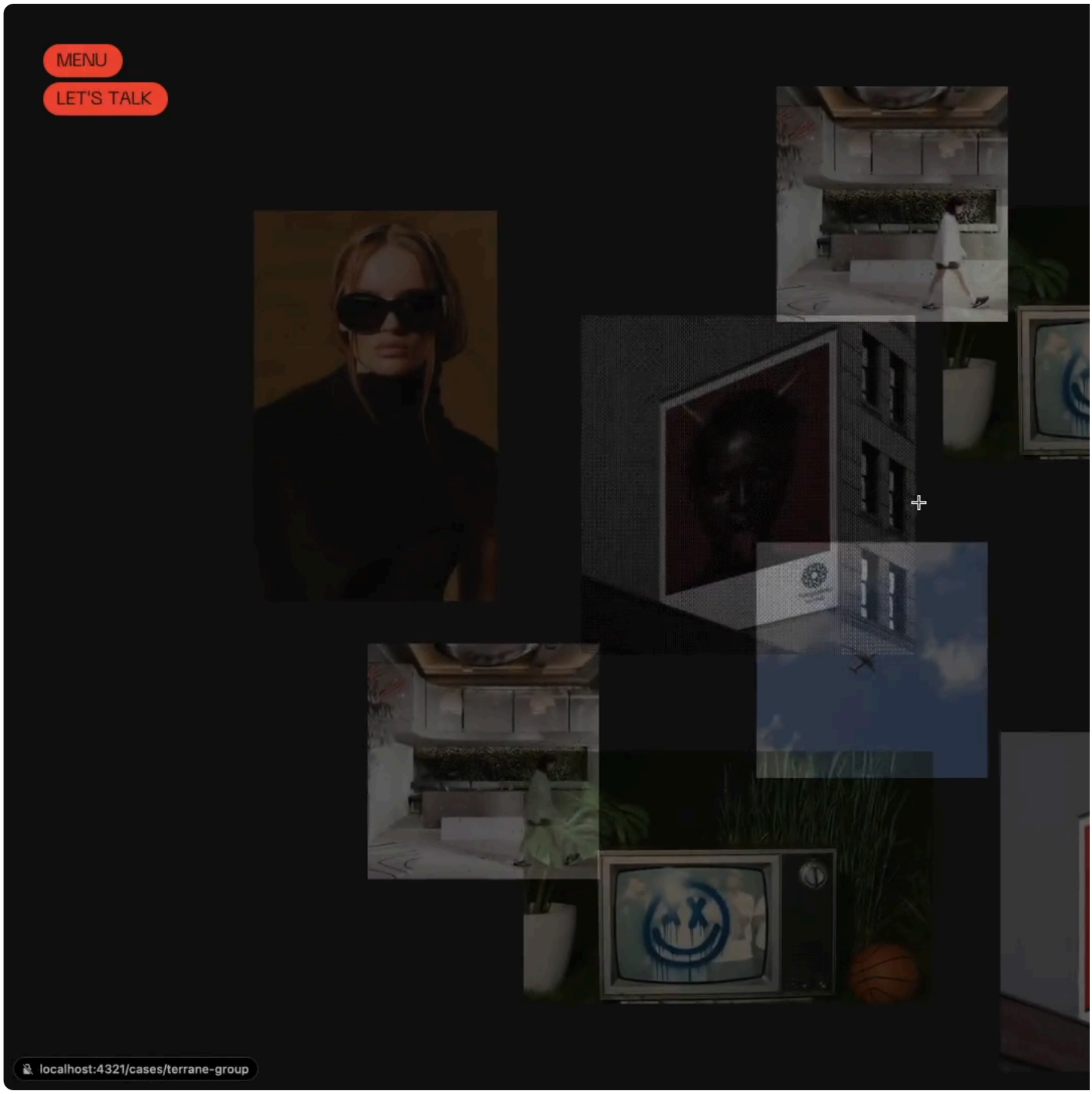
```
 For the awards cursor
is.quickX = gsap.quickTo(this.section, '--x', {duration: 0.2, ease: 'power2.out'})
is.quickY = gsap.quickTo(this.section, '--y', {duration: 0.2, ease: 'power2.out'})
```

Beyond scroll control, the cube also responds to mouse movement for an additional layer of interactivity:

```
window.addEventListener('mousemove', (e) =>
{
    if(!this.isInView) return

    const x = (e.clientX / window.innerWidth - 0.5) * 4
    const y = (e.clientY / window.innerHeight - 0.5) * 4

    this.values.mouseX = x
    this.values.mouseY = y
})
```

#4: Projects Grid with WebGL Enhancements

The final section of the homepage features a grid of my projects, each enhanced with WebGL effects that respond to user interaction. At first glance, what appears to be a standard portfolio grid is actually a sophisticated blend of Three.js and GSAP animations working together to create an immersive experience.



Each project thumbnail features a custom dithering shader similar to the one used in the hero video but with additional interactive behaviors:

```
this.imgsStore.forEach(({img, material}) =>
{
    const item = img.parentElement

    item.addEventListener('mouseenter', () =>
    {
        gsap.to(material.uniforms.uHover, {value: 1, duration: 0.4})
    })
```

```
    item.addEventListener('mouseleave', () =>
    {
        gsap.to(material.uniforms.uHover, {value: 0, duration: 0.4})
    })
})
```

The shader itself applies ordered dithering using a Bayer matrix to create a distinctive visual style:

```
vec3 orderedDither(vec2 uv, vec3 color)
{
    float threshold = 0.0

    int x = int(uv.x * uRes.x) % 8
    int y = int(uv.y * uRes.y) % 8
    threshold = bayerMatrix8x8[y * 8 + x] - 0.88

    color.rgb += threshold
    color.r = floor(color.r * (uColorNum - 1.0) + 0.5) / (uColorNum - 1.0)
    color.g = floor(color.g * (uColorNum - 1.0) + 0.5) / (uColorNum - 1.0)
    color.b = floor(color.b * (uColorNum - 1.0) + 0.5) / (uColorNum - 1.0)

    return color
}
```

Enhancing the sense of depth, each project image moves at a slightly different rate as the user scrolls, creating a parallax effect:

```
this.scrollTl = gsap.timeline({defaults: {ease: 'none'}})

this.imgsStore.forEach(({img}, index) =>
{
    const random = gsap.utils.random(0.9, 1.1, 0.05)

    this.scrollTl.fromTo(img.parentElement,
        { '--scrollY': 0 },
        { '--scrollY': -this.sizes.height / 2 * random }, 0)
})

ScrollTrigger.create(
{
    trigger: this.section,
    start: 'top top',
    end: 'bottom bottom-=100%',
    scrub: true,
    animation: this.scrollTl,
})
```

I use another GSAP utility method to randomize the movement speeds (`random = gsap.utils.random(0.9, 1.1, 0.05)`) ensuring that each project moves at a slightly different rate, creating a more natural and dynamic scrolling experience.

Beyond scroll effects, the projects also respond to mouse movement using the same `quickTo` technique employed in the cube section:

```
parallax()
{
    this.mouse = {x: 0, y: 0}

    window.addEventListener('mousemove', (e) =>
    {
        this.mouse.x = e.clientX - window.innerWidth / 2
        this.mouse.y = e.clientY - window.innerHeight / 2

        this.items.forEach((item, index) =>
        {
            const quickX = this.quicksX[index]
            const quickY = this.quicksY[index]

            const x = this.mouse.x * 0.6 * -1 * this.ramdoms[index]
            const y = this.mouse.y * 0.6 * -1 * this.ramdoms[index]

            quickX(x)
            quickY(y)
        })
    })
}
```

Each project thumbnail moves in response to the mouse position but at slightly different rates, based on the same random values used for scroll parallax. This creates a cohesive sense of depth that seamlessly blends scrolling and mouse interactions.

The most impressive aspect of this section is how the projects transition into their respective case studies. When a user clicks a project, Barba.js and GSAP’s Flip plugin work together to create a smooth visual transition:

```
this.state = Flip.getState(this.currentImg)
this.nextContainerImageParent.appendChild(this.currentImg)

this.flip = Flip.from(this.state,
{
    duration: 1.3,
    ease: 'power3.inOut',
```

```
        onUpdate: () => this.update(this.flip.progress())
    })
```

This creates the illusion that the thumbnail is morphing into the hero image of the case study page, providing visual continuity between pages.

Cases Page

After exploring the innovative elements of the homepage, let’s dive into the Cases page—my absolute favorite section of the portfolio. This page represents countless hours of experimentation and refinement. I still get lost in the interactions, moving the cursor around and scrolling columns independently just to see the visual effects in motion.

The Cases page showcases projects in a multi-column layout that adapts based on viewport size and responds to touch and drag through GSAP’s Draggable plugin. One of its most striking aspects is that each column operates independently of the others in terms of scroll behavior.

The number of visible columns is dynamically determined based on screen width:

```
getColLength()
{
    let colLength = 0
    if(this.sizes.width > this.breakpoint.tablet)
    {
```

```
            colLength = 5  // Desktop view
        }
        else if(this.sizes.width > this.breakpoint.mobile)
        {
            colLength = 3  // Tablet view
        }
        else
        {
            colLength = 1  // Mobile view
        }
        return colLength
    }
```

Each column maintains its own scrolling context, resulting in a visually engaging grid that invites exploration.

```
    init(splitItem, randomScrollPosition)
    {
        let start = window.innerWidth < this.breakpoint.mobile ? 100 : randomScrollPos
        let scrollSpeed = 0
        let oldScrollY = start
        let scrollY = start

        // Initialize each column with a random scroll position
        const index = Array.from(splitItem.parentElement.children).indexOf(splitItem)
        const list = splitItem
        const item = list.querySelectorAll('.project_item')

        // Set up mouse wheel interaction for each column
        list.addEventListener('wheel', (e) => scrollY = this.handleMouseWheel(e, scrol

        // Add touch and drag capabilities through GSAP's Draggable
        Draggable.create(splitItem,
        {
            type: 'y',
            inertia: true,
            onDrag: function()
            {
                gsap.set(splitItem, {y: 0, zIndex: 1})
                scrollY += this.deltaY * 0.8
            }
        })
    }
```

One of the most distinctive aspects of the Cases page is how project thumbnails react to scrolling with realistic, physics-based distortions.

The scroll velocity is calculated by tracking the difference between the current and previous scroll positions:

```
update()
{
    y = this.lerp(y, scrollY, 0.1)
    scrollSpeed = y - oldScrollY

    if(Math.floor(oldScrollY) != Math.floor(y))
    {
        this.updateScroll(y, item, numItems, itemHeight, wrapHeight, numItemHeight
        this.projectScrollSpeed[index] = Math.abs(scrollSpeed)
```

```
        }

        oldScrollY = y
    }
```

The scroll velocity is then passed to the fragment shader to generate dynamic distortion effects.

```
// Fragment shader excerpt showing scroll-velocity based distortion
float speed = clamp(uScrollSpeed, 0.0, 60.0)
float normalizedSpeed = clamp(uScrollSpeed / 20.0, 0.0, 1.0)
float invertStrength = pow(normalizedSpeed, 1.2)

float area = smoothstep(0.3, 0., vUv.y)
area = pow(area, 2.0)

float area2 = smoothstep(0.7, 1.0, vUv.y)
area2 = pow(area2, 2.0)

area += area2

uv.x -= (vUv.x - uLeft) * 0.1 * area * speed * 0.1
vec4 displacement = texture2D(uDisplacement, uv)
uv -= displacement.rg * 0.005
```

Case Study Page

Moving beyond the dynamic layout of the Cases page, each individual case study offers a unique opportunity to highlight not only the final results of client projects but also the specific technical approaches and animations used in those projects. Instead of applying a uniform template to all case studies, I designed each detail page to reflect the unique techniques and visual language of the original project.

Unique Animation Systems for Each Project

Every case study page features animations and interactions that mirror those I developed for the client project. This approach serves two purposes: it demonstrates the techniques in a practical context and allows visitors to experience the project’s interactive elements firsthand.

```
// Example from Runway case study page - atmospheric cloud simulation
init()
{
    this.sky = await import('./meshs/sky')
    this.sky = new this.sky.default(this.app, this.main, this.gl, this.resources)

    this.skyOutput = await import('./meshs/skyOutput')
```

```
        this.skyOutput = new this.skyOutput.default(this.app, this.main, this.gl)
    }
```

For example, the Runway case study showcases a cloud simulation technique developed specifically for that project. In contrast, the Bulletproof case features dynamic SVG morphing transitions that are central to its design.

Current Section Navigation

One of the most technically intriguing components across all case studies is the current section highlighter, which visually indicates the user’s position within the case study content.

The section tracker works by monitoring scroll positions and updating a navigation indicator accordingly:

```
init()
{
    this.scrolls = []

    this.sections.forEach((section, index) =>
    {
        const currentText = section.getAttribute('current-section')
        const prevSection = this.sections[index - 1]
        const prevText = prevSection ? prevSection.getAttribute('current-section')
```

```
            const nextSection = this.sections[index + 1]
            const nextText = nextSection ? nextSection.getAttribute('current-section')

            const trigger = ScrollTrigger.create(
            {
                trigger: section,
                start: 'top 50%',
                end: 'bottom 50%',
                onEnter: () =>
                {
                    if(!prevSection && currentText !== this.currentText)
                    {
                        this.changeTitle(currentText)
                    }
                    else if(prevText !== currentText && currentText !== this.currentTe
                    {
                        this.changeTitle(currentText)
                    }
                },
                onEnterBack: () =>
                {
                    if(!nextSection && currentText !== this.currentText)
                    {
                        this.changeTitle(currentText)
                    }
                    else if(nextText !== currentText && currentText !== this.currentTe
                    {
                        this.changeTitle(currentText)
                    }
                },
            })

            this.scrolls.push(trigger)
        })
    }
```

This system creates `ScrollTrigger` instances for each content section, detecting when a section enters or leaves the viewport. When this happens, the system updates the navigation title to reflect the current section.

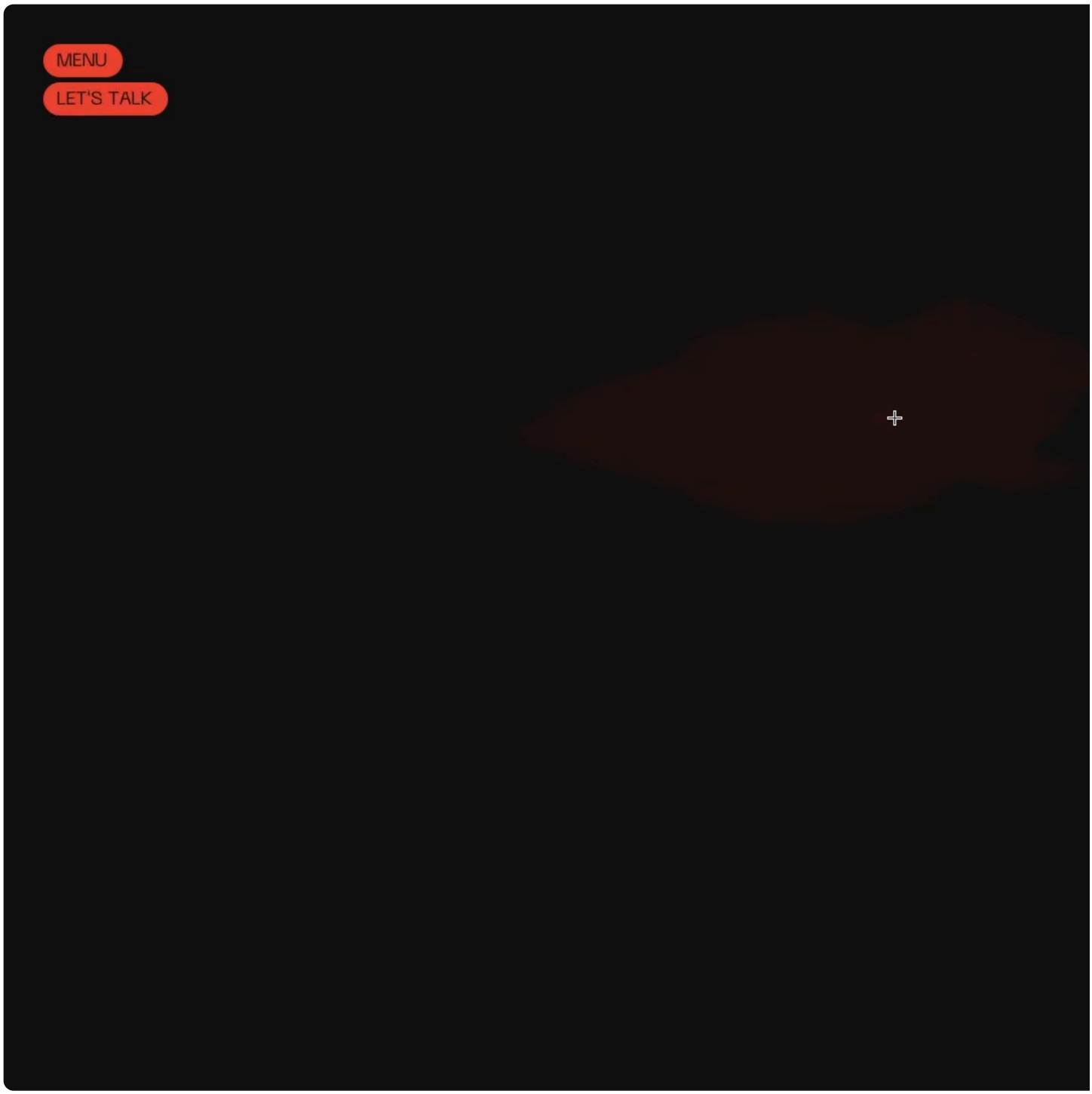
What makes this system particularly engaging is the text transition animation that occurs when switching between sections:

```
    changeTitle(text)
    {
        if(this.split) this.split.revert()

        this.navTitle.innerHTML = text
        this.split = new SplitText(this.navTitle, {type: 'chars, lines', charsClass: '

        this.animation = gsap.timeline()

        this.split.lines.forEach((line, index) =>
        {
            const chars = line.querySelectorAll('.char')

            this.animation.from(chars, this.charsAnimation, index * 0.1)
            .from(chars, this.charsScrumble, index * 0.1)
        })
```

```
        this.currentText = text
    }
```

Rather than simply updating the text, the system creates a visually engaging transition using GSAP’s SplitText plugin. The text characters scramble and fade in, creating a dynamic typographic effect that draws attention to the section change.

Thank You for Reading!

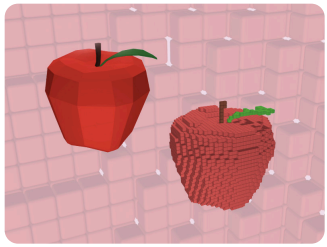


Thank you for taking the time to explore this project with me. Happy coding, and may your own creative endeavors be both technically fascinating and visually captivating!

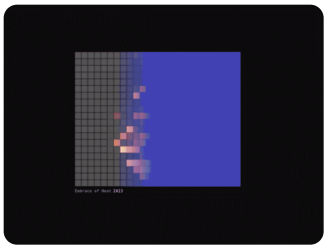
Stas Bondar

Creative Developer

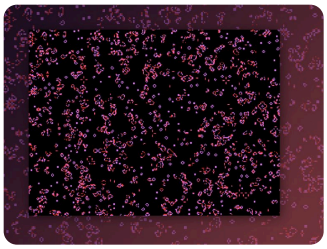
3d astro case-study gsap three-js



Turning 3D Models to Voxel Art with Three.js



On-Scroll Revealing WebGL Images



Conway’s Game Of Life – Cellular Automata and Renderbuffers in Three.js