BATTLE FORCE

A Technical Case Study in Web-Based 3D Game Development

Course: Computer Graphics

Team: Ahmed Mesbah Ahmed & Ziad Ahmed Al-Junaidi

The Objective: Survive the Horde

BATTLE FORCE is a first-person shooter where players defend against endless waves of increasingly difficult zombie hordes in a post-apocalyptic arena.

Primary Goal: Survive as long as possible by eliminating all zombies in each wave, managing resources, and achieving the highest score.

Progression System

- **Wave-Based**: Each wave increases in difficulty, starting with 10 zombies.
- **Boss Waves**: Every 5th wave features a significant increase in enemy count and variety.

The Arsenal: Four Tools of Survival

Players have access to a diverse arsenal, each with distinct tactical advantages.

Pistol

Damage: 25 HP

Magazine: 12 rounds

Role: Reliable sidearm

Rifle

Damage: 35 HP

Magazine: 30 rounds

Role: Automatic assault, 80ms fire rate

Shotgun

Damage: 8 pellets/shot

Magazine: 6 rounds

Role: Close-range devastator, spread pattern

Sniper Rifle

Damage: 100 HP

Magazine: 5 rounds

Role: High-precision, slow fire rate

The Enemy: A Diverse Undead Threat

The horde is composed of four distinct zombie archetypes, each requiring a different tactical approach.

Normal Zombie

- 100 HP
- Standard Speed
- Base threat

Fast Zombie

- 50 HP
- High Speed
- Evasive zigzag movement

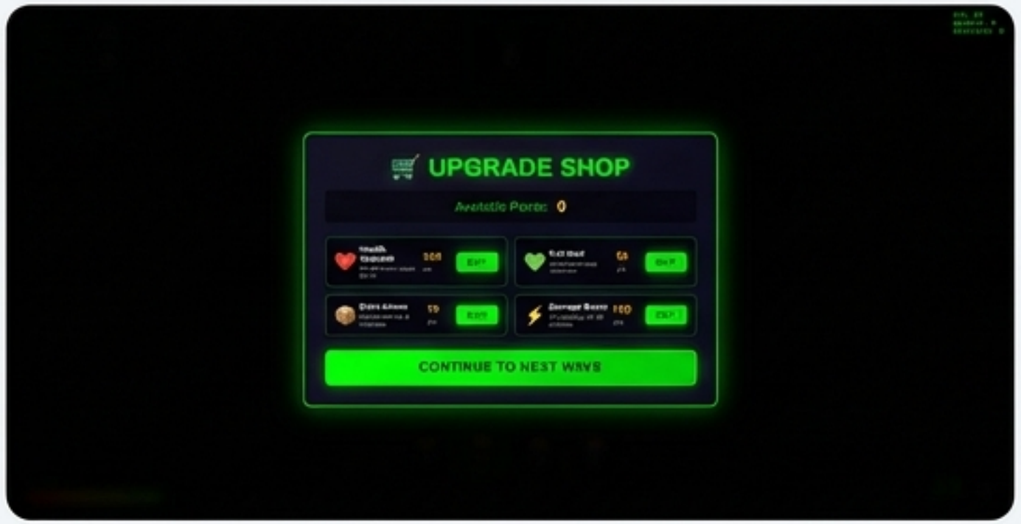
Tank Zombie

- 300 HP
- Slow Speed
- Heavily armored “bullet sponge”

Exploder Zombie

- 75 HP
- Standard Speed
- Detonates on death or proximity, causing area damage

Engineering Engagement: Scoring, Strategy, and Upgrades



Dynamic Scoring System

- **Headshot Bonus:** 2x points and 2x damage for precision shots.
- **Combo Multiplier:** Up to 3x multiplier for rapid kills within 3 seconds.

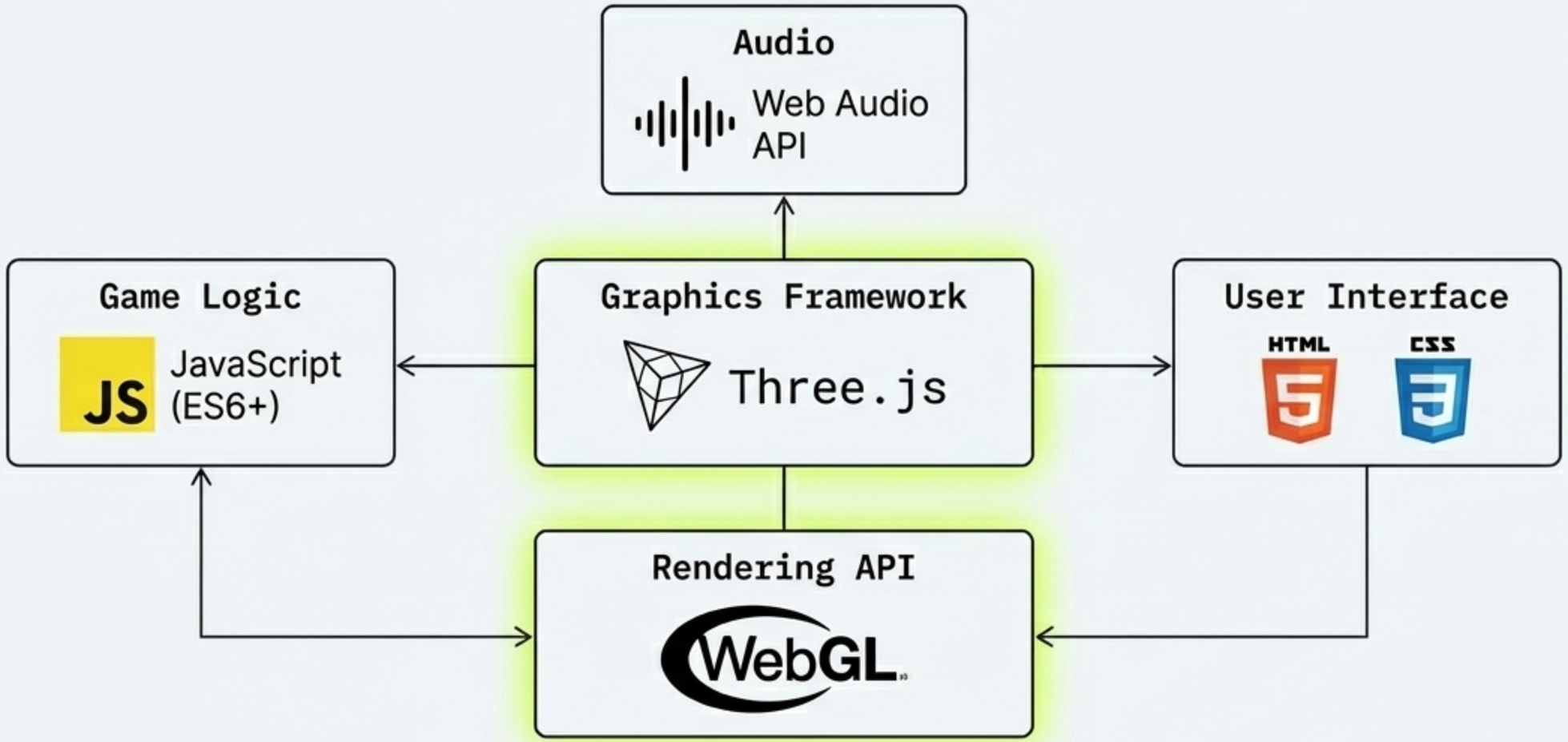
Interactive Environment

- **Cover:** Destructible wooden crates and indestructible metal barrels.
- **Hazards:** Explosive barrels that deal 75 damage in a blast radius.

Player Progression

Between waves, players spend points in an Upgrade Shop for health, ammo, and temporary damage boosts.

The Engine: A Modern 3D Game Built for the Web



The entire experience is delivered in-browser, requiring no installation and leveraging hardware-accelerated graphics via standard web technologies.

Crafting Atmosphere with a Multi-Layered Lighting System

A sophisticated, four-component lighting model creates depth, enhances 3D perception, and establishes the game's mood.

1. Ambient Light

`THREE.AmbientLight` (Intensity: 0.4) provides global base illumination, ensuring no object is ever completely black.

2. Directional Light

`THREE.DirectionalLight` (Intensity: 0.8) simulates a primary sun source, casting shadows for depth perception.

3. Hemisphere Light

`THREE.HemisphereLight` (Intensity: 0.6) creates a natural sky-to-ground color gradient (light blue to dark gray).

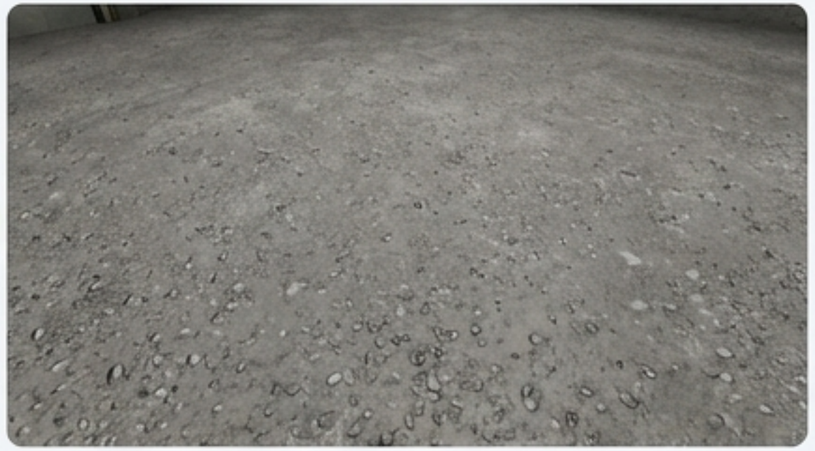
4. Dynamic Point Lights

Short-lived, orange point lights are spawned for muzzle flashes, providing immediate visual feedback for player actions.

Code as Evidence: Procedural Texture Generation

To create detailed and varied surfaces without large texture files, we generated textures procedurally using the HTML5 Canvas API. This approach is memory-efficient and highly customizable.

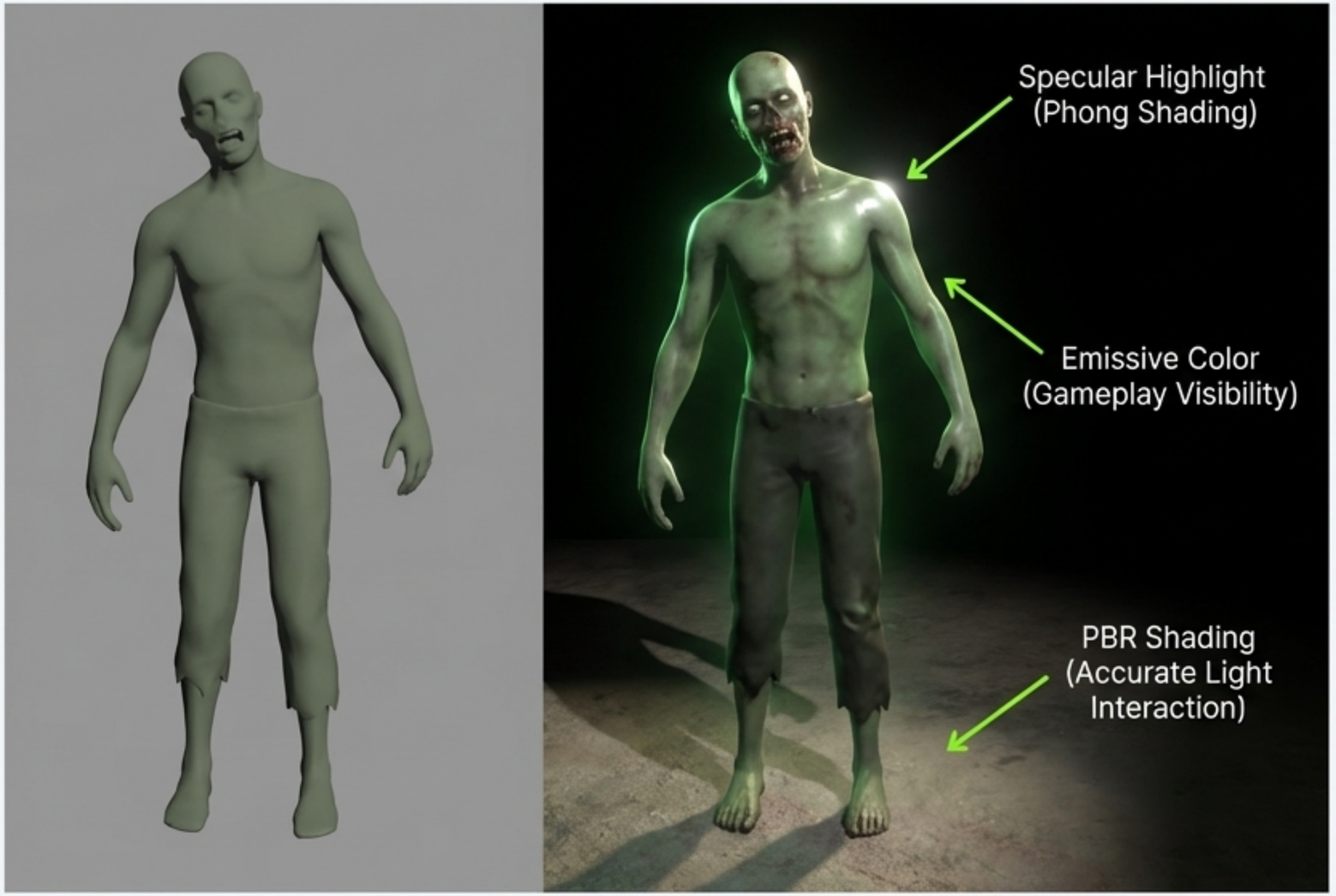
```
createConcreteTexture() {
  const canvas = document.createElement('canvas');
  canvas.width = 512; canvas.height = 512;
  const ctx = canvas.gatContext('2d');
  ctx.fillStyle = '#666666';
  ctx.fillRect(0, 0, 512, 512);
  for (let i = 0; i < 5000; i++) {
     const x = Math.random() * 512;
     const y = Math.random() * 512;
     const brightness = Math.random() * 50 - 25;
     ctx.fillStyle = `rgb(${102+brightness}, ${102+brightness}, ${102+brightnes
     ctx.fillRect(x, y, 2, 2);
  }
  return new THREE.CanvasTexture(canvas);
```



```
createSkinTexture(baseColor) {
  const canvas = document.createElement('canvas');
  canvas.width = 256; canvas.height = 256;
  const ctx = canvas.getContext('2d');
  const r = (baseColor >> 16) & 255;
  const g = (baseColor >> 8) & 255;
  const b = baseColor & 255;
  ctx.fillStyle = `rgb(${r}, ${g}, ${b})`;
  ctx.fillRect(0, 0, 256, 256);
  for (let i = 0; i < 1000; i++) {
     const x = Math.random() * 256;
     const y = Math.random() * 256;
     const variation = Math.random() * 30 - 15;
     ctx.fillStyle = `rgb(${r+variation}, ${g+variation}, ${b+variation})`;
     ctx.fillRect(x, y, 1, 1);
  }
  return new THREE.CanvasTexture(canvas);
```

Rendering the World: Materials and Shaders

Leveraging Three.js's built-in shader programs for high-quality, real-time rendering.



Shading Models Used

- **Phong Shading** (`MeshPhongMaterial`): Used for most game objects. This model calculates ambient, diffuse, and specular components on a per-pixel basis for smooth, realistic highlights.
- **PBR Shading** (`MeshStandardMaterial`): Used for the ground. This Physically Based Rendering approach provides more accurate light interaction using `metalness` and `roughness` properties.

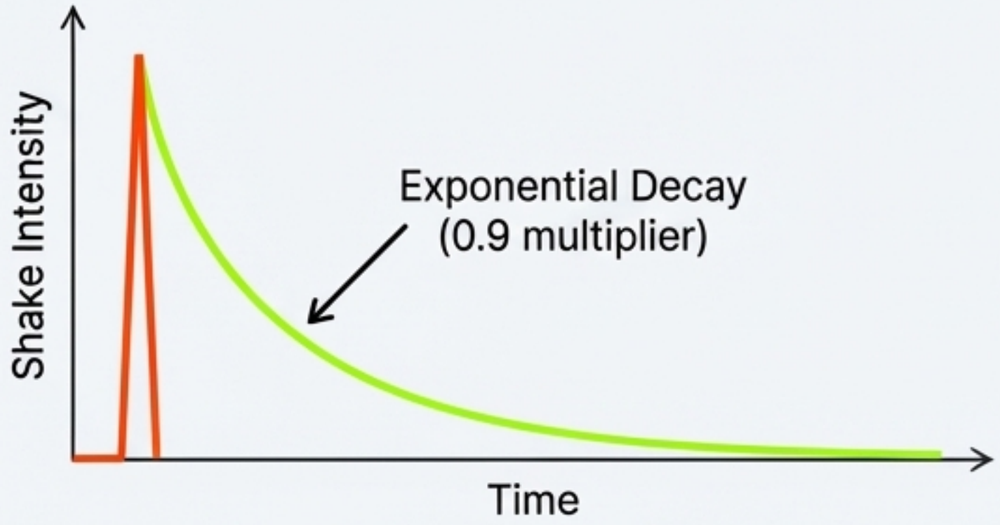
Gameplay Enhancement: Emissive Materials

- Zombies have an emissive color property, making them "glow" subtly and improving visibility in dark areas.
- Each zombie type has a unique emissive color: Normal (dull green), Fast (red), Exploder (orange).

Engineering Player Feedback: Dynamic Camera & Particles

First-Person Camera System

- **Mouse-Based Control**: Full 360° yaw and clamped pitch (-89° to +89°) for intuitive aiming.
- **Dynamic Camera Shake**: Implemented for shooting and explosions to provide haptic-like feedback. Shake intensity is event-driven and dampens smoothly using an exponential decay (0.9 multiplier).

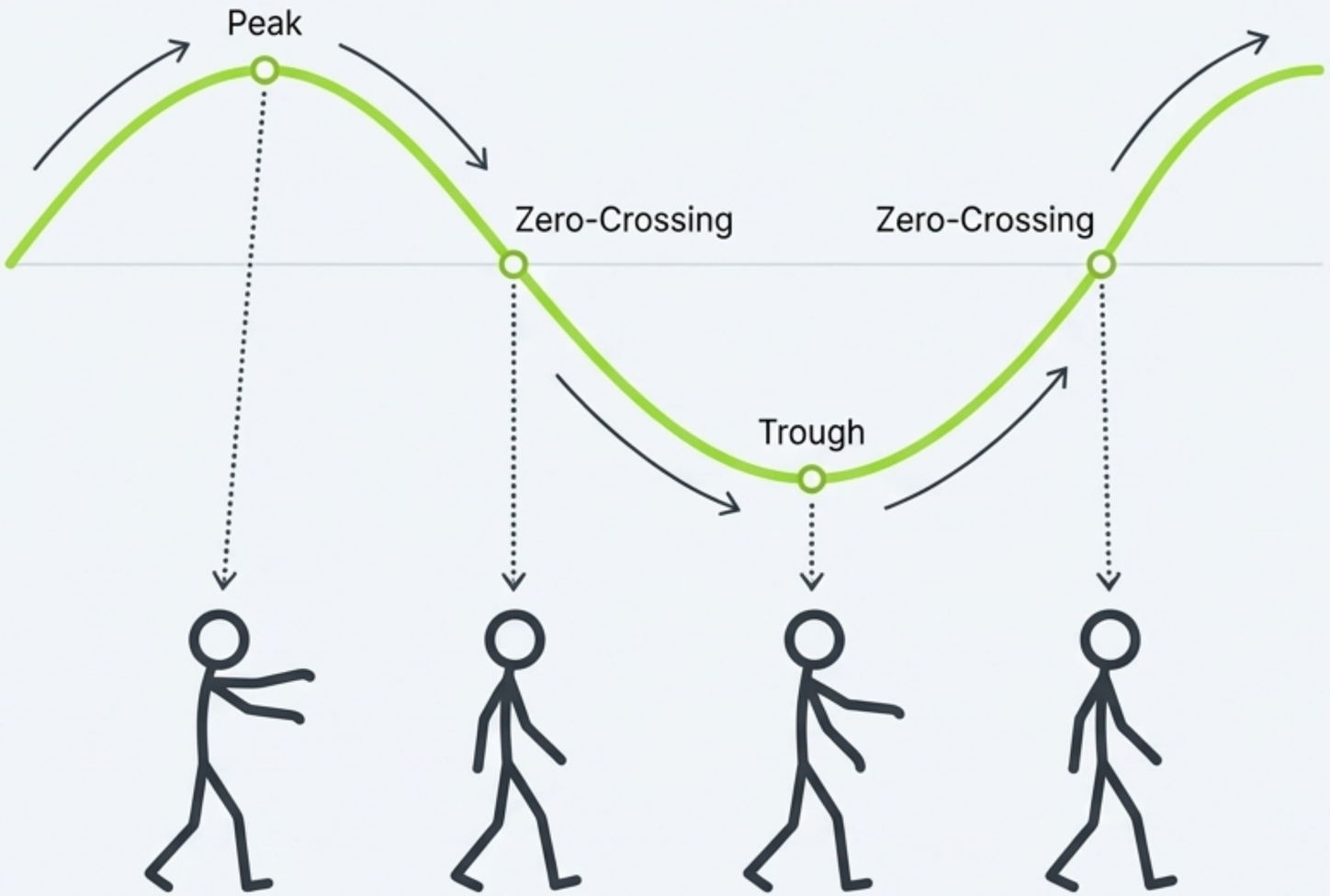


High-Performance Particle Systems

- **Muzzle Flash**: Bright, short-lived particles at the weapon barrel, accompanied by a point light.
- **Blood Splatter**: Dark red particle bursts at hit locations, influenced by gravity and drag.
- **Explosions**: Large radial bursts of orange/yellow particles.
- **Optimization**: Particle pooling is used to maintain performance by reusing particles instead of creating and destroying them.

Bringing the Horde to Life with Procedural Animation

Instead of using pre-baked skeletal animations, zombie movement is generated procedurally in real-time using trigonometric functions. This method is lightweight and dynamically tied to gameplay variables.



How It Works

A series of `Math.sin()` functions control the limbs.

- **Head Bobbing:** A sine wave applied to the head's Y-position and Z-rotation.
- **Arm/Leg Swing:** Opposing sine waves (with a `Math.PI` phase shift) create a natural alternating motion for arms and legs.
- **Dynamic Speed**: The animation frequency is directly proportional to the zombie's movement speed.

```
// Arm swinging
const bobSpeed = zombie.speed * 0.05;
zombie.mesh.children[2].rotation.x =
  Math.sin(Date.now() * bobSpeed) * 0.3;
zombie.mesh.children[3].rotation.x =
  Math.sin(Date.now() * bobSpeed + Math.PI) * 0.3;
```

Meet the Architects

Ahmed Mesbah Ahmed

Areas of Ownership:

- Core Gameplay & Combat Systems (Game Loop, Weapons, Raycasting)
- Zombie AI & Wave Mechanics (Pathfinding, Spawning, Unique Behaviors)
- Particle Engine & Performance Optimization (Pooling, FPS Monitoring)

Ziad Ahmed Al-Junaidi

Areas of Ownership:

- 3D Modeling, Scene & Environment Construction
- Procedural Textures, Materials & Lighting System
- UI/HUD Design, Audio Engineering & Animations (Player Feedback)

Conclusion: A Professional 3D Shooter on the Open Web

Project Summary

BATTLE FORCE successfully demonstrates a comprehensive application of computer graphics principles, delivering a polished, performant, and engaging FPS experience entirely within a standard web browser using Three.js and WebGL.

Key Technical Achievements

- **Real-time 3D rendering** with a multi-layered lighting system.
- Efficient **procedural generation** for textures and animations.
- Robust **player feedback** through dynamic camera and particle effects.
- **Complex gameplay mechanics** with varied enemies and a progression system.

Future Scope

The modular architecture allows for future extensions, such as shadow mapping, post-processing effects, new weapons, or multiplayer functionality.