COMP4422 Computer Graphics – Group Project Report

Project Title: Star Collector

Group 8

Workload declaration table

Name	SID	Key tasks
HUANG Boxuan	23100165D	Atmosphere, fox
Liu Limeng	20101524D	Mountain
Mo Fengyuan	25011254X	Stars

1. Story

"Star Collector" tells a story about a fox collecting stars. The narrative progresses through a 30-second timeline as follows:

0s— The fox stands on the top of a hill, breathing gently, looking up at the stars, time change into night.

11-17s The first star slowly descends from the sky, and disappears in flash. Then the fox tilted its head slightly.

26-32s The second star descends, and the fox's tail gently sways.

33s— The camera zooms out, and the third star appears, emitting particle-like starlight upwards.

2. Scene Design

At the center of the scene is a small hill, and the fox is standing under a tree on the hill. The surrounding mountains undulate irregularly, with sparse trees scattered on top. The sky transitions from dusk to night, with stars on it.

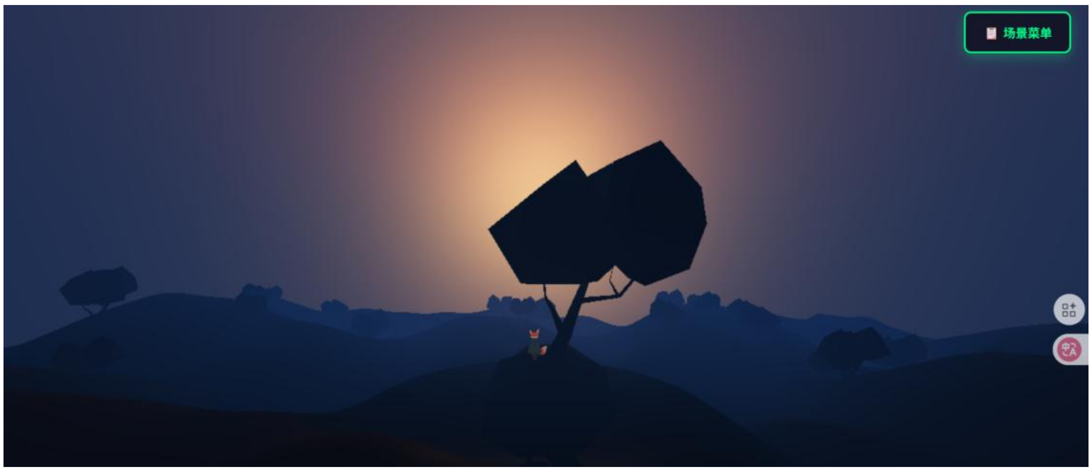
3. Object models

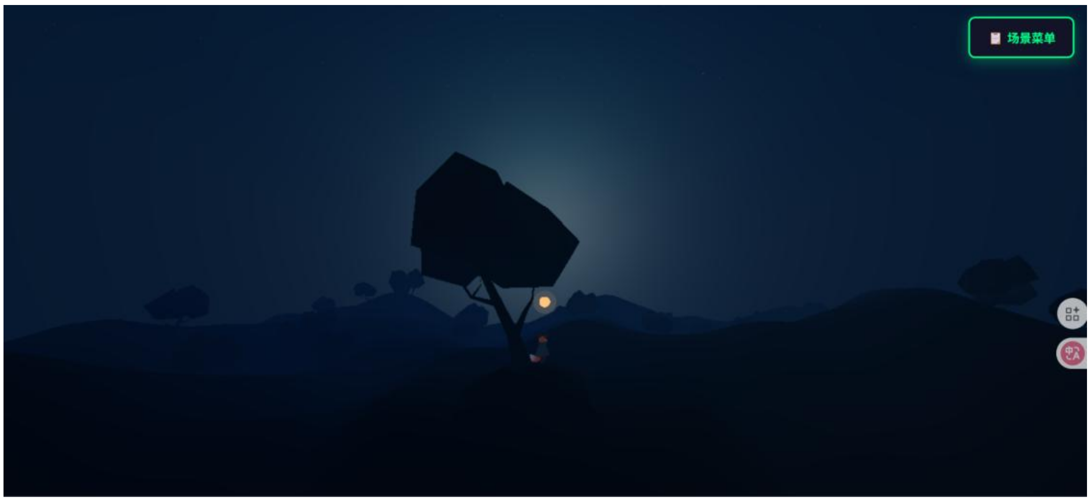
- **Fox model:** A GLTF fox model downloaded online, with 11.1k triangles and including skeleton rigging.
- **Tree-hill model:** A GLTF tree-hill terrain resource downloaded online, containing 746 triangles.
- **Terrain model:** A mountain terrain generated using Perlin noise.
- **Star model:** A double-layer luminous sphere, with the outer layer simulating glow.
- **Star particles:** Luminous spheres.

4. Visuals & Rendering Techniques

4.1 Lighting

- Implemented an environment and directional light system that changes with time, enabling the simulation of sunlight and moonlight.
- Achieved the glowing effects of stars.



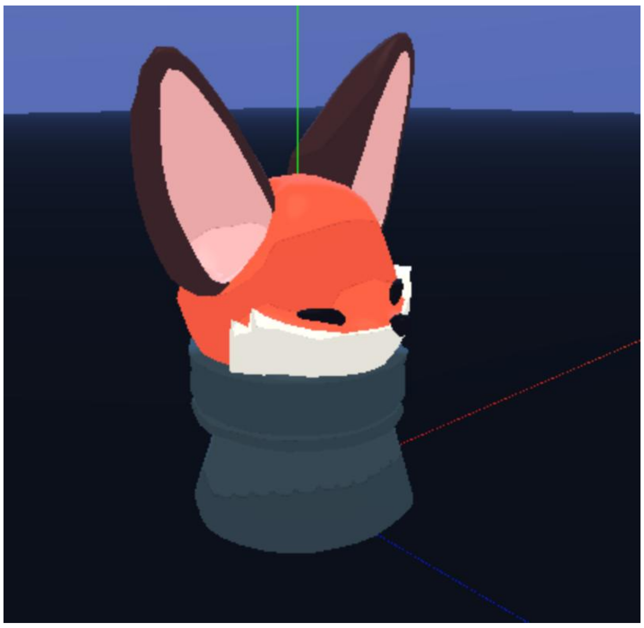


4.2 Shader

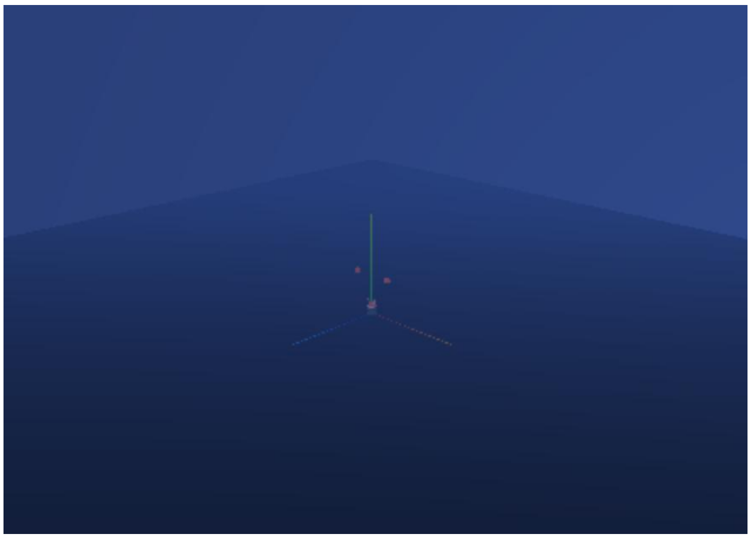
When rendering the fox, a cartoon-style shading is used. The following effects are implemented:

- Gradated light intensity

- Specular highlights
- Rim lighting
- Ambient light
- Shadows

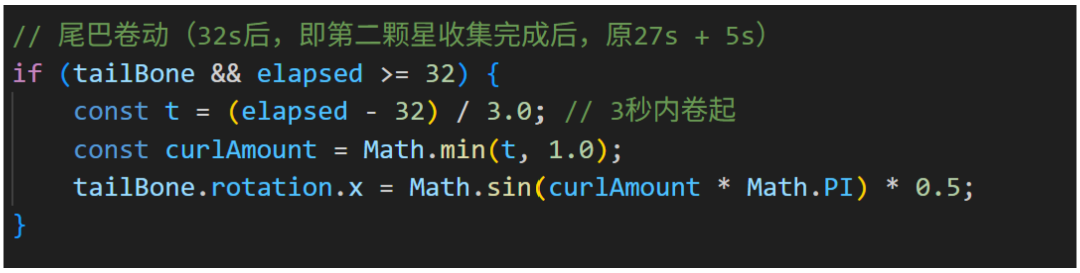


The project also implement Atmospheric scattering and fog effects through shaders.

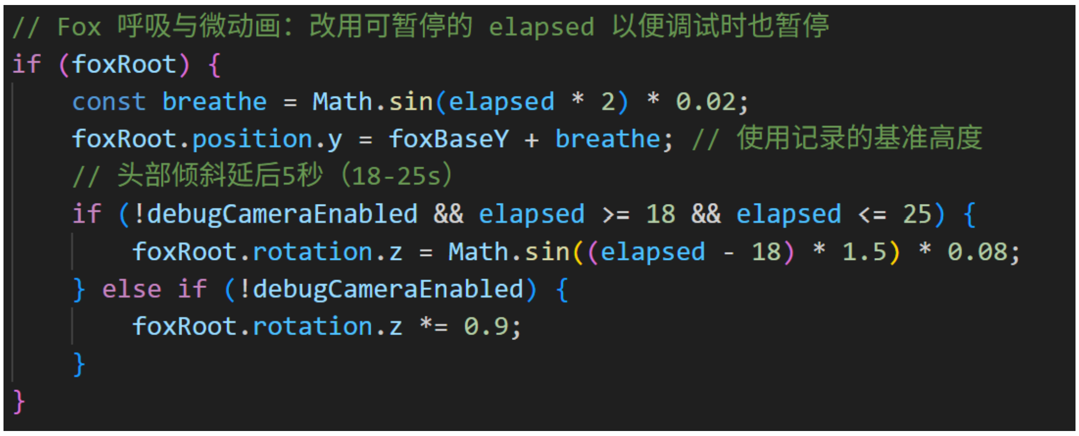


5. Animation

- **Fox Animation**: The tail curling is achieved through skeletal animation. The breathing rise and fall is simulated via position animation. The head tilting is realized using rotation animation.

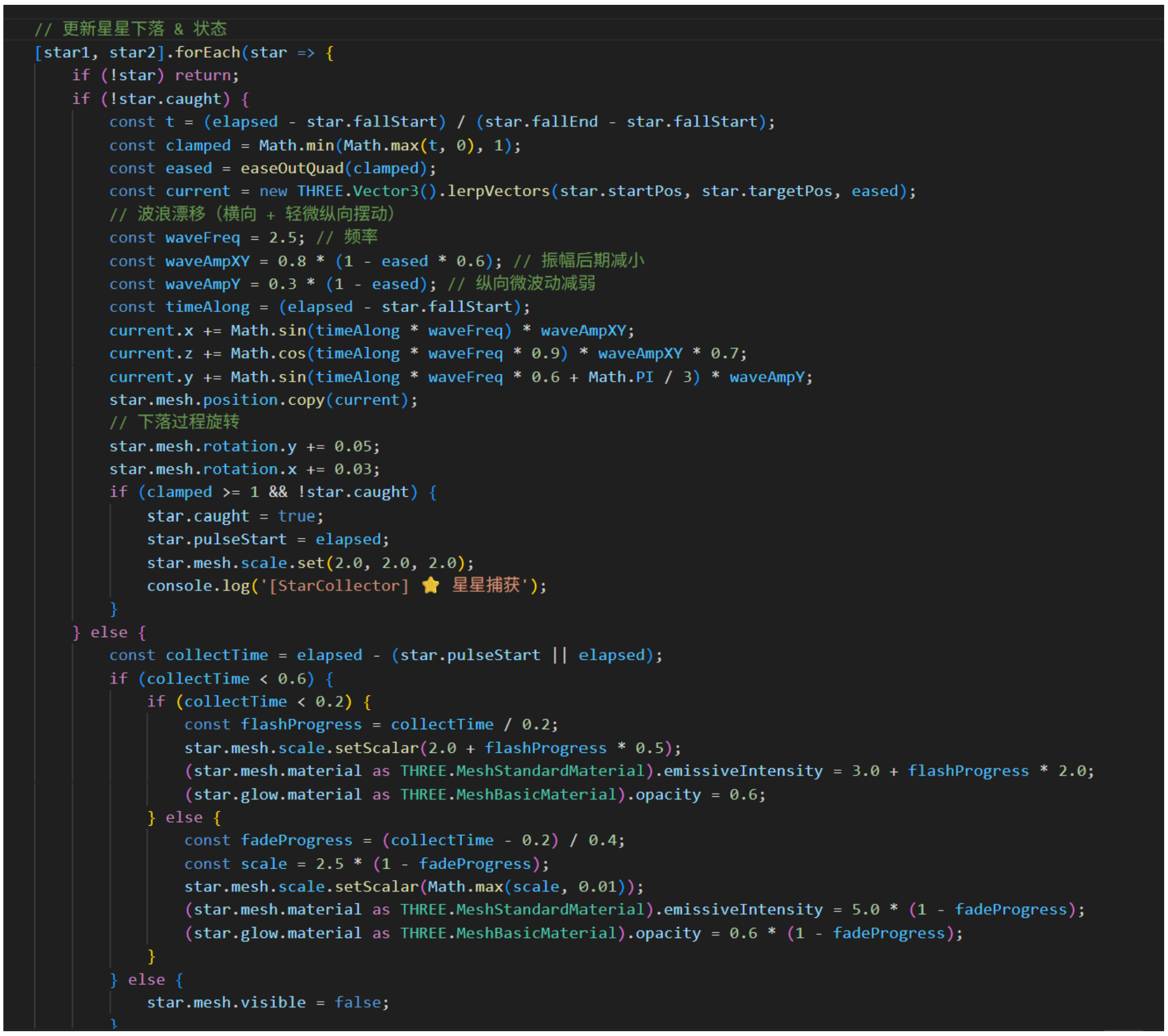


```
// 尾巴卷动（32s后，即第二颗星收集完成后，原27s + 5s）
if (tailBone && elapsed >= 32) {
    const t = (elapsed - 32) / 3.0; // 3秒内卷起
    const curlAmount = Math.min(t, 1.0);
    tailBone.rotation.x = Math.sin(curlAmount * Math.PI) * 0.5;
}
```

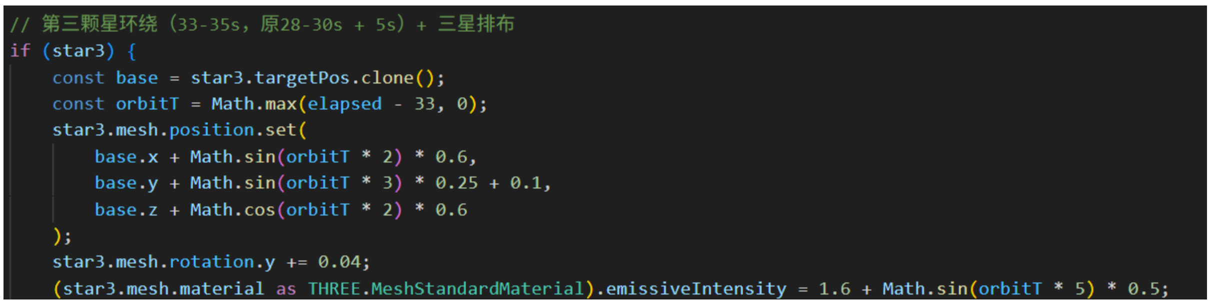


```
// Fox 呼吸与微动画：改用可暂停的 elapsed 以便调试时也暂停
if (foxRoot) {
    const breathe = Math.sin(elapsed * 2) * 0.02;
    foxRoot.position.y = foxBaseY + breathe; // 使用记录的基准高度
    // 头部倾斜延后5秒（18-25s）
    if (!debugCameraEnabled && elapsed >= 18 && elapsed <= 25) {
        foxRoot.rotation.z = Math.sin((elapsed - 18) * 1.5) * 0.08;
    } else if (!debugCameraEnabled) {
        foxRoot.rotation.z *= 0.9;
    }
}
```

- **Star Animation**:
    - **Falling Phase**: The slowing down of the falling speed is implemented with an easing function. The star is wavy drifting (horizontal + slight vertical swaying), and rotating during the falling process.
    - **Collection and Disappearance Phase**: In the collection phase, the light-emitting intensity fades rapidly; in the disappearance phase, the star shrinks quickly.
    - **Floating Phase**: The star moves and rotates around its spawn position.



```
// 更新星星下落 & 状态
[star1, star2].forEach(star => {
    if (!star) return;
    if (!star.caught) {
        const t = (elapsed - star.fallStart) / (star.fallEnd - star.fallStart);
        const clamped = Math.min(Math.max(t, 0), 1);
        const eased = easeOutQuad(clamped);
        const current = new THREE.Vector3().lerpVectors(star.startPos, star.targetPos, eased);
        // 波浪漂移 (横向 + 轻微纵向摆动)
        const waveFreq = 2.5; // 频率
        const waveAmpXY = 0.8 * (1 - eased * 0.6); // 振幅后期减小
        const waveAmpY = 0.3 * (1 - eased); // 纵向微波动减弱
        const timeAlong = (elapsed - star.fallStart);
        current.x += Math.sin(timeAlong * waveFreq) * waveAmpXY;
        current.z += Math.cos(timeAlong * waveFreq * 0.9) * waveAmpXY * 0.7;
        current.y += Math.sin(timeAlong * waveFreq * 0.6 + Math.PI / 3) * waveAmpY;
        star.mesh.position.copy(current);
        // 下落过程旋转
        star.mesh.rotation.y += 0.05;
        star.mesh.rotation.x += 0.03;
        if (clamped >= 1 && !star.caught) {
            star.caught = true;
            star.pulseStart = elapsed;
            star.mesh.scale.set(2.0, 2.0, 2.0);
            console.log('[StarCollector] ⭐ 星星捕获');
        }
    } else {
        const collectTime = elapsed - (star.pulseStart || elapsed);
        if (collectTime < 0.6) {
            if (collectTime < 0.2) {
                const flashProgress = collectTime / 0.2;
                star.mesh.scale.setScalar(2.0 + flashProgress * 0.5);
                (star.mesh.material as THREE.MeshStandardMaterial).emissiveIntensity = 3.0 + flashProgress * 2.0;
                (star.glow.material as THREE.MeshBasicMaterial).opacity = 0.6;
            } else {
                const fadeProgress = (collectTime - 0.2) / 0.4;
                const scale = 2.5 * (1 - fadeProgress);
                star.mesh.scale.setScalar(Math.max(scale, 0.01));
                (star.mesh.material as THREE.MeshStandardMaterial).emissiveIntensity = 5.0 * (1 - fadeProgress);
                (star.glow.material as THREE.MeshBasicMaterial).opacity = 0.6 * (1 - fadeProgress);
            }
        } else {
            star.mesh.visible = false;
        }
```

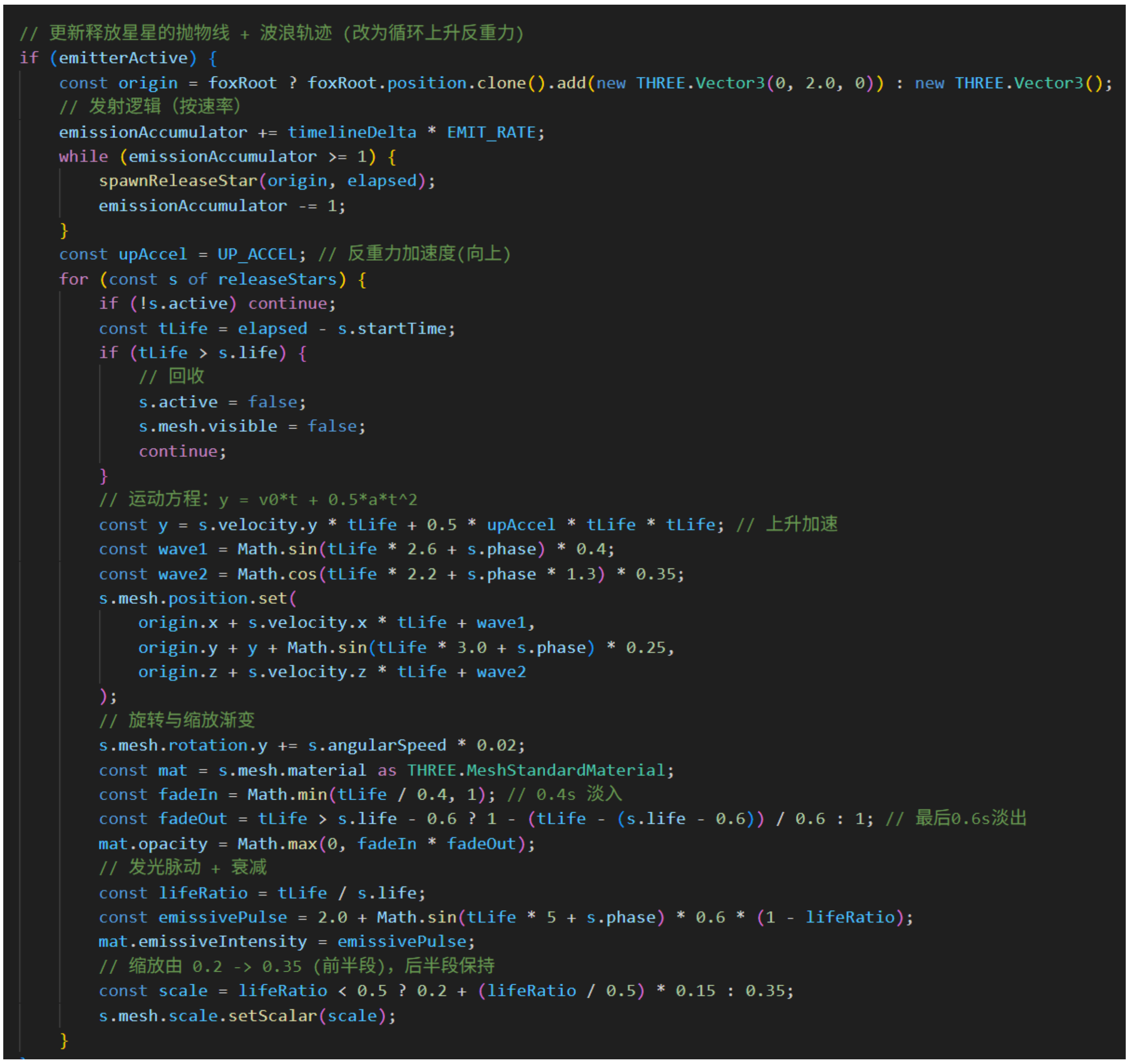


```
// 第三颗星环绕 (33-35s, 原28-30s + 5s) + 三星排布
if (star3) {
    const base = star3.targetPos.clone();
    const orbitT = Math.max(elapsed - 33, 0);
    star3.mesh.position.set(
        base.x + Math.sin(orbitT * 2) * 0.6,
        base.y + Math.sin(orbitT * 3) * 0.25 + 0.1,
        base.z + Math.cos(orbitT * 2) * 0.6
    );
    star3.mesh.rotation.y += 0.04;
    (star3.mesh.material as THREE.MeshStandardMaterial).emissiveIntensity = 1.6 + Math.sin(orbitT * 5) * 0.5;
```

- Star Particle Animation

  Star particles are generated at a specified emission rate. They accelerate upward from the starting point, with a wavy drift (horizontal + slight vertical oscillation) superimposed during the movement.

Within their lifetime, the particles first fade in and then fade out, while rotating at a constant speed. Their luminous intensity changes in pulses and gradually diminishes. The size of the particles increases in the first half of their lifetime and remains unchanged afterward.



```javascript
// 更新释放星星的抛物线 + 波浪轨迹 (改为循环上升反重力)
if (emitterActive) {
    const origin = foxRoot ? foxRoot.position.clone().add(new THREE.Vector3(0, 2.0, 0)) : new THREE.Vector3();
    // 发射逻辑 (按速率)
    emissionAccumulator += timelineDelta * EMIT_RATE;
    while (emissionAccumulator >= 1) {
        spawnReleaseStar(origin, elapsed);
        emissionAccumulator -= 1;
    }
    const upAccel = UP_ACCEL; // 反重力加速度(向上)
    for (const s of releaseStars) {
        if (!s.active) continue;
        const tLife = elapsed - s.startTime;
        if (tLife > s.life) {
            // 回收
            s.active = false;
            s.mesh.visible = false;
            continue;
        }
        // 运动方程: y = v0*t + 0.5*a*t^2
        const y = s.velocity.y * tLife + 0.5 * upAccel * tLife * tLife; // 上升加速
        const wave1 = Math.sin(tLife * 2.6 + s.phase) * 0.4;
        const wave2 = Math.cos(tLife * 2.2 + s.phase * 1.3) * 0.35;
        s.mesh.position.set(
            origin.x + s.velocity.x * tLife + wave1,
            origin.y + y + Math.sin(tLife * 3.0 + s.phase) * 0.25,
            origin.z + s.velocity.z * tLife + wave2
        );
        // 旋转与缩放渐变
        s.mesh.rotation.y += s.angularSpeed * 0.02;
        const mat = s.mesh.material as THREE.MeshStandardMaterial;
        const fadeIn = Math.min(tLife / 0.4, 1); // 0.4s 淡入
        const fadeOut = tLife > s.life - 0.6 ? 1 - (tLife - (s.life - 0.6)) / 0.6 : 1; // 最后0.6s淡出
        mat.opacity = Math.max(0, fadeIn * fadeOut);
        // 发光脉动 + 衰减
        const lifeRatio = tLife / s.life;
        const emissivePulse = 2.0 + Math.sin(tLife * 5 + s.phase) * 0.6 * (1 - lifeRatio);
        mat.emissiveIntensity = emissivePulse;
        // 缩放由 0.2 -> 0.35 (前半段), 后半段保持
        const scale = lifeRatio < 0.5 ? 0.2 + (lifeRatio / 0.5) * 0.15 : 0.35;
        s.mesh.scale.setScalar(scale);
    }
```

- **Animation Synchronization:** All animations are uniformly driven by a timer, and each animation is triggered according to time nodes.

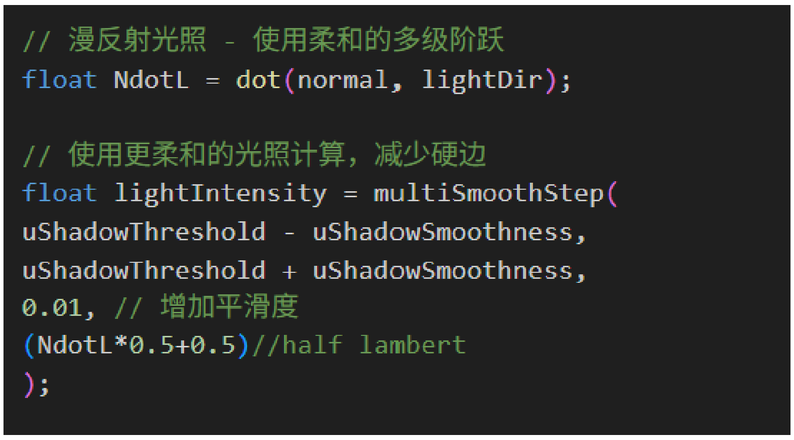
6. Discussion

6.1 Concept and Implementation of Toon Shader

To achieve the anime-style shading of the fox, we used the half-Lambert lighting model in the fragment shader and applied gradating to the light intensity.

- **6.1.1 Half-Lambert Lighting Model**

  The Lambert lighting model calculates the diffuse reflection intensity by computing the dot product (N•L) of the surface normal and the light source direction. When the dot product is less than or equal to 0, completely dark areas appear, which does not conform to the anime style we want. The Half-Lambert lighting model calculates the diffuse intensity using the formula



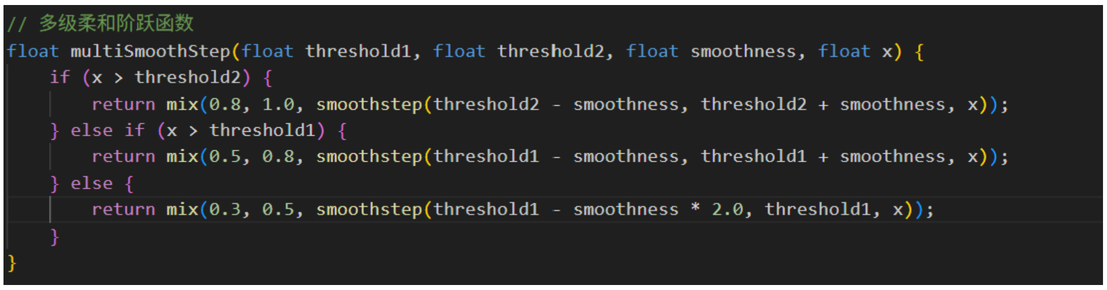
```
// 漫反射光照 - 使用柔和的多级阶跃
float NdotL = dot(normal, lightDir);

// 使用更柔和的光照计算，减少硬边
float lightIntensity = multiSmoothStep(
uShadowThreshold - uShadowSmoothness,
uShadowThreshold + uShadowSmoothness,
0.01, // 增加平滑度
(NdotL*0.5+0.5)//half lambert
);
```

  {0.5 (N•L) + 0.5}, avoiding the situation where the intensity is 0.

- **6.1.2 Gradated Light Intensity**

  Gradated Light Intensity effect refers to discretize the light intensity to produce color blocks. In this project, the multiSmoothStep() function is used to achieve tonal grading. It discretizes the light intensity into three values [0.3, 0.8, 1.0] based on three intervals: [0, threshold1], [threshold1, threshold2], and [threshold2, inf). Meanwhile, GLSL's smoothstep() func tion is used to achieve smooth transitions at the interval junctions, preventing aliasing.



```
// 多级柔和阶跃函数
float multiSmoothStep(float threshold1, float threshold2, float smoothness, float x) {
    if (x > threshold2) {
        return mix(0.8, 1.0, smoothstep(threshold2 - smoothness, threshold2 + smoothness, x));
    } else if (x > threshold1) {
        return mix(0.5, 0.8, smoothstep(threshold1 - smoothness, threshold1 + smoothness, x));
    } else {
        return mix(0.3, 0.5, smoothstep(threshold1 - smoothness * 2.0, threshold1, x));
    }
}
```

6.2 大气散射的概念和实现

Core Physical Concepts

1. Rayleigh Scattering

Scattering of light by atmospheric molecules (oxygen, nitrogen)

Strong wavelength dependence (blue light scatters more)

Responsible for blue daytime sky and red sunsets

Implemented using altitude-based coefficient (kr) and wavelength-dependent color

2. Mie Scattering

Scattering by larger atmospheric particles (aerosols, dust, water droplets)

Less wavelength-dependent (white/gray appearance)

Creates atmospheric haze, sun/moon halos, and daytime glow

Controlled by Henyey-Greenstein phase function

3. Optical Depth

Measures light attenuation through atmosphere

Depends on path length through atmosphere and scattering coefficients

Calculated using Chapman function approximation for spherical atmosphere

Implementation Components

1. Celestial Body Positioning

Sun Position: Direct light source with realistic rise/set calculation

Moon Position: Opposite to sun (simplified model) with own scattering

Starfield: Texture mapped to celestial sphere with intensity modulation

2. Atmospheric Calculations

Extinction: Exponential attenuation of light through atmosphere

In-Scattering: Light scattered toward viewer

Rayleigh phase function (symmetrical)

Mie phase function (forward-scattering using Henyey-Greenstein)

Zenith Angle Optimization: Chapman function approximation for computational efficiency

3. Phase Functions

Rayleigh Phase: 2.0 + 0.5 * pow(cosθ, 2.0)

Mie Phase (Henyey-Greenstein):

mieG.x / pow(mieG.y - mieG.z * cosθ, 1.5)

where mieG = vec3(1-g^2, 1+g^2, 2g) with g = asymmetry parameter

4. Day/Night Cycle System

Sunrise/Sunset: Smooth transition using horizon masking

Celestial Transition: Sun fades out as moon fades in

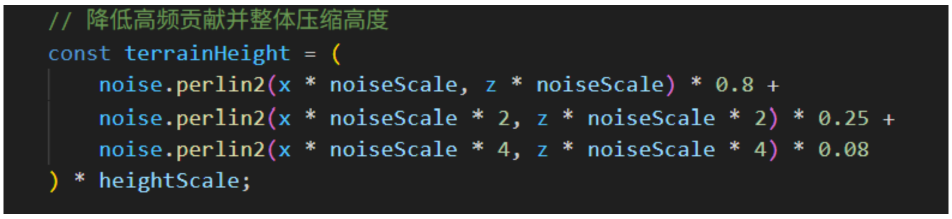
Horizon Effect: Enhanced scattering near horizon (horizonMask)

6.3 Concept and Implementation of Terrain Generation Using Perlin Noise

Perlin Noise is an algorithm for generating continuous pseudorandom signals. In this project, we use Perlin Noise to generate the height of each vertex on the terrain, thereby creating a natural hill terrain.
In practice, we used the noise.perlin2() function from the noisejs library and addressed the following details during use.

- **6.3.1 Achieving More Natural Terrain by Superimposing Noises with Multiple Frequencies and Amplitudes**

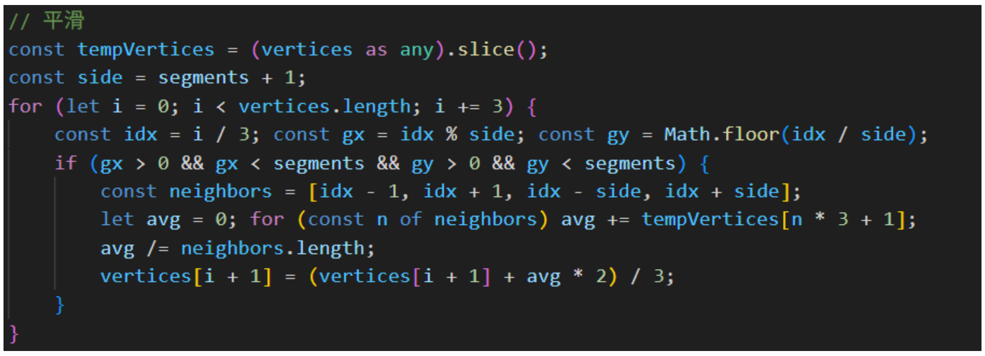
  The frequency of noise can be controlled by multiplying the input values of the noise function by different coefficients; a larger coefficient leads to faster noise variation.
  The amplitude of noise can be controlled by multiplying the output values of the noise function by different coefficients; a larger coefficient results in more intense noise variation.
  By combining noise values with low frequency and high amplitude, and values with high frequency and low amplitude, a more natural mountain model with different levels of relief can be generated.



```
// 降低高频贡献并整体压缩高度
const terrainHeight = (
    noise.perlin2(x * noiseScale, z * noiseScale) * 0.8 +
    noise.perlin2(x * noiseScale * 2, z * noiseScale * 2) * 0.25 +
    noise.perlin2(x * noiseScale * 4, z * noiseScale * 4) * 0.08
) * heightScale;
```

- 6.3.2 Generating Gentle Hills Through Smoothing

  Direct use of the noise function results in relatively steep terrain. To generate hilly terrain, the height of each vertex is weighted and averaged with the heights of its four surrounding vertices, making the terrain gentler.



```
// 平滑
const tempVertices = (vertices as any).slice();
const side = segments + 1;
for (let i = 0; i < vertices.length; i += 3) {
    const idx = i / 3; const gx = idx % side; const gy = Math.floor(idx / side);
    if (gx > 0 && gx < segments && gy > 0 && gy < segments) {
        const neighbors = [idx - 1, idx + 1, idx - side, idx + side];
        let avg = 0; for (const n of neighbors) avg += tempVertices[n * 3 + 1];
        avg /= neighbors.length;
        vertices[i + 1] = (vertices[i + 1] + avg * 2) / 3;
    }
}
```

7. Conclusion

- Achievements:
  - Complete 30s story-driven WebGL animation
  - Custom toon + atmospheric shaders
  - Sophisticated particle & animation system
  - Precise timeline synchronization

- Live Demo:

  morphex-mo.github.io/COMP4422-Computer-Graphics-Proj/?scene=starCollectorScene

- Source Code:

  github.com/Morphex-Mo/COMP4422-Computer-Graphics-Proj

Reference

Fox model:
https://sketchfab.com/3d-models/fox-12bf801481ee46d0bfab65a4f661e25b

Hill-tree model:
https://sketchfab.com/3d-models/hill-tree-41c4f1b3979940dab34d08b1bce197f9

Noisejs library:
https://www.npmjs.com/package/noisejs