Лабораторна робота №10: Фіналізація проєкту, оптимізація та реліз WebXR-застосунку

1. Мета роботи

- Навчитися проводити комплексний аудит продуктивності 3D-застосунків у браузері.
- Опанувати методи оптимізації графічних ресурсів (геометрії, текстур) та логіки коду.
- Підготувати проєкт до розгортання на хостинг-платформі з обов'язковою підтримкою протоколу HTTPS.

2. Теоретична частина (Build & Optimize)

2.1. Графічний конвеєр та Draw Calls

Draw Call — це команда, яку CPU надсилає GPU для малювання конкретного об'єкта.

- **Вузьке місце:** Велика кількість Draw Calls перевантажує CPU, що призводить до падіння FPS, навіть якщо GPU потужний.
- **Рішення:** Об’єднання об'єктів (Batching), використання атласів текстур та інстансингу.

2.2. Оптимізація ресурсів

1. **Draco Compression:** Бібліотека для стиснення геометрії (vertices, indices). Дозволяє зменшити розмір `.gltf`/`.glb` файлів у 5-10 разів без помітної втрати якості. Для обробки використовуйте gltf-pipeline.
2. **Texture Compression (KTX2 / Basis Universal):** На відміну від JPG/PNG, які розпаковуються в пам'яті GPU у формат RGBA8 (займаючи багато VRAM), формати KTX2 залишаються стиснутими навіть у відеопам'яті, що критично для мобільних AR-пристроїв.
3. **InstancedMesh:** Спеціальний клас у Three.js, який дозволяє рендерити тисячі однакових мешів за один Draw Call. Ідеально для лісів, натовпів або часток.

2.3. Особливості релізу WebXR

Браузерні API для AR та VR (`navigator.xr`) працюють **виключно в захищеному контексті (HTTPS)**. Винятком є лише `localhost`. Для тестування на реальних смартфонах зовні локальної мережі SSL-сертифікат є обов'язковим.

3. Практичне завдання (Фіналізація)

Крок 1: Профілювання

Встановіть та підключіть бібліотеку `stats.js` для моніторингу продуктивності.

```
import Stats from 'stats.js';


// Ініціалізація панелі статистики
const stats = new Stats();
stats.showPanel(0); // 0: fps, 1: ms, 2: mb
document.body.appendChild(stats.dom);


function animate() {
    stats.begin();
    // Логіка оновлення сцени
    renderer.render(scene, camera);
    stats.end();
    requestAnimationFrame(animate);
}
```

Крок 2: Оптимізація сцени за допомогою InstancedMesh

Якщо у вашому проєкті є багато однакових об'єктів (наприклад, ліхтарі або каміння), замініть їх на `InstancedMesh`.

```
// Приклад створення 1000 кубів одним викликом малювання
const geometry = new THREE.BoxGeometry(0.1, 0.1, 0.1);
const material = new THREE.MeshStandardMaterial({ color: 0x00ff00 });
const count = 1000;
const instancedMesh = new THREE.InstancedMesh(geometry, material, count);


const dummy = new THREE.Object3D();
```

```
for (let i = 0; i < count; i++) {
    dummy.position.set(Math.random() * 10, Math.random() * 10, Math.random() * 10);
    dummy.updateMatrix();
    instancedMesh.setMatrixAt(i, dummy.matrix);
}
scene.add(instancedMesh);

// Не забувайте очищувати пам'ять при видаленні об'єктів
function cleanup() {
    geometry.dispose();
    material.dispose();
    scene.remove(instancedMesh);
}
```

Крок 3: Build-процес

Використовуйте Vite для створення фінальної збірки:

1. **Команда:** `npm run build`.
2. **Tree-shaking:** Vite автоматично видалить частини бібліотеки Three.js, які ви не використовуєте.
3. **Minification:** Код буде стиснуто, змінні перейменовано на коротші для зменшення ваги JS-файлу.

Крок 4: Деплой

Розгорніть проєкт на одну з платформ:

- **Vercel / Netlify:** Автоматичний HTTPS, просте підключення GitHub-репозиторію.
- **GitHub Pages:** Безкоштовно, потребує налаштування `base` шляху у `vite.config.js`.

4. Завдання для самостійного виконання

1. **Динамічна якість:** Реалізуйте скрипт, який кожні 5 секунд перевіряє середній FPS. Якщо він менше 30 — вимкніть `renderer.shadowMap.enabled` або зменште `pixelRatio`.
2. **PWA (Progressive Web App):** Створіть файл `manifest.json` та зареєструйте Service Worker, щоб застосунок можна було "встановити" на головний екран смартфона.
3. **Loading Screen:** Використовуйте `THREE.LoadingManager`.
    - Показуйте `div` з індикатором завантаження.
    - Приховуйте його у функціях `onLoad` або `onProgress`.

```
const manager = new THREE.LoadingManager();
const loadingOverlay = document.getElementById('loading-screen');

manager.onLoad = function () {
    console.log('Усі ресурси завантажено!');
    loadingOverlay.style.display = 'none'; // Приховати екран завантаження
};

const loader = new GLTFLoader(manager);
```

5. Контрольні запитання

1. Яка максимальна кількість полігонів рекомендується для стабільної роботи мобільного AR-застосунку на середньому пристрої?
2. Чим відрізняється режим `npm run dev` (HMR, source maps) від `npm run build` у контексті швидкодії рендерингу?
3. Навіщо викликати метод `.dispose()` для геометрії та матеріалів, якщо в JavaScript є Garbage Collector?
4. Як протестувати AR-функціонал на смартфоні, використовуючи локальний сервер розробки (наприклад, через `ngrok` або локальну IP-адресу з самопідписаним сертифікатом)?