Документација за WebGL проект за симулација на парче ткаенина

Факултет за Информатички Науки и Компјутерско Инженерство, Универзитет Св. Кирил и Методиј Скопје, Р.С. Македонија

Дарко Пројковски
email: Darko.Projkovski@students.finki.ukim.mk

Апстракт *- Цел на овој проект е да се покажат можностите кои ни ги нуди WebGL со помош на неговите библиотеките. Градење на мала симулација на парче ткаенина кој ни ги демонстрира физичките својства кои се можни користејќи ги WebGL библиотеките како THREE.js и CANNON.js*

1 Вовед

Во 2023 повеќе од 5 милијарда луѓе го користат интернетот, прелистувајќи по веб страни кои голем број од нив се основни html и css со по некоја слики. Зошто прелистувањето по интернет да не биде интересно полно со можности за интеракција на лиѓето што го поседуваат веб сајтот со самиот веб сајт и услугите кој тој ги нуди. Тука влага во игра WebGL еволуција на Canvas 3Д експеримент започнат од Владимир Вукичевич базирано на OpenGL нуди API за 3Д графики. WebGL ни ја отвара вратата кон можностите да градиме 3Д свет на интернет.

2 Користени технологии

Пред да го започнеме проектот технологиите кои ќе ги користиме за да го изработиме истиот се следните:

-Visual Studio Code како работна околина

-Parcel bundler за која не е потребна никаква конфигурација

-WebGL за рендерирање на проектот

-THREE.js која е JavaScript библиотека изградена врз WebGL и помага при креација на 3Д објекти

-CANNON.js која е JavaScript библиотека која помага да симулира реален физички свет

3 Имплементација на проектот

Го започнуваме проектот со повикување на интанца од THREE.WebGLRenderer класата која ги рендерира објектите во сцената потоа ја поставуваме големината и бојата на сцената и ја додаваме на телото на страната. Антиалиас го поставуваме да биде вистина за рабовите на објектите да бидат по остри.

```
const renderer = new THREE.WebGLRenderer({antialias: true});
renderer.setSize(window.innerWidth, window.innerHeight);
renderer.setClearColor(0xA3A3A3);
document.body.appendChild(renderer.domElement);
```

Потоа креираме инстанца од THREE.Scene која ќе ни биде сцената во која ќе додаваме објекти, инстанца од THREE.PerspectiveCamera односно камерата низ која ќе ја гледаме сцената и инстанца од OrbitControls кои ни овозможува да ја движиме камерата. Ја поставуваме позицијата на камерата во просторот и и викаме да гледа кон центарот на координатиот систем, па ја ажурираме инстанцата од OrbitControls.

```
const scene = new THREE.Scene();

const camera = new THREE.PerspectiveCamera(24, window.innerWidth/window.innerHeight, 1, 2000)

const orbit = new OrbitControls(camera, renderer.domElement);


camera.position.set(4, 1, 1);
camera.lookAt(0, 0, 0);

orbit.update();
```

Следниов дел од кодот е само додавање на извор на светлини на сцената. Креираме три вида на светлина: амбиентална светлина која свети насекаде подеднакво, точкаста светлина која испушта светлина насочена кон центарот на координатниот систем со радоус кој ние го предефинираме и директиона светлина која наликува на сонце која има извор и е исто така насочена кој центарот на координатниот систем.

```
const ambientLight = new THREE.AmbientLight(0xffffff, 0.1);
scene.add(ambientLight);

const spotLight = new THREE.SpotLight(0xffffff, 0.9, 0, Math.PI / 8, 1);
spotLight.position.set(-3, 3, 10);
spotLight.target.position.set(0, 0, 0);
scene.add(spotLight);

const directionalLight = new THREE.DirectionalLight(0xffffff, 0.5);
directionalLight.position.set(0, 0, -10);
directionalLight.target.position.set(0, 0, 0);
scene.add(directionalLight);
```

Следниот чекор е креирање на инстанца од

CANNON.World односно тука ги дефинираме физичките својства кои ќе делуваат врз нашите објекти. Во нашиот случај ја поставуваме само гравитацијата со вредност -9.81 односно ќе тера сите објекти да се придвижуваат 9.81 единици по y-оската и со минусот кажуваме да е негативно односно придвижување надолу.

```
const world = new CANNON.World({
    gravity: new CANNON.Vec3(0, -9.81, 0)
});
```

За да може физичките својства да делуваат на нашите објекти треба да креираме инстанца од CANNON.Body во која ќе ги зададеме потребните параметри. За да може тоа да го направиме за почеток креираме инстанце од CANNON.Particle кој всушност му кажува дека нашиот објект ќе ја има формата на честичка односно точка. Ќе креираме низа која во себе ќе содржи поднизи од инстанци од CANNON.Body објекти. Секоја инстанца од CANNON.Body прима неколку параметри како што е масата на телото кое секогаш ќе биде 1 односно ќе бидат повлечени надолу од гравитацијата освен за оние честички кои се наоѓаат на почетокот на секоја подниза која ќе имаат вредност 0 за да го имаме ефектот дека нашата ткаенина е закачена од горната страна, потоа обликот кој го задаваме дека е од претходно креираната инстанца на честичка, позицијата на честичката која е всушност три димензионален вектор на кој што ги пресметуваме x, y и z координатите така што секој body објект ќе биде на растојание од останатите за претходно дефинирана dist константа и velocity или брзината која честичките ќе се движат за да имитираат еден вид ветер.

```
for (let i = 0; i < Nx + 1; i++){
    particles.push([]);
    for(let j = 0; j < Ny + 1; j++){
        const particle = new CANNON.Body({
            mass: j === Ny ? 0: mass,
            shape,
            position: new CANNON.Vec3((i - Nx * 0.5) * dist, (j - Ny * 0.5) * dist, 0),
            velocity: new CANNON.Vec3(0, 0, -0.1 * (Ny - j))
        })
        particles[i].push(particle);
        world.addBody(particle);
    }
}
```

Бидејќи секоја честица која ја додаваме е сама за себе CANNON.Body објект мораме ние да креираме функција која ќе им дава ограничување на честичките да ги споиме во една целина. За да го постигнеме тоа ја креираме финкцијата connect како влезни параметри ги приме две соседни точки/-честички и им додава ограничување кое е инстанца од CANNON.DistanceConstraint класата која зема една честица како прв параметар, втора честичка како втор параметар и растојанието кое треба да го држат помеѓу нив како трет параметар. За да го направиме ова за секоја соседна честичка креираме два вгнездени циклуси.

```
function connect(i1, j1, i2, j2){
    world.addConstraint(new CANNON.DistanceConstraint(particles[i1][j1], particles[i2][j2], dist));
}

for (let i = 0; i < Nx + 1; i++){
    for (let j = 0; j < Ny + 1; j++){
        if(i < Nx)
            connect(i, j, i + 1, j);
        if(j < Ny)
            connect(i, j, i, j + 1);
    }
}
```

Ги креиравме честичките на кој ќе дејствуваат физичките својства, но тие не се видливи на сцената за да ги направиме видливи треба најпрво да креираме една рамнина со инстанцирање на THREE.PlaneGeometry со исто број темиња како и бројот на честички, потоа инстанца од THREE.MeshPhongMaterial на кој најпрво велиме дека сакаме материјалот да биде двостран а потоа мапираме текстура односно инстанца од THREE.TextureLoader која ја лоадира напочетокот импортираната слика. За да биде ова една целина со помош на THREE.Mesh кажуваме кое геометриско тело, кој материјал ќе му припаѓа и потоа го додаваме на сцената.

```
const clothGeometry = new THREE.PlaneGeometry(1, 1, Nx, Ny)

const clothMat = new THREE.MeshPhongMaterial({
    side: THREE.DoubleSide,
    map: new THREE.TextureLoader().load(nebula)
})

const clothMesh = new THREE.Mesh(clothGeometry, clothMat);
scene.add(clothMesh);
```

Моментално геометриската рамнина и честичките се две различни целини за да го добиеме целиот изглед на една ткаенина мора истовремено позициите на честичките и геометриската рамнина да се придвижуваат. За ова да го постигнеме креираме функција во која има два вгнездени циклуси кои ќе ги ажурираат позициите на двете целини. Внатре во циклусите ќе креираме променлива/низа во која ќе се наоѓаат координатите на секој од темињата на рамнината односно секој три членови на низата креираат една xyz-координата и уште една променлива која ќе ги содржи позициите/координатите на честичките. Од променливата која ги содржи координатите на темињата на рамнината ја повикуваме функцијата setXYZ кое ги ажурира позициите на xyz-координатите на темињата според позициите на секоја кореспондирана честичка. За да знаеме која честичка се софпаѓа со кое теме ја користиме променливата index која работи на принцип ако $i = 0$ и $j = 0$ така индексот ќе биде 0 и така координатите на првото теме од првата колона ќе се ажурираат односно првата тројка од броеви додека па ако $i = 0$ и $j = 1$ индексот е еднаков на 16 па второто теме од првата колона ќе се ажурира односно втората тројка од броеви. Причината поради која започнуваме од $N_y - j$ е бидејќи кога ја креиравме честичката го користевеме push методот кое работи на методот прв влага, а последен излага, па координатите од првата честичка се наоѓа на крајот од низата.

```
function updateParticules(){
    for (let i = 0; i < Nx + 1; i++){
        for (let j = 0; j < Ny + 1; j++){
            const index = j * (Nx + 1) + i

            const positionAttribute = clothGeometry.attributes.position;

            const position = particles[i][Ny - j].position;

            positionAttribute.setXYZ(index, position.x, position.y, position.z)

            positionAttribute.needsUpdate = true
        }
    }
}
```

Како помошен елемент на сцената ќе креираме една обична топка нешто слично како што ја креиравме рамнината погоре, но наместо THREE.PlaneGeometry ќе користиме инстанца од THREE.PlaneGeometry класата и за да топката може да стапи во контакт со рамнината треба CANNON.Sphere тело и на телото му ја даваме формата на sphereGeometry и го додаваме со сцената и во физичкиот свет.

```
const sphereSize = 0.1;
const movementRadius = 0.2;

const sphereGeometry = new THREE.SphereGeometry(sphereSize);
const sphereMat = new THREE.MeshPhongMaterial();

const sphereMesh = new THREE.Mesh(sphereGeometry, sphereMat);
scene.add(sphereMesh);

const sphereShape = new CANNON.Sphere(sphereSize * 1.5);
sphereBody = new CANNON.Body({
    shape: sphereShape
})
world.addBody(sphereBody);
```

За крај имаме две работи слушач на настан кој го ажурира прозорецот во зависност од големината на прозорецот на прелистувачот и главниот анимациски циклус повикан од рендерот. Кој анимациски циклус ја повикува функцијата animate со параметарот time кој автомацки е пратен и го содржи моменталното време. Внатре во функцијата се повикува функцијата updateParticules која цело време прави пресметки каде се наоѓаат честичките однисно ткаенината и потоа ја ажурира состојбата на физичкиот симулациски свет шеесет пати во секунда. Потоа ја поставуваме позицијата на физичкото телото на топката така да се придвижува по x и z оската додека следната линија код после тоа ја копира позицијата на физичкото тело, THREE.js објектот за да се изрендерира топкана на сцената со истата позиција. И последно се повикува render функцијата која се повикува секој фраме да ја зжурира сцената со два параметри сцената која треба да ја рендерира како прв и камерата од која ја гледаме сцената како втор.

```
const timeStep = 1/60;

function animate(time){
    updateParticules();
    world.step(timeStep);
    sphereBody.position.set(movementRadius * Math.sin(time / 1000), 0, movementRadius * Math.sin(time / 1000));
    sphereMesh.position.copy(sphereBody.position);

    renderer.render(scene, camera);
}

renderer.setAnimationLoop(animate);

window.addEventListener('resize', function(){
    camera.aspect = this.window.innerWidth / this.window.innerHeight;
    camera.updateProjectionMatrix();
    renderer.setSize(this.window.innerWidth, this.window.innerHeight);
});
```

4 GitHub Repository

https://github.com/DarkoProjkovski/NVD$_{P}roject.git$