Міністерство освіти і науки України
Запорізький національний університет

О. Г. Спиця, А. О. Лісняк, В. В. Мухін, О. М. Мильцев

ПРОГРАМУВАННЯ КОМП'ЮТЕРНОЇ ГРАФІКИ У ВЕБ

**Навчальний посібник
для здобувачів ступеня вищої освіти бакалавра
спеціальності «Інженерія програмного забезпечення»
освітньо-професійної програми «Програмна інженерія»**

Затверджено
вченою радою ЗНУ
Протокол № 11 від
30 квітня 2024

Запоріжжя
2024

УДК 004.946(075.8)
П784

Спиця О. Г., Лісняк А. О., Мухін В. В., Мильцев О. М. Програмування комп’ютерної графіки у Веб : навчальний посібник для здобувачів ступеня вищої освіти бакалавра спеціальності «Інженерія програмного забезпечення» освітньо-професійної програми «Програмна інженерія». Запоріжжя : Запорізький національний університет, 2024. 191 с.

Навчальний посібник містить теоретичні відомості з технологій побудови тривимірної графіки в браузері WebGl. Розглянуті засоби бібліотеки Three.js для створення тривимірних геометричних об’єктів, анімації та елементів інтерактивності.

У посібнику наведено приклади створення тривимірних геометричних об’єктів.

Для здобувачів ступеня вищої освіти бакалавра спеціальності «Інженерія програмного забезпечення» освітньої програми «Програмна інженерія».

Рецензент
С. М. Гребенюк, доктор технічних наук, професор, завідувач кафедри фундаментальної та прикладної математики

Відповідальний за випуск
А. О. Лісняк, кандидат фізико-математичних наук, доцент, завідувач кафедри програмної інженерії

ЗМІСТ

Вступ			5
1	Бібліотека Three.js		7
	1.1	Створення сцени	8
	1.2	Створення об’єктів на сцені	9
	1.3	Декартова система координат	10
	1.4	Анімація	12
	1.5	Керування оглядом сцени	12
2	Примітиви Three.js		15
	2.1	BoxGeometry	15
	2.2	CircleGeometry	16
	2.3	ConeGeometry	18
	2.4	CylinderGeometry	19
	2.5	Dodecahedron	20
	2.6	IcosahedronGeometry	21
	2.7	OctahedronGeometry	22
	2.8	PlaneGeometry	23
	2.9	SphereGeometry	24
	2.10	TetrahedronGeometry	26
	2.11	TorusGeometry	27
	2.12	Використання бібліотеки dat.GUI	28
3	Додавання текстури на об'єкт. Матеріали Three.js		31
	3.1	Клас для завантаження текстури TextureLoader	33
	3.2	Повторення, зміщення, обертання текстури	35
	3.3	MeshBasicMaterial	37
	3.4	MeshLambertMaterial	40
	3.5	MeshPhongMaterial	41
	3.6	MeshStandardMaterial	41
	3.7	alphaMap (карта прозорості)	41
4	Огляд помічників		43
	4.1	ArrowHelper	43
	4.2	AxisHelper	44
	4.3	BoxHelper	45
	4.4	CameraHelper	45
	4.5	DirectionalLightHelper	47
	4.6	GridHelper	48
	4.7	HemisphereLightHelper	49
	4.8	PointLightHelper	50
	4.9	SpotLightHelper	52
5	Тіні. Освітлення у three.js.		54
	5.1	Тіні	54
	5.2	AmbientLight	56
	5.3	PointLight	58
	5.4	SpotLight	61

	5.5	DirectionalLight	64
6	Використання відео		67
	6.1	Використання відео як текстури	67
	6.1	Додавання 3D тексту	69
	6.2	Таймер Clock	73
	6.3	Створення структурних об’єктів	75
7	Створення анімації за допомогою Tween.js		81
	7.1	Побудова плоского кола	81
	7.2	Створення анімації за допомогою Tween.js	82
	7.3	Зміна функції плавності	83
	7.4	Управління анімацією	85
	7.5	Функція yoyo	86
	7.6	Функція delay	86
	7.7	Функція onUpdate	86
	7.8	Функція onComplete	87
	7.9	Рисування ліній та площині	87
	7.10	Побудова параметричних кривих	90
8	Конструктивна твердотільна геометрія		94
	8.1	Логічні операції над моделлю за допомогою ThreeCSG	94
	8.2	Поверхні обертання	97
	8.3	Криволінійні циліндри з ExtrudeGeometry	98
Використана література			103
Додатки			104

ВСТУП

Курс «Програмування комп'ютерної графіки у Веб» належить до *вибіркових курсів* циклу професійної підготовки спеціальності.

Метою вивчення навчальної дисципліни «Програмування комп'ютерної графіки у Веб» є вивчення технології побудови тривимірної графіки в браузері WebGl. Розглянуті засоби бібліотеки Three.js для створення тривимірних геометричних об'єктів, анімації та елементів інтерактивності.

Основними *завданнями* вивчення дисципліни «Програмування комп'ютерної графіки у Веб» є:

- застосовувати засоби бібліотеки Three.js;
- створювати анімацію за допомогою JavaScript бібліотеки Tween.js;
- застосовувати конструктивну твердотільну геометрію ThreeCSG.js;
- створювати власні вебсторінки, комерційні реклами, сайти.

Міждисциплінарні зв'язки. Для успішного вивчення курсу «Програмування комп'ютерної графіки у Web» здобувачам освіти необхідні базові знання та уміння з дисциплін «Теорія алгоритмів та програмування» та «Лінійна алгебра та аналітична геометрія». Знання та уміння набуті під час вивчення дисципліни «Програмування комп'ютерної графіки у Web» будуть корисні при виконанні курсових робіт і кваліфікаційної роботи бакалавра та в подальшій професійній діяльності.

У результаті вивчення навчальної дисципліни студент повинен

знати:

- мову HTML(XHTML);
- таблиці стилів CSS;
- мову JavaScript;
- набір готових класів бібліотеки Three.js для створення і відображення інтерактивної комп'ютерної 3D графіки в WebGl;

вміти:

- застосовувати засоби бібліотеки Three.js для створення тривимірних геометричних об'єктів, анімації;
- створювати власні WEB-сторінки, комерційні реклами, сайти з використанням комп'ютерної графіки.

Змістове наповнення курсу, що викладається на лекційних і лаораторних заняттях та засвоюється студентом під час самостійної роботи, забезпечує набуття компетентностей:

- *Здатність до абстрактного мислення, аналізу та синтезу.*
- *Здатність до розуміння предметної області та професійної діяльності.*
- *Здатність вчитися і оволодівати сучасними знаннями.*
- *Здатність до пошуку, оброблення та узагальнення інформації з різних джерел.*

– *Здатність управляти та користуватися сучасними інформаційно-комунікаційними системами та технологіями (у тому числі такими, що базуються на використанні Інтернет)*

Програмні результати навчання. У разі успішного завершення курсу студент зможе:

– застосовувати засоби бібліотеки Three.js;
– створювати анімацію за допомогою JavaScript бібліотеки Tween.js;
– застосовувати конструктивну твердотільну геометрію ThreeCSG.js;
– створювати власні вебсторінки, комерційні реклами, сайти.

Згідно з робочою програмою дисципліни «Програмування комп'ютерної графіки у Веб» посібник охоплює теоретичний матеріал до наступних тем:

– бібліотека Three.js;
– примітиви Three.js. Використання бібліотеки dat.GUI;
– додавання текстури на об'єкт;
– матеріали Three.js;
– огляд помічників;
– тіні;
– освітлення у three.js;
– використання відео як текстури;
– додавання 3D тексту;
– таймер Clock;
– створення структурних об'єктів;
– побудова плоского кола;
– створення анімації за допомогою Tween.js;
– конструктивна твердотільна геометрія;
– рисування ліній та площині;
– побудова параметричних кривих;
– поверхні обертання;
– криволінійні циліндри з ExtrudeGeometry.

У кожній темі окрім теоретичного матеріалу представлена велика кількість прикладів (додатків) з детальними коментарями.

1 БІБЛІОТЕКА THREE.JS

Three.js -це кросбраузерна JavaScript бібліотека, яка використовується для створення та відображення анімованої тривимірної комп'ютерної графіки у веб-браузері. Three.js використовує WebGl.

WebGl– це програмна бібліотека для JavaScript, яка дозволяє створювати 3D графіку, яка функціонує у браузерах:

Three.js – JavaScript 3D Library

https://threejs.org (рисунок 1.1)

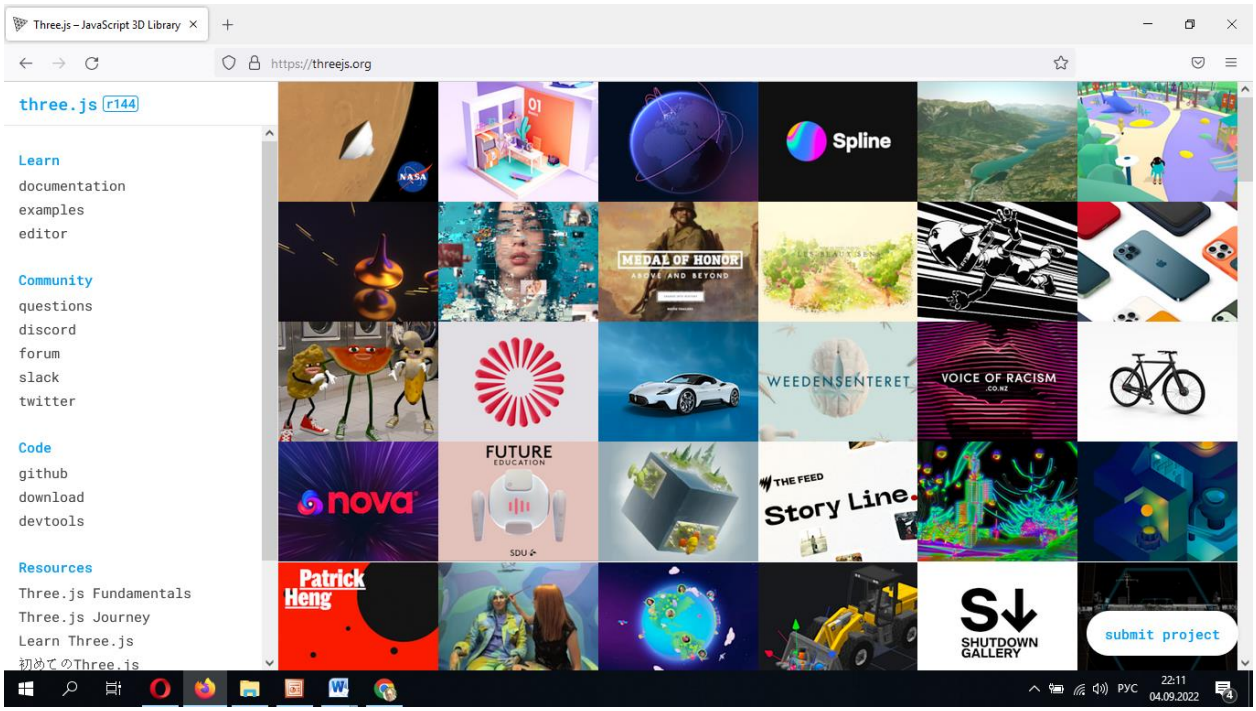


Рисунок 1.1 – Threejs.org

На сайті є документація, приклади та редактор.

Моделювання графіки з використанням Three.js дає можливість оперувати такими поняттями як сцена, світло, камера, об'єкти та їх матеріали.

У цьому розділі ми розглянемо основи використання Three.js, структуру необхідного HTML (додаток А), фундаментальні об'єкти бібліотеки та як створити просту сцену:

```
<!DOCTYPE html>
<html>
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
        <title>Приклад 01-01 – Порожня сторінка</title>
         <script src="./js/three.js"></script>
```

```
</head>
<body>
	<script>
		init();
		function init()
		{
		}
	</script>
</body>
</html>
```

1.1 Створення сцени

Для того, щоб була можливість щось відображати з використанням Three.js, потрібні три речі: сцена (***scene***), камера (***camera***) і візуалізатор (***renderer***) рендерер, щоб була можливість показувати сцену, зняту камерою.

- ***Scene*** своєрідна платформа, де розміщуються всі об’єкти, які ми створюємо;
- ***Camera*** -по суті – це "око", яке буде спрямовано на сцену. Камера знімає та відображає об’єкти, які розташовані на сцені;
- ***Renderer***-візуалізатор, який дозволяє відображати сцену, зняту камерою.

```
//Ініціалізації рендерера
var renderer = new THREE.WebGLRenderer();
renderer.setClearColor(new THREE.Color(0xEEEEEE));
// Встановлення розміру
renderer.setSize( window.innerWidth, window.innerHeight );
```

//Додавання елемента рендерера на сторінку. Робиться це так і однаково для всіх типів рендерерів:

```
document.body.appendChild( renderer.domElement );

var camera = new THREE.PerspectiveCamera(45,window.innerWidth / window.innerHeight, 0.1,1000);
var scene = new THREE.Scene();
```

У даному прикладі використовується перспективна камера.

Перспективна камера приймає 4 аргументи:

- ***Fov*** або Field Of View (поле/кут зору) – визначає кут, який можна бачити навколо центру камери;
- ***Aspect ratio*** – пропорція, або співвідношення ширини до висоти екрана. При великих значеннях поля зору видимий розмір об’єктів швидко зменшується на відстані. При невеликих значеннях, навпаки, видимий розмір об’єктів слабо залежить від відстані;

– ***Near & Far*** – мінімальна та максимальна відстань від камери, яка потрапляє у рендеринг. Так, дуже далекі точки не будуть рисуватись взагалі, як і точки, які знаходяться дуже близько.

Таким чином, ми створили сцену (scene), камеру (camera) і візуалізатор, але сцена в нас порожня.

1.2 Створення об’єктів на сцені

Об’єкт, який створюється на сцені, називається ***Mesh***.

Mesh – це клас, який представляє об’єкти на основі трикутної полігональної сітки.

Цей клас приймає 2 аргументи:

– ***Geometry*** – описує форму (положення вершин, грані, радіус тощо);
– ***Material*** – описує зовнішній вигляд об’єктів (колір, текстура, прозорість тощо).

Спробуємо створити найпростіші об’єкти – осі координат, площину та куб:

```
var axes = new THREE.AxisHelper( 20 );
scene.add(axes);

var planeGeometry = new THREE.PlaneGeometry(60,20,1,1);
var planeMaterial = new THREE.MeshBasicMaterial({color: 0xcccccc});
var plane = new THREE.Mesh(planeGeometry,planeMaterial);

plane.rotation.x=-0.5*Math.PI;
plane.position.x = 15;
plane.position.y = 0;
plane.position.z = 0;
scene.add(plane);

var cubeGeometry = new THREE.CubeGeometry(4,4,4);
var cubeMaterial = new THREE.MeshBasicMaterial({color: 0xff0000, wireframe: true});
var cube = new THREE.Mesh(cubeGeometry, cubeMaterial);

cube.position.x = -4;
cube.position.y = 3;
cube.position.z = 0;
scene.add(cube);
```

Three.js надає кілька типів матеріалів. Вони визначають, як об’єкти з’являтимуться на сцені. Які матеріали ви використовуєте залежить від того, чого ви намагаєтеся досягти.

В даному прикладі (додаток Б) використовується матеріал **MeshBasicMaterial**, який не залежить від світла.

Основні властивості якого:

color: колір матеріалу (за замовчуванням встановлений білий (0xffffff)).
wireframe (каркас): візуалізація геометрії як каркаса, тобто вершин та ребер. За промовчанням -false.

В наведеному прикладі ми зустріли не розглянуті функції position та rotation.

1.3 Декартова система координат

Для розташування фігур у просторі використовується декартова система координат (рисунок 1.2):

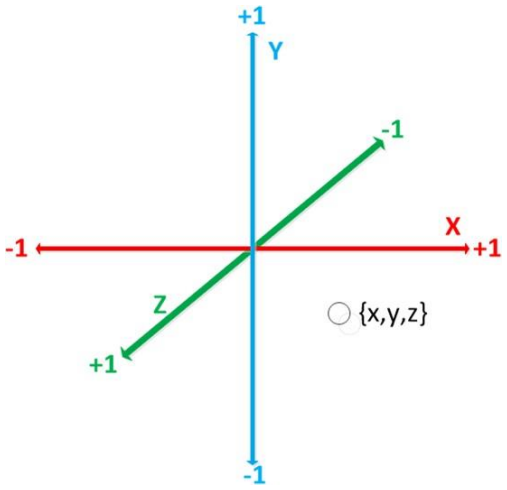


Рисунок 1.2 – Декартова система координат у WebGL

Як завжди, для програм, які працюють із ***3D***, вгору спрямована вісь ***Y***. На рисунку також вказані правила зміни знаків. Наприклад, при наближенні до нас значення координати ***Z*** збільшується.

При додаванні об’єкта на сцену ***Three.js*** можна вказати його координати:

```
object.position.x = -50; object.position.y = 20; object.position.z = 60;
```

або одним рядком:

```
object.position.set(-50, 20, 60);
```

або за допомогою класу тривимірних векторів ***Vector3***:

```
object.position = new THREE.Vector3( -50, 20, 60 );
```

При цьому у зазначеній точці зазвичай розташований геометричний

центр тіла. Якщо ж координати об’єкта не вказані, всі вони рівні нулю.

Можна встановити координати об’єкта, прирівнявши їх до координат іншого:

```
object2.position = object1.position;
```

Для завдання кутів повороту тіла використовують властивість rotation. Кути вказуються у радіанах. Наприклад, команда

```
object.rotation.y = Math.PI/2;
```

означає, що об’єкт обертається на 90 градусів проти годинникової стрілки, якщо дивитися "зверху" – з боку позитивного напрямку вісі ***OY***, при цьому віссю обертання служить вісь ординат ***OY***. Синтаксис rotation такий же, як і в position.

Останньою дією нам необхідно запустити рендеринг.

Далі йде візуалізатор (рендерер).

Тут можна пояснити, що таке рендеринг (rendering), рендер (render) і рендерер (renderer).

Рендеринг (візуалізація) це процес перерахунку тривимірної сцени, тобто всіх об’єктів, які знаходяться в полі видимості камери, двовимірне зображення. Ось це підсумкове зображення, яке відображається на екрані монітора, називається рендером. А програма, яка виконує рендеринг, відповідно називається рендерером або візуалізатором.

Крім створення екземпляра візуалізатора (рендерера), також потрібно встановити розміри, в яких буде відображатися наша програма. Хорошою ідеєю буде використання ширини та висоти області, яку нам хочеться зайняти нашим додатком – у даному випадку, це ширина та висота вікна браузера. При виконанні ресурсоємних програм також можна задати параметру setSize менші значення, window.innerWidth/2 і window.innerHeight/2, які зроблять візуалізацію програми у половинному розмірі.

Якщо потрібно зберегти розміри програми, але відобразити її з нижчою роздільною здатністю, можна зробити це, викликавши параметр setSize зі значенням false як параметр updateStyle.

Наприклад, код

```
setSize(window.innerWidth/2, window.innerHeight/2, false)
```

зробить візуалізацію програми у половинному розмірі, з урахуванням того, що наш <canvas> має 100%. -ву ширину і висоту від розмірів вікна браузера.

І нарешті, що не менш важливо, до HTML-документу додаємо елемент візуалізатора. Це елемент<canvas>, де рендерер і відображає сцену (додаток Б, рисунок 1.3).

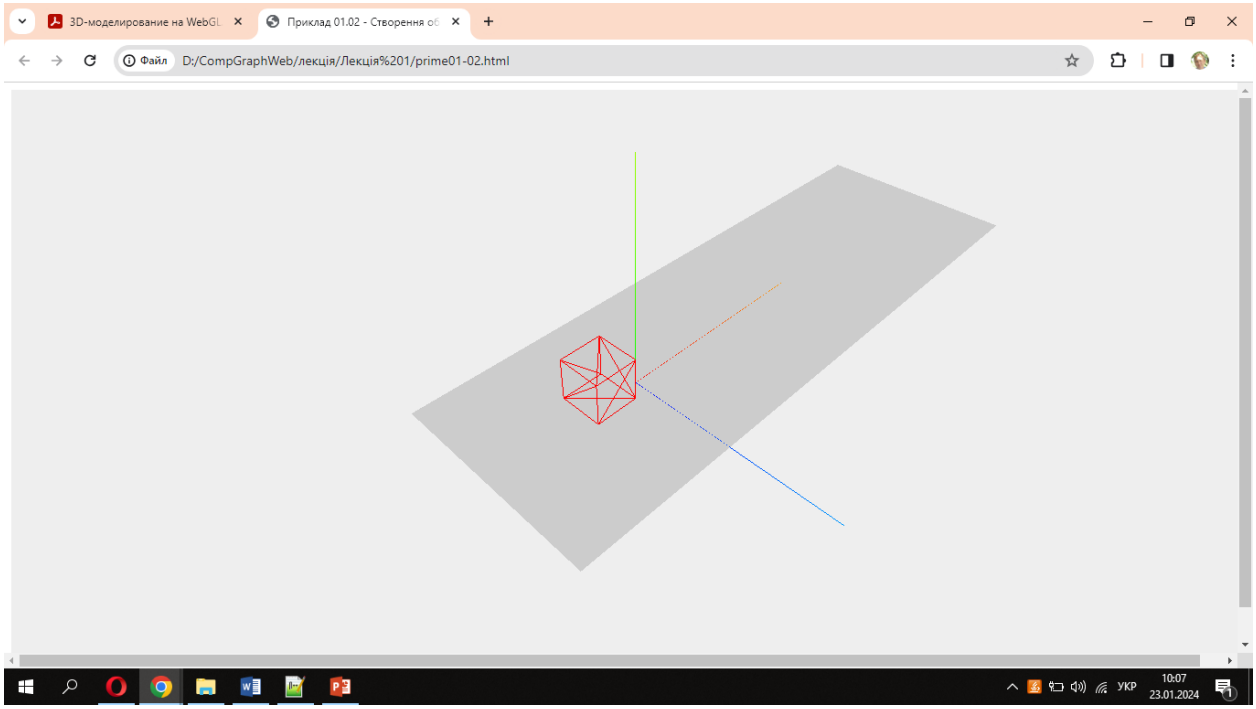


Рисунок 1.3 – Створення об’єктів на сцені

1.4 Анімація

Анімація – послідовний неодноразовий рендеринг сцени (для відображення динаміки). Таке послідовне відображення здійснюється функцією requestAnimationFrame. Приклад 01-03.html наведено в додатку В.

```
function renderScene()
{
  cube.rotation.x += 0.02;
  cube.rotation.y+=0.02;
  cube.rotation.z += 0.02;
  requestAnimationFrame(renderScene);
  renderer.render (scene, camera);
}
```

1.5 Керування оглядом сцени

З використанням перспективної камери важко досить повно оцінити тривимірність об’єктів на сцені. Хочеться мати можливість розглядати об’єкти з різних боків.

Для цього Three.js є можливість керування оглядом сцени за допомогою спеціальних «контролів», додавання яких дозволяє змінювати точку огляду камери за допомогою мишки, наближатися або віддалятися від сцени.

Для керування оглядом необхідно завантажити файл TrackballControls.js сайту http://Threejs.org у папку js і для його використання додати посилання (додаток Г):

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 01 – 04 Керування оглядом сцени </title>
<script src="./js/three.js"></script>
<script src="./js/TrackballControls.js"></script>
</head>
```

Тепер, після створення камери, можна додати контрол:

```
var camera = new THREE.PerspectiveCamera(45,window.innerWidth / window.innerHeight, 0.1,1000);
var scene = new THREE.Scene();
var controls = new THREE.TrackballControls( camera );
```

Перший параметр Camera.

Після створення контролу можна задати, наприклад, швидкість обертання камери під час руху миші:

```
controls.rotateSpeed=2;
```

чи змінюватиметься положення камери (наближатися або віддалятися від сцени) при крученні коліщатка миші (за замовчуванням false – змінюється), і швидкість такої зміни:

```
controls.noZoom=false;
controls.zoomSpeed=1.2;
```

Також можна вказати, чи камера після зупинки миші трохи рухатиметься за інерцією чи ні, за замовчуванням false (рухається):

```
controls.staticMoving=true;
```

Також є інші контроли. Наприклад, FirstPersonConrols дозволяє "бачити" сцену від першої особи, і пересуватися по ній за допомогою стрілок (або кнопок WASD) і миші.

Завдання

Для данного прикладу геометричної фігури (додатки Б, В, Г):

– зробити власну геометричну фігуру (куля, піраміда, ромб…). Управління оглядом сцени за допомогою TrackballControls;

– надіслати звіт і архів всіх файлів (включаючи папку з бібліотеками).

Рекомендована література

1. Dirksen J. Learn Three.js : Program 3D Animations and Visualizations for the Web with JavaScript and WebGl. Packt Publishing, Limited, 2023. 554 p.
2. Фреймворк three.js : офіційний сайт. URL : http://threejs.org (дата звернення : 18.01.2024).

Контрольні запитання

1. Що таке бібліотека three.js і для чого вона використовується?
2. Як встановити та підключити three.js до мого веб-проекту?
3. Як створити сцену в three.js?
4. Як додати об’єкти (наприклад, куби, сфери) на сцену в three.js?
5. Як керувати камерою в three.js?
6. Як керувати оглядом сцени?
7. Як встановити рух (анімацію) об’єктів в three.js?

2 ПРИМІТИВИ THREE.JS

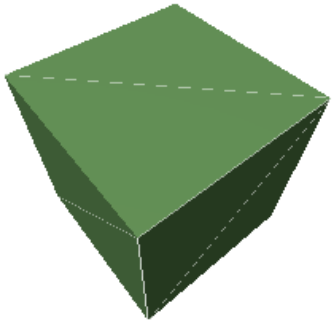
Three.js має велику кількість примітивів. Примітиви зазвичай є тривимірними фігурами, які генеруються під час виконання з безліччю параметрів.

Примітиви особливо часто використовуються для експериментів та початку роботи з 3D.

Багато наведених нижче примітивів мають значення за замовчуванням для деяких або всіх параметрів, тому ви можете використовувати більше або менше залежно від ваших потреб.

2.1 BoxGeometry

BoxGeometry – це клас геометрії для прямокутного кубоїда із заданою "шириною", "висотою" та "глибиною". При створенні кубоїд центрується на початку координат, причому кожен край паралельний до однієї з осей (рисунки 2.1, 2.2).



```
const width = 8;
const height = 8;
const depth = 8;
const geometry = new THREE.BoxGeometry( width, height, depth );
```

Рисунок 2.1 – BoxGeometry

Конструктор

BoxGeometry(width: Float, height: Float, depth: Float, widthSegments: Integer, heightSegments: Integer, depthSegments: Integer). Тут

width – ширина. Тобто довжина ребер, паралельних осі X. Необов’язковий параметр. За замовчуванням 1;

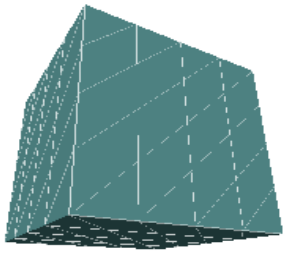
height – висота. Тобто довжина ребер, паралельних осі Y. Необов’язковий параметр. За замовчуванням 1;

depth – довжина. Тобто довжина ребер, паралельних осі Z. Необов'язковий параметр. За замовчуванням 1;

widthSegments – кількість сегментованих прямокутних граней по ширині сторін. Необов'язковий параметр. За замовчуванням 1;

heightSegments – кількість сегментованих прямокутних граней по висоті сторін. Необов'язковий параметр. За замовчуванням 1;

depthSegments – кількість сегментованих прямокутних граней по довжині сторін. Необов'язковий параметр. За замовчуванням 1.



```
const width = 8;
const height = 8;
const depth = 8;
const widthSegments = 4;
const heightSegments = 4;
const depthSegments = 4;
const geometry = new THREE.BoxGeometry(
    width, height, depth,
    widthSegments, heightSegments, depthSegments );
```

Рисунок 2.2 – BoxGeometry з використанням кількісті сегментованих прямокутних граней

2.2 CircleGeometry

CircleGeometry–це проста форма евклідової геометрії (плоске коло). Плоске коло складається з ряду трикутних сегментів, які орієнтовані навколо центральної точки та простягаються на заданий радіус. Коло будується проти годинникової стрілки від початкового кута та заданого центрального кута. Його також можна використовувати для створення правильних багатокутників, де кількість сегментів визначає кількість сторін (див. рис. 2.3 – 2.4).

Конструктор

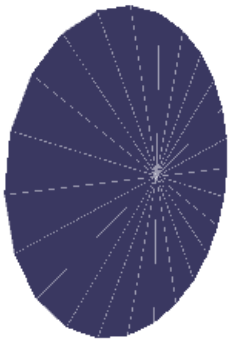
CircleGeometry(radius: Float, segments: Integer, thetaStart: Float, thetaLength: Float). Тут:

radius – радіус кола, за замовчуванням = 1;

segments – кількість сегментів (трикутників), мінімум = 3, за замовчуванням = 32;

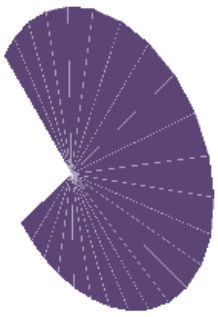
thetaStart – початковий кут для першого сегмента, за замовчуванням = 0 (положення на три години);

thetaLength – центральний кут, який часто називають тета, кругового сектора. За замовчуванням 2*Pi, що означає повне коло.



```
const radius = 7;
const segments = 24;
const geometry = new THREE.CircleGeometry( radius, segments );
```

Рисунок 2.3 – CircleGeometry (плоске коло)

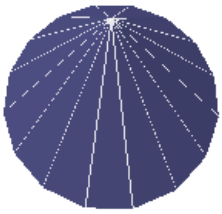


```
const radius = 7;
const segments = 24;
const thetaStart = Math.PI * 0.25;
const thetaLength = Math.PI * 1.5;
const geometry = new THREE.CircleGeometry(
    radius, segments, thetaStart, thetaLength );
```

Рисунок 2.4 – Плоске коло з використанням початкового та центрального кутів

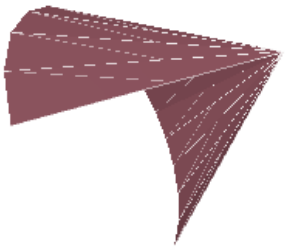
2.3 ConeGeometry

Клас для створення геометрії конуса (рисунки 2.5, 2.6).



```
const radius = 6;
const height = 8;
const radialSegments = 16;
const geometry = new THREE.ConeGeometry( radius, height, radialSegments );
```

Рисунок 2.5 – Конус



```
const radius = 6;
const height = 8;
const radialSegments = 16;
const heightSegments = 2;
const openEnded = true;
const thetaStart = Math.PI * 0.25;
const thetaLength = Math.PI * 1.5;
const geometry = new THREE.ConeGeometry(
    radius, height,
    radialSegments, heightSegments,
    openEnded,
    thetaStart, thetaLength );
```

Рисунок 2.6 – Конус (з додатковими параметрами)

Конструктор

ConeGeometry(radius: Float, height: Float, radialSegments: Integer, heightSegments: Integer, openEnded: Boolean, thetaStart: Float, thetaLength: Float). Тут:

radius – радіус основи конуса. За замовчуванням 1;

height – висота конуса. За замовчуванням 1;

radialSegments – кількість сегментованих граней по колу конуса. За замовчуванням 32;

heightSegments – кількість рядів граней по висоті конуса. За замовчуванням 1;

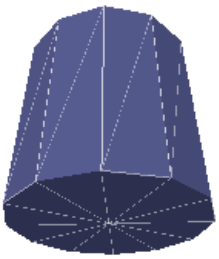
openEnded – логічне значення, яке вказує, чи є основа конуса відкритою чи закритою. За замовчуванням значення false, що означає обмеження;

thetaStart – початковий кут для першого сегмента, за замовчуванням = 0 (положення на три години);

thetaLength – центральний кут, який часто називають тета, кругового сектора. За замовчуванням 2*Pi, що створює повний конус.

2.4 CylinderGeometry

Клас для створення геометрії циліндра (рисунки 2.7 – 2.8).



```
const radiusTop = 4;
const radiusBottom = 4;
const height = 8;
const radialSegments = 12;
const geometry = new THREE.CylinderGeometry(
    radiusTop, radiusBottom, height, radialSegments );
```

Рисунок 2.7 – Циліндр

Конструктор

CylinderGeometry(radiusTop: Float, radiusBottom: Float, height: Float, radialSegments: Integer, heightSegments: Integer, openEnded: Boolean, thetaStart: Float, thetaLength: Float). Тут:

radiusTop – радіус циліндра зверху. За замовчуванням 1;

radiusBottom – радіус циліндра внизу. За замовчуванням 1;

height – висота циліндра. За замовчуванням 1;

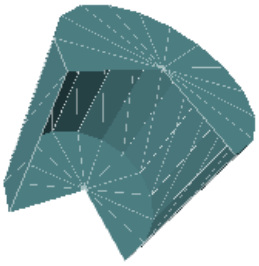
radialSegments – кількість сегментованих граней по колу циліндра. За замовчуванням 32;

heightSegments – кількість рядів граней по висоті циліндра. За замовчуванням 1;

openEnded – логічне значення, яке вказує, чи відкриті кінці циліндра чи закриті. За замовчуванням значення false, що означає обмеження;

thetaStart – початковий кут для першого сегмента, за замовчуванням = 0 (положення на три години);

thetaLength – центральний кут, який часто називають тета, кругового сектора. За замовчуванням 2*Pi, що означає повний циліндр.



```
const radiusTop = 4;
const radiusBottom = 4;
const height = 8;
const radialSegments = 12;
const heightSegments = 2;
const openEnded = false;
const thetaStart = Math.PI * 0.25;
const thetaLength = Math.PI * 1.5;
const geometry = new THREE.CylinderGeometry(
    radiusTop, radiusBottom, height,
    radialSegments, heightSegments,
    openEnded,
    thetaStart, thetaLength );
```

Рисунок 2.8 – Циліндр (з додатковими параметрами)

2.5 Dodecahedron

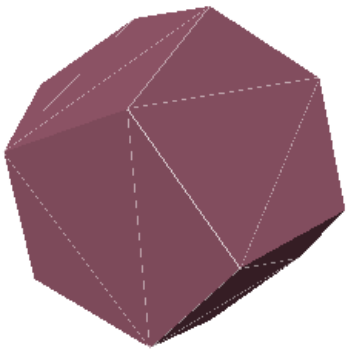
Клас для генерації геометрій додекаедра (рисунки 2.9 – 2.10).

Конструктор

DodecahedronGeometry(radius: Float, detail: Integer). Тут:

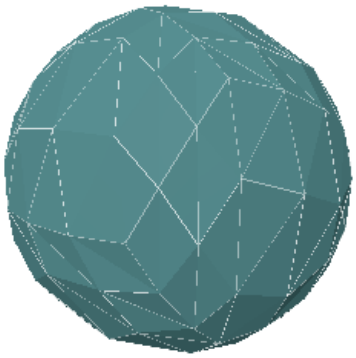
radius – радіус додекаедра. За замовчуванням 1;

detail – за замовчуванням 0. Встановлення значення, більшого за 0, додає вершини, роблячи його більше не додекаедром.



```
const radius = 7;
const geometry = new THREE.DodecahedronGeometry( radius );
```

Рисунок 2.9 – Додекаедр (12 сторін)



```
const radius = 7;
const detail = 2;
const geometry = new THREE.DodecahedronGeometry( radius, detail );
```

Рисунок 2.10 – Додекаедр (з використанням параметру detail)

2.6 IcosahedronGeometry

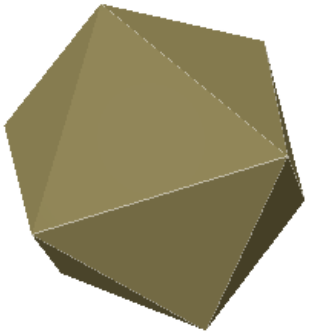
Клас для створення геометрії ікосаедра (рисунки 2.11 – 2.12).

Конструктор

IcosahedronGeometry(radius: Float, detail: Integer). Тут:

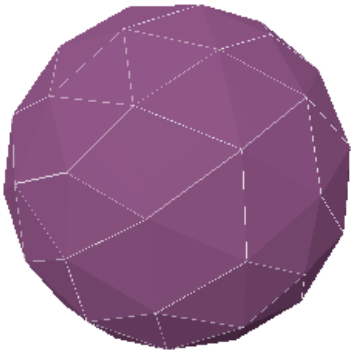
radius–за замовчуванням 1;

detail–за замовчуванням 0. Встановлення цього значення більше 0 додає більше вершин, роблячи його більше не ікосаедром. Коли деталь більше 1, це фактично сфера.



```
const radius = 7;
const geometry = new THREE.IcosahedronGeometry( radius );
```

Рисунок 2.11 – Ікосаедр (20 сторін)

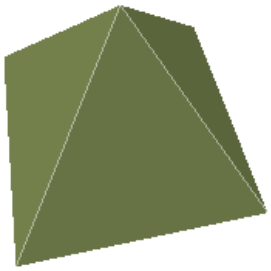


```
const radius = 7;
const detail = 2;
const geometry = new THREE.IcosahedronGeometry( radius, detail );
```

Рисунок 2.12 – Ікосаедр (з використанням параметру detail)

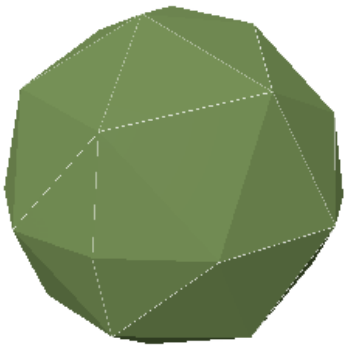
2.7 OctahedronGeometry

Клас для генерації геометрії октаедра (рисунки 2.13 – 2.14).
Конструктор
OctahedronGeometry(radius: Float, detail: Integer). Тут:
radius –радіус октаедра. За замовчуванням 1;
detail – за замовчуванням–0. Встановлення значення, більшого за нуль, додає вершини, щоб він більше не був октаедром.



```
const radius = 7;
const geometry = new THREE.OctahedronGeometry( radius );
```

Рисунок 2.13 – Октаедр (8 сторін)



```
const radius = 7;
const detail = 2;
const geometry = new THREE.OctahedronGeometry( radius, detail );
```

Рисунок 2.14 – Октаедр (з використанням параметру detail)

2.8 PlaneGeometry

Клас для створення плоских геометрій (рисунок 2.15 – 2.16).

Конструктор

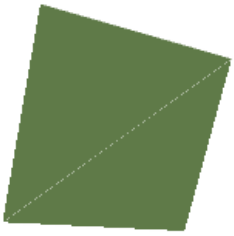
PlaneGeometry(width: Float, height: Float, widthSegments: Integer, heightSegments: Integer). Тут:

width–ширина по осі X. За замовчуванням 1;

height–висота по осі Y. За замовчуванням 1;

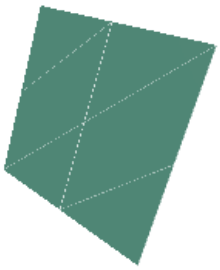
widthSegments–додатково. За замовчуванням 1;

heightSegments–додатково. За замовчуванням 1.



```
const width = 9;
const height = 9;
const geometry = new THREE.PlaneGeometry( width, height );
```

Рисунок 2.15 – Площина



```
const width = 9;
const height = 9;
const widthSegments = 2;
const heightSegments = 2;
const geometry = new THREE.PlaneGeometry(
    width, height,
    widthSegments, heightSegments );
```

Рисунок 2.16 – Площина (з використанням додаткових параметрів)

2.9 SphereGeometry

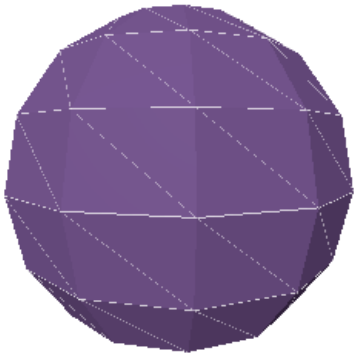
Клас для створення геометрій сфери (рисунки 2.17 – 2.18).

Конструктор

SphereGeometry(radius: Float, widthSegments: Integer, heightSegments: Integer, phiStart: Float, phiLength: Float, thetaStart: Float, thetaLength: Float). Тут:

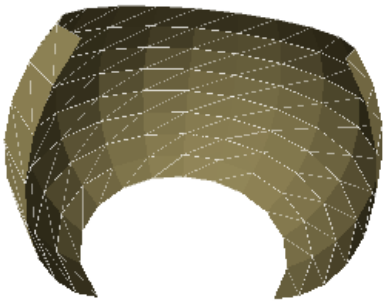
radius – радіус сфери. За замовчуванням 1;

widthSegments – кількість горизонтальних відрізків. Мінімальне значення – 3, а за замовчуванням – 32;



```
const radius = 7;
const widthSegments = 12;
const heightSegments = 8;
const geometry = new THREE.SphereGeometry( radius, widthSegments, heightSegments );
```

Рисунок 2.17 – Сфера



```
const radius = 7;
const widthSegments = 12;
const heightSegments = 8;
const phiStart = Math.PI * 0.25;
const phiLength = Math.PI * 1.5;
const thetaStart = Math.PI * 0.25;
const thetaLength = Math.PI * 0.5;
const geometry = new THREE.SphereGeometry(
    radius,
    widthSegments, heightSegments,
    phiStart, phiLength,
    thetaStart, thetaLength );
```

Рисунок 2.18 – Неповна сфера створена за допомогою різних значень phiStart, phiLength, thetaStart і thetaLength

heightSegments – кількість вертикальних сегментів. Мінімальне значення – 2, а за замовчуванням – 16;

phiStart – вкажіть горизонтальний початковий кут. За замовчуванням 0;

phiLength – вкажіть розмір кута горизонтальної розгортки. Типовим є Math.PI * 2;

thetaStart – вкажіть вертикальний початковий кут. За замовчуванням 0;

thetaLength – вкажіть розмір кута вертикальної розгортки. Типовим є Math.PI.

Геометрія створюється шляхом згортання та обчислення вершин навколо осі Y (горизонтальна розгортка) та осі Z (вертикальна розгортка). Таким чином, неповні сфери (схожі на «зрізи сфери») можуть бути створені за допомогою різних значень phiStart, phiLength, thetaStart і thetaLength, щоб визначити точки, в яких ми починаємо (або закінчуємо) обчислення цих вершин.

2.10 TetrahedronGeometry

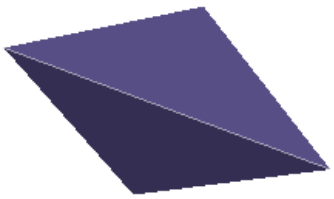
Клас для генерації геометрії тетраедра (рисунки 2.19 – 2.20).

Конструктор

TetrahedronGeometry(radius: Float, detail: Integer). Тут:

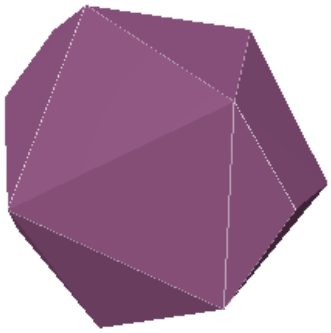
radius – радіус тетраедра. За замовчуванням 1;

detail – за замовчуванням 0. Встановлення значення, більшого за 0, додає вершини, роблячи його більше не тетраедром.



```
const radius = 7;
const geometry = new THREE.TetrahedronGeometry( radius );
```

Рисунок 2.19 – Тетраедр (4 сторони)

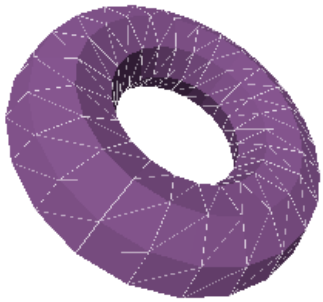


```
const radius = 7;
const detail = 2;
const geometry = new THREE.TetrahedronGeometry( radius, detail );
```

Рисунок 2.20 – Тетраедр (з використанням параметру detail)

2.11 TorusGeometry

Клас для створення геометрій тора (рисунок 2.21).



```
const radius = 5;
const tubeRadius = 2;
const radialSegments = 8;
const tubularSegments = 24;
const geometry = new THREE.TorusGeometry(
    radius, tubeRadius,
    radialSegments, tubularSegments );
```

Рисунок 2.21 – Тор

Конструктор

TorusGeometry(radius: Float, tube: Float, radialSegments: Integer, tubularSegments: Integer, arc: Float). Тут:

radius – радіус тора від центра тора до центру труби. За замовчуванням 1;

tube – радіус труби. За замовчуванням 0,4;

radialSegments – за замовчуванням 12;

tubularSegments – за замовчуванням 48;

arc – центральний кут. Типовим є Math.PI * 2.

2.12 Використання бібліотеки dat.GUI

Бібліотека під назвою dat.GUI дозволяє нам легко створювати компоненти інтерфейсу користувача. Використовуємо цю бібліотеку, додаючи інтерфейс користувача до нашого прикладу, що дозволить нам керувати обертанням куба.

Як і для інших бібліотек, ми спочатку додамо dat.GUI елемент на нашій сторінці, використовуючи наступний код:

```
<script src="./js/dat.gui.js"></script>
```

Наступне, що нам необхідно зробити – це налаштувати JavaScript об’єкт, який аналізуватиме властивості, які ми хочемо змінити, використовуючи бібліотеку dat.GUI. В основній частині нашого коду ми додамо наступний JavaScript об’єкт:

```
var controls = new function() {
        this.rotationSpeed = 0.02;
}
```

У цьому об’єкті ми визначимо властивість: this.rotationSpeed разом з його значенням за замовчуванням. Далі ми передаємо цей об’єкт у новий об’єкт dat.GUI і визначаємо діапазон цих двох властивостей, як показано нижче:

```
var gui=new dat.GUI();
gui.add(controls, ‘rotationSpeed’,0,0.5);
```

Властивість rotationSpeed встановлено в діапазоні від 0 до 0,5. Все, що нам необхідно зараз зробити – це переконатися, що всередині циклу нашої функції render ми посилаємося на цю властивість безпосередньо, так що коли ми вносимо зміни за допомогою інтерфейсу користувача dat.GUI, вони відразу впливають на обертання нашого об’єкта. Це робиться так (додаток Д):

```
function renderScene()
{
```

```
    cube.rotation.x += controls.rotationSpeed;
    cube.rotation.y += controls.rotationSpeed;
    cube.rotation.z += controls.rotationSpeed;
    requestAnimationFrame( renderScene );
    trackballcontrols.update();
    renderer.render (scene, camera);
}
```

Тепер, при запуску нашого прикладу, ви побачите простий інтерфейс користувача (рисунок 2.22), який можна використовувати для управління швидкістю обертання об’єкта.

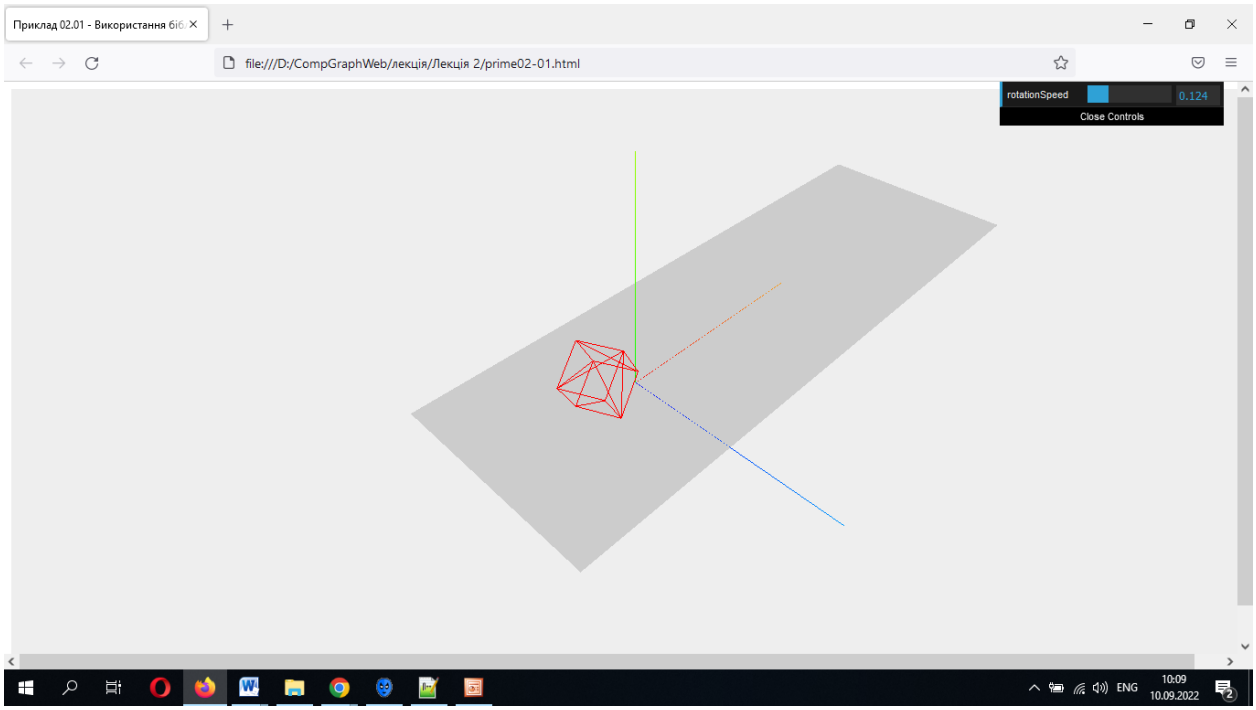


Рисунок 2.22 – Інтерфейс користувача

Завдання

На підставі 1 лабораторної роботи (вибраних Вами власних геометричних фігур):

– використати бібліотеку dat.GUI для створення компонент інтерфейсу користувача (додаток Д);
– надіслати звіт і архів всіх файлів (включаючи папку з бібліотеками).

Рекомендована література

1. Dirksen J. Learn Three.js : Program 3D Animations and Visualizations for the Web with JavaScript and WebGl. Packt Publishing, Limited, 2023. 554 p.
2. Фреймворк three.js : офіційний сайт. URL : http://threejs.org (дата звернення : 18.01.2024).

Контрольні запитання

1. Що таке примітиви в бібліотеці three.js?
2. Які основні типи примітивів підтримуються в three.js?
3. Як створити куб за допомогою класу BoxGeometry в three.js?
4. Як створити сферу за допомогою класу SphereGeometry в three.js?
5. Як створити площину за допомогою класу PlaneGeometry в three.js?
6. Як встановити матеріал (колір, текстуру тощо) для примітиву в three.js?
7. Як встановити позицію та розмір примітиву в three.js?
8. Як додати примітив на сцену у three.js?
9. Як анімувати примітиви в three.js?
10. Як створити компоненти інтерфейсу користувача?

3 ДОДАВАННЯ ТЕКСТУРИ НА ОБ'ЄКТ. МАТЕРІАЛИ THREE.JS

З міркувань безпеки, WebGL не може безпосередньо використовувати зображення з жорсткого диска. Це означає, що для розробки вам потрібно використовувати веб-сервер.

Three.js надає кілька типів матеріалів. Вони визначають, як об’єкти з’являтимуться на сцені. Які матеріали ви використовуєте залежить від того, чого ви намагаєтеся досягти.

Розглянемо основний набір матеріалів Three.js.

Якщо ви віддаєте перевагу веб-серверу з інтерфейсом користувача, то є Servez (https://greggman.github.io/servez/) – це окрема програма, на якій працює простий веб-сервер, з графічним інтерфейсом для запуску/зупинки та вибору папки для обслуговування (рисунок 3.1).

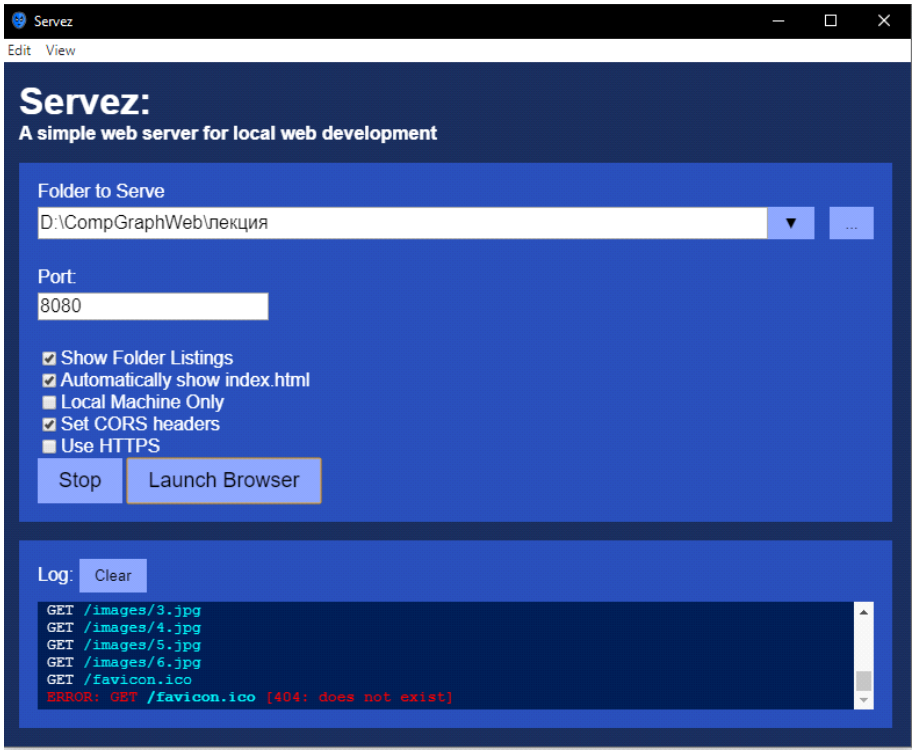


Рисунок 3.1 – Програма, на якій працює простий веб-сервер

Текстура – це растрове зображення, яке проєктується на межі тривимірного об’єкта за певними правилами.

Для накладання текстур Three.js є спеціальний клас ImageUtils з методом loadTexture, головним параметром якого є адреса (шлях) до зображення текстури. На основі текстури Texture створюється матеріал із параметром map: Texture. Далі – все як завжди, створюється фігура із заданою геометрією та нашим матеріалом.

Map – найочевидніший спосіб проєктування, де колір пікселя текстури стає кольором фрагмента грані.

Візьмемо зображення і помістимо його в папку "/images/" та використаємо дане зображення в якості текстури (додаток Е, рисунок 3.2).

```
var cube_geometry=new THREE.CubeGeometry(4, 4, 4);
//Накладення однієї текстури
var cube_material=new THREE.MeshBasicMaterial({map: THREE.ImageUtils.loadTexture (‘./images/1.jpg’), overdraw:true});
cube=new THREE.Mesh(cube_geometry, cube_material);
scene.add(cube);
```

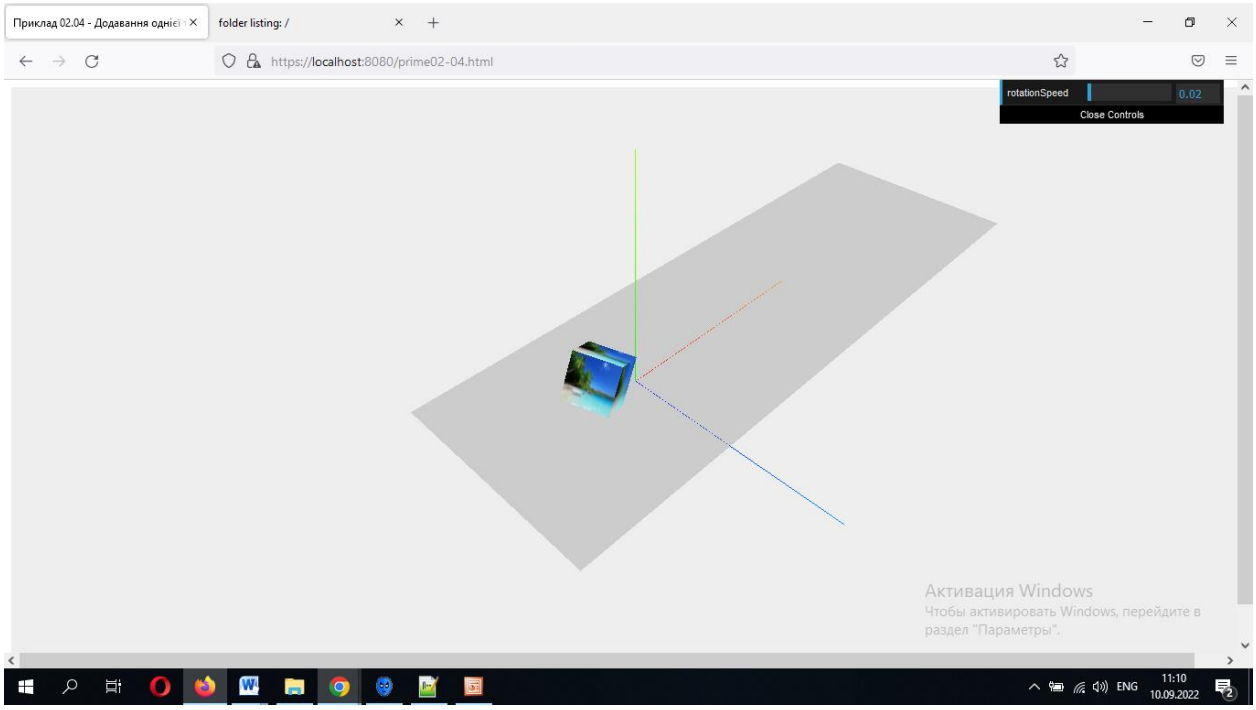


Рисунок 3.2 – Додавання однієї текстури на об’єкт (клас ImageUtils)

Якщо необхідно накласти на різні боки різні текстури, необхідно створити масив матеріалів (додаток Ж, рисунок 3.3).

```
var materials = [];
for (i=1; i<=6; i++) {
var Texture = New THREE.ImageUtils.loadTexture(‘./images/’ + String(i) + ‘.jpg’);
var Material = New THREE.MeshBasicMaterial({ map: Texture, color: 0x00dcff} );
materials.push( Material );}
```

Далі робимо аналогічно: збираємо матеріал, оголошуємо геометрію та створюємо куб:

```
var material = new THREE.MeshFaceMaterial(materials);
var geometry = New THREE.BoxGeometry( 4, 4, 4);
var cube = new THREE.Mesh(geometry, material);
```

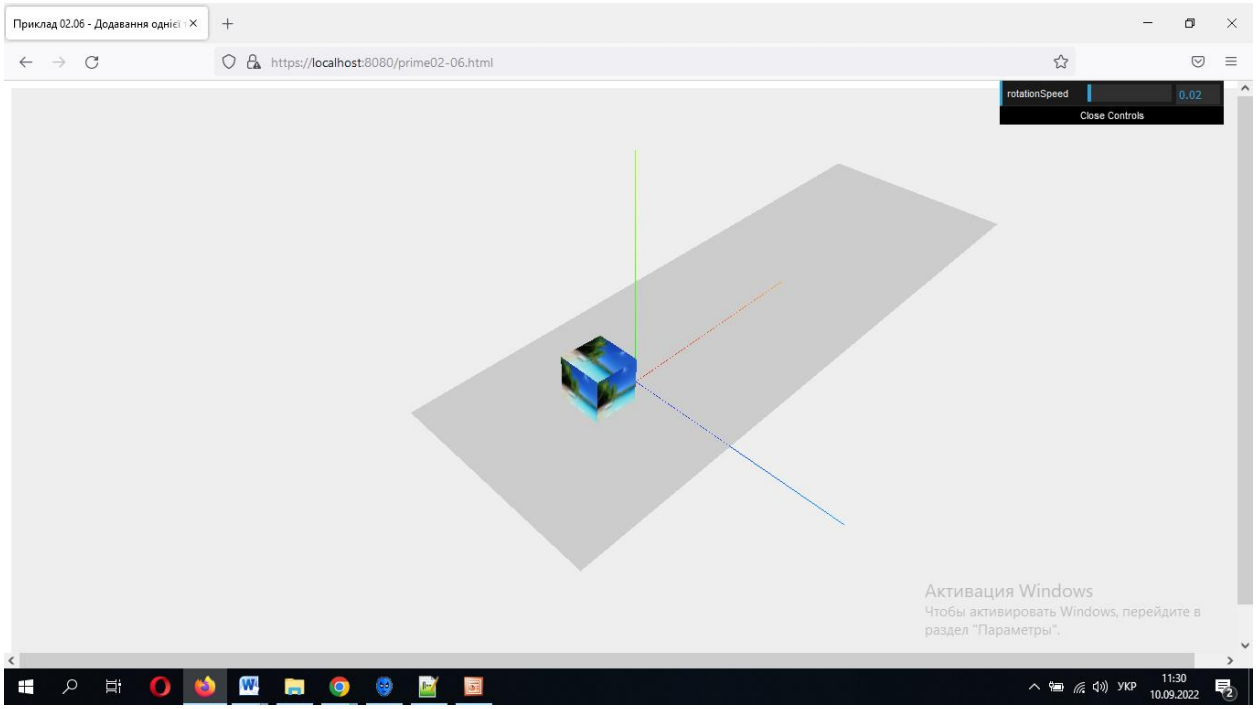


Рисунок 3.3 – Декілька текстур на один об’єкт (клас ImageUtils)

3.1 Клас для завантаження текстури TextureLoader

Створюємо TextureLoader (клас для завантаження текстури), а потім викликаємо його метод load. Метод починає завантаження із зазначеної URL-адреси, а також повертає новий об’єкт текстури, який може бути використаний безпосередньо для створення матеріалу. Приклад накладання однієї тектури наведено в додатку И, рисунок 3.4.

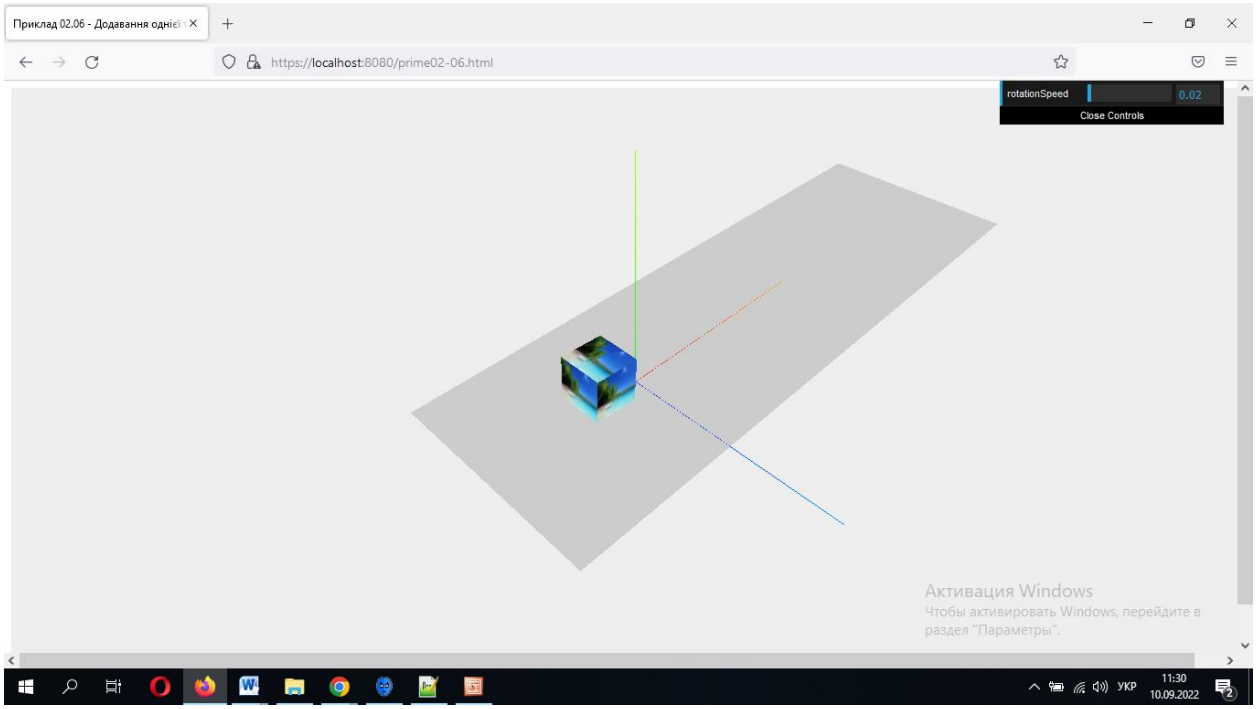


Рисунок 3.4 – Додавання однієї текстури на об’єкт (клас TextureLoader)

Накладання однієї текстури з використанням класу TextureLoader:

```
var cube_geometry = New THREE.CubeGeometry(4, 4, 4);
var cube_texture = New THREE.TextureLoader().load( ‘./images/1.jpg’ );
var cube_material = New THREE.MeshBasicMaterial({map: cube_texture});
var cube = New THREE.Mesh (cube_geometry, cube_material);
```

Накладання декількох текстур з використанням класу TextureLoader (додаток К, рисунок 3.5):

```
var materials = [];
for (i=1; i<=6; i++) {
var Texture = New THREE.TextureLoader().load(‘./images/’ + String(i) + ‘.jpg’);
var Material = New THREE.MeshBasicMaterial({ map: Texture, color: 0x00dcff} );
materials.push( Material );}
var material = new THREE.MeshFaceMaterial(materials);
var geometry = New THREE.BoxGeometry( 4, 4, 4);
var cube = new THREE.Mesh(geometry, material);
```

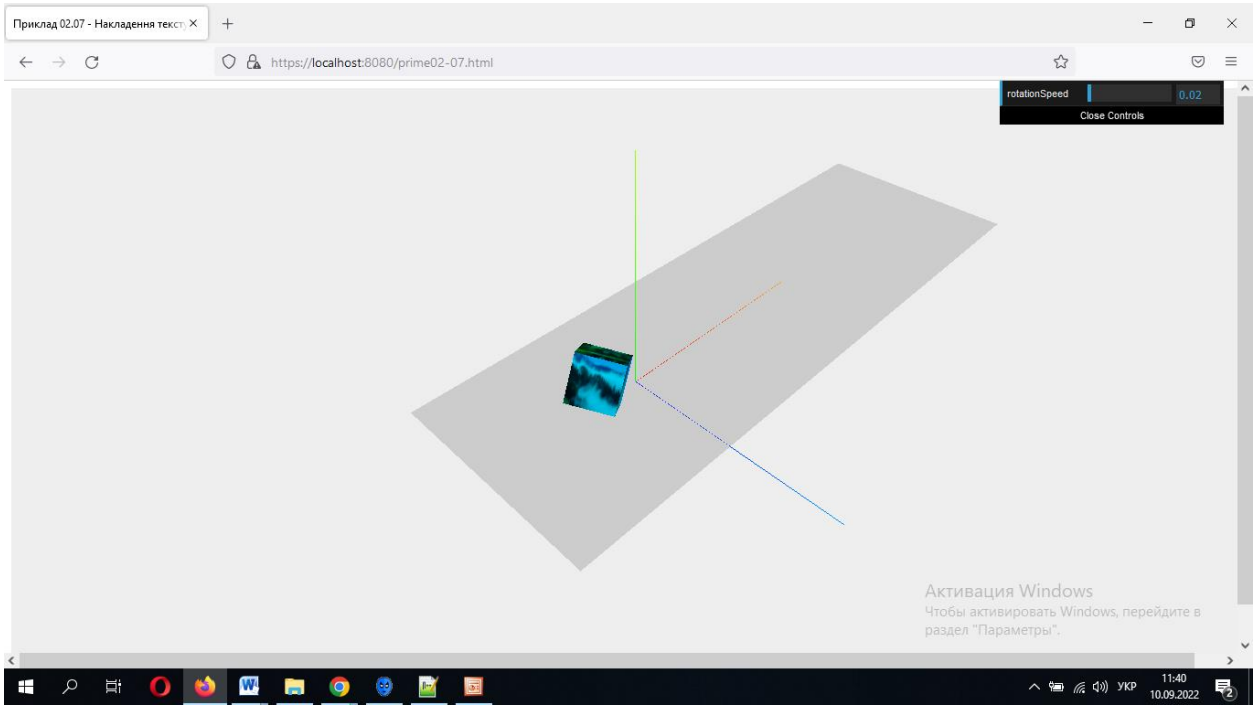


Рисунок 3.5 – Декілька текстур на один об’єкт (клас TextureLoader)

3.2 Повторення, зміщення, обертання текстури

Текстури мають налаштування для повторення, зміщення та повороту текстури.

За замовчуванням текстури в three.js не повторюються. Щоб встановити, чи повторюється текстура, є 2 властивості: wrapS – для горизонтального та wrapT – для вертикального перенесення.

Вони можуть мати одне з наступних значень:

– THREE.ClampToEdgeWrapping– останній піксел текстури розтягується до краю сітки (mesh);

– THREE.RepeatWrapping – текстура повторюється до безкінечності;

– THREE.MirroredRepeatWrapping – текстура повторюватиметься нескінченно, дзеркально відображається при кожному повторі.

Наприклад, щоб увімкнути поворот в обох напрямках:

```
someTexture.wrapS=THREE.RepeatWrapping;
someTexture.wrapT=THREE.RepeatWrapping;
```

Повторення визначається властивістю повторення [repeat] (рис. 3.6):

```
someTexture1.wrapS=THREE.RepeatWrapping;
someTexture1.wrapT=THREE.RepeatWrapping;
const timesToRepeatHorizontally = 4;
const timesToRepeatVertically = 2;
someTexture1.repeat.set(timesToRepeatHorizontally,
timesToRepeatVertically);
var cube_geometry1=new THREE.CubeGeometry(4, 4, 4);
var cube_material1=new
THREE.MeshBasicMaterial({map:someTexture1});
cube1=new THREE.Mesh(cube_geometry1, cube_material1);
```

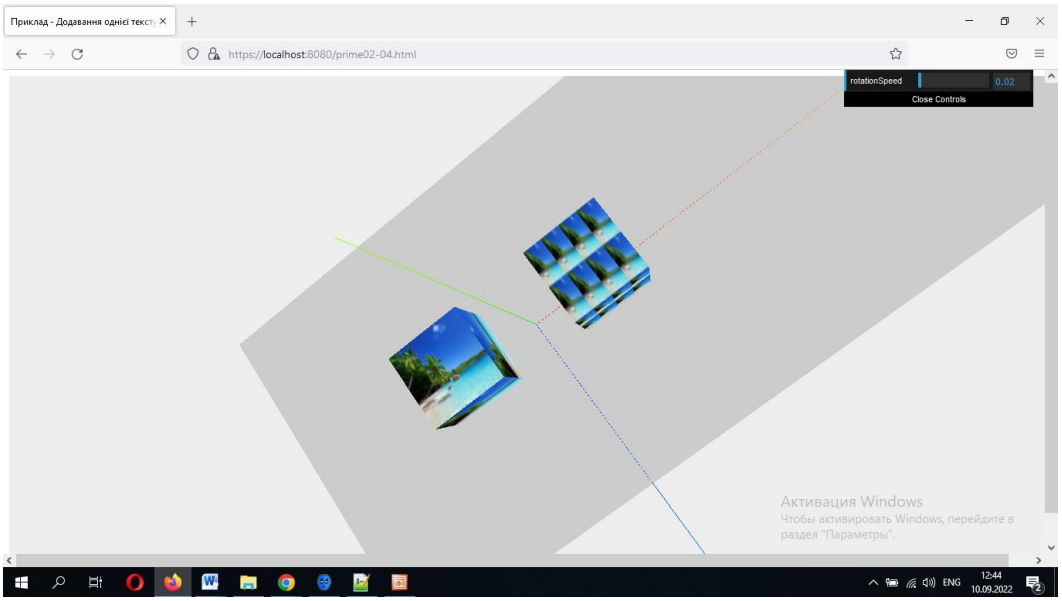


Рисунок 3.6 – Повторення текстури

Зміщення текстури можна виконати, задавши offset властивість. Текстури зміщені з одиницями, де 1 одиниця = 1 розмір текстури. Іншими словами, 0 = без зміщення та 1 = зміщення на одну повну кількість текстури (рисунок 3.7):

```
someTexture1 = THREE.ImageUtils.loadTexture('./images/1.jpg');
var cube_geometry1=new THREE.CubeGeometry(4, 4, 4);
const xOffset = .25; // Зміщення на 1/4 текстури
const yOffset = .25; // Зміщення на 1/4 текстури
someTexture1.offset.set(xOffset, yOffset);
var cube_material1=new THREE.MeshBasicMaterial({map:someTexture1
});
cube1=new THREE.Mesh(cube_geometry1, cube_material1);
```

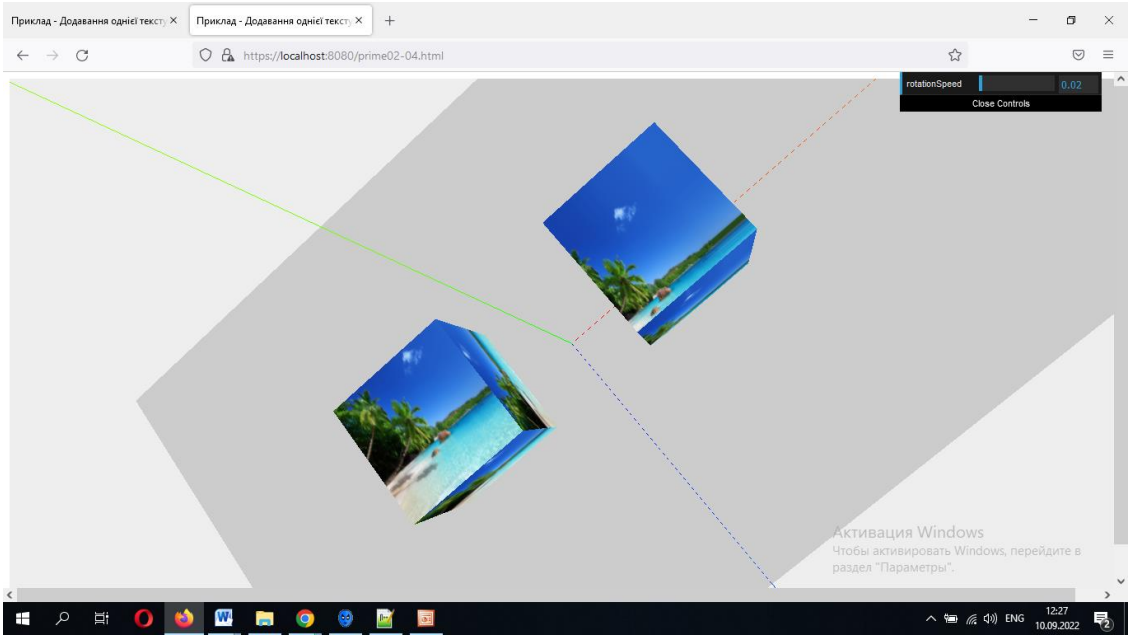


Рисунок 3.7 – Зміщення текстури

Поворот текстури можна задати, задавши rotation властивість у радіанах, а також center властивість для вибору центру обертання. За замовчуванням 0,0, що обертається від нижнього лівого кута. Як і у разі усунення, ці блоки мають розмір текстури, тому їх установка .5, .5 обертатиметься навколо центру текстури (рисунок 3.8):

```
someTexture1 = THREE.ImageUtils.loadTexture('./images/1.jpg');
var cube_geometry1=new THREE.CubeGeometry(4, 4, 4);
someTexture1.center.set(.5, .5);
someTexture1.rotation = THREE.MathUtils.degToRad(45);
var cube_material1=new
THREE.MeshBasicMaterial({map:someTexture1});
cube1=new THREE.Mesh(cube_geometry1, cube_material1);
```

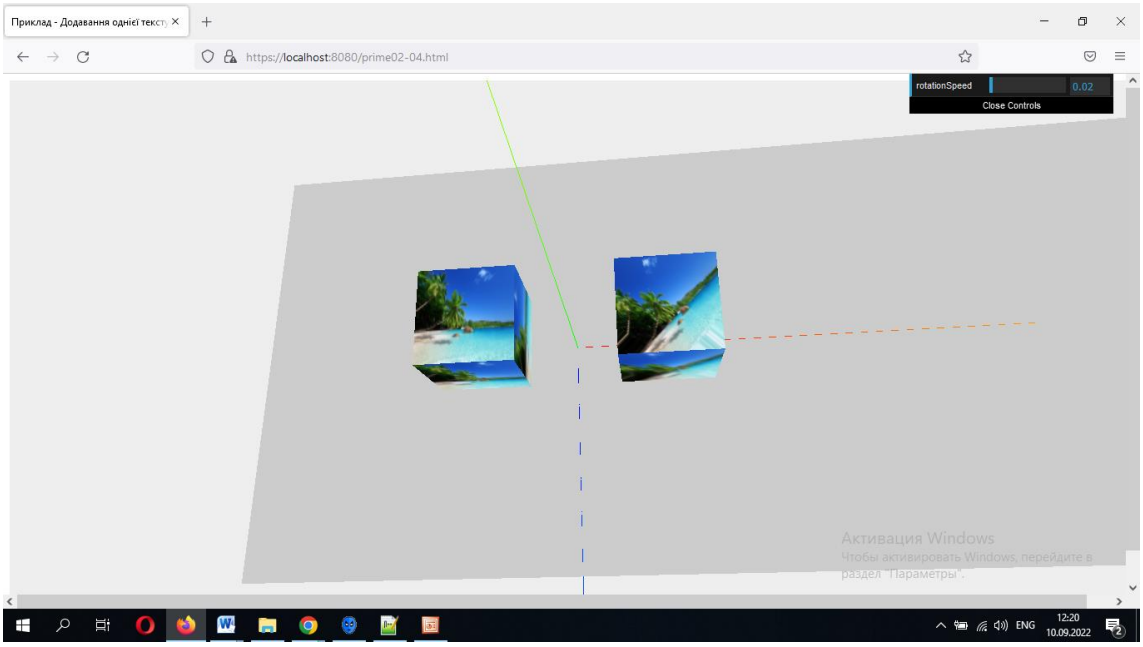


Рисунок 3.8 – Поворот текстури

Якщо ви змініте wrapS або wrapT на текстуру, ви також повинні встановити texture.needsUpdate. Інші налаштування застосовуються автоматично.

3.3 MeshBasicMaterial

color: колір матеріалу за замовчуванням встановлено білий (0xffffff) (рисунок 3.9).

```
var cubeGeometry = new THREE.CubeGeometry(4,4,4);
var cubeMaterial = new THREE.MeshBasicMaterial({color: 0xff0000});
var cube = new THREE.Mesh(cubeGeometry, cubeMaterial);
```

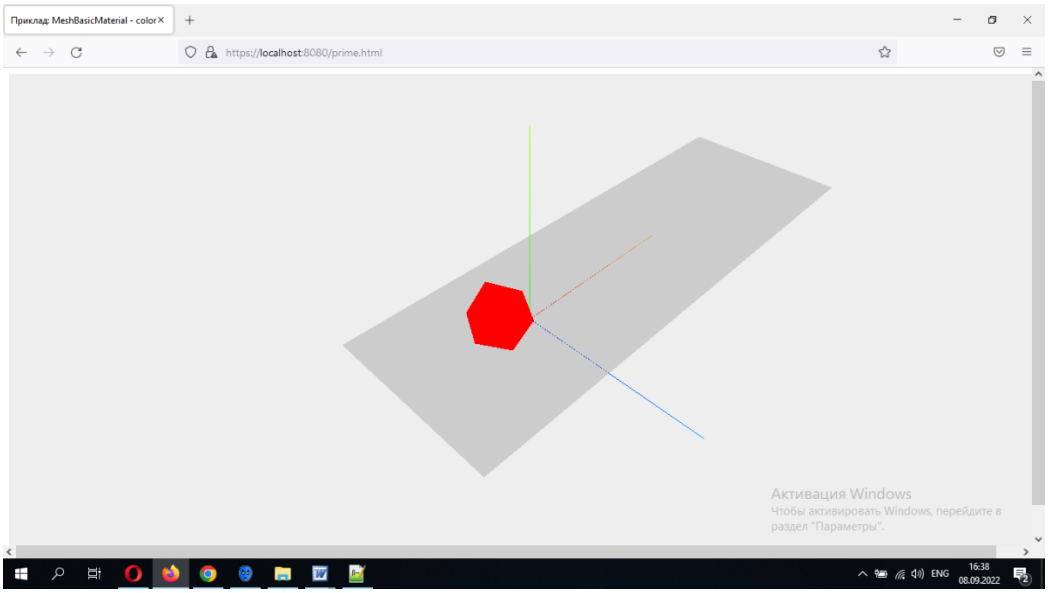


Рисунок 3.9 – Властивість color

wireframe – (каркас) властивість дозволяє відобразити геометричну фігуру як каркас. Значення за замовчуванням встановлено false (тобто каркас не видно і візуалізація відбувається за допомогою плоских багатокутників (рисунок 3.10).

```
var cubeGeometry = new THREE.CubeGeometry(4,4,4);
var cubeMaterial = new THREE.MeshBasicMaterial({color: 0xff0000,wireframe: true});
var cube = new THREE.Mesh(cubeGeometry, cubeMaterial);
```

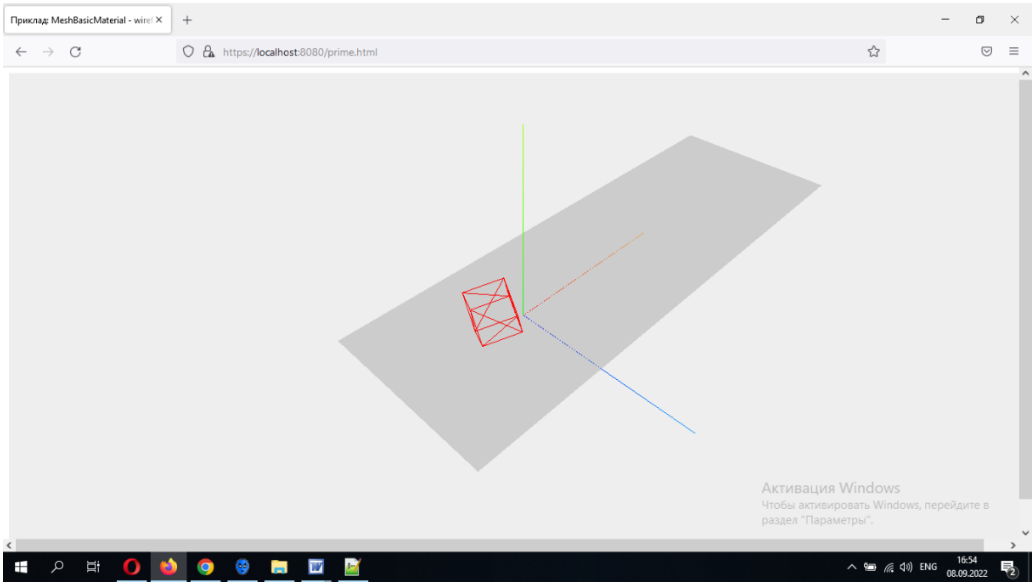


Рисунок 3.10 – Властивість wireframe

transparent **(прозорий)**: визначає, чи цей матеріал є прозорим. Це впливає на рендеринг, оскільки прозорі об’єкти вимагають особливої обробки та відображаються після непрозорих об’єктів. Якщо встановлено значення true, рівень прозорості матеріалу контролюється установкою його властивості opacity (непрозорість). За замовчуванням – false (рисунок 3.11).

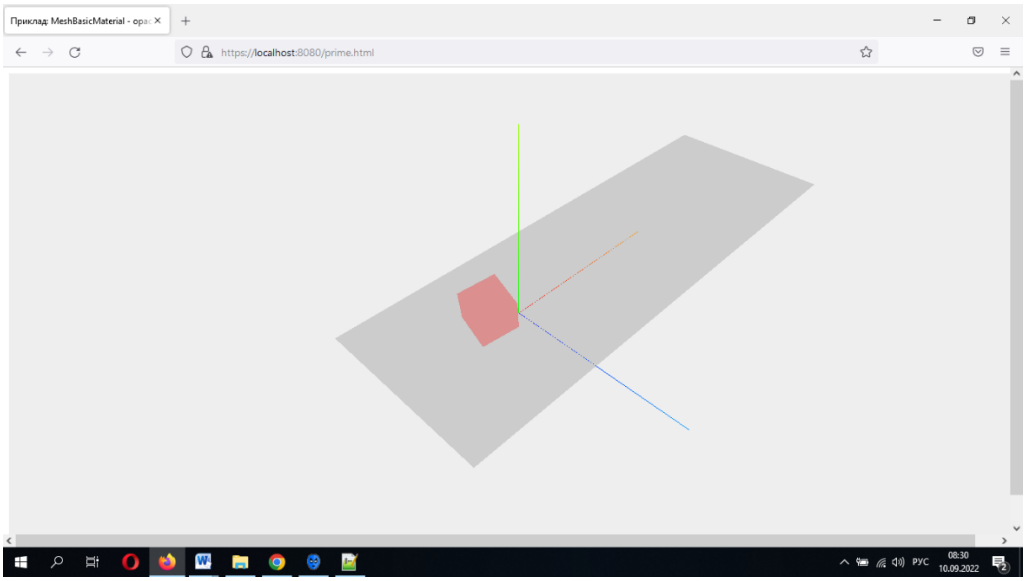


Рисунок 3.11 – Властивість transparent

opacity [oʊˈpæs.ə.ṭi] (непрозорість) – змінюється в діапазоні 0,0 – 1,0, що вказує, наскільки прозорий матеріал. Значення 0,0 вказує на повну прозорість, 1,0 – на повну непрозорість. Якщо для властивості прозорості матеріалу не встановлено значення true, матеріал залишиться повністю непрозорим, і це значення впливатиме лише на його колір:

```
var cubeGeometry = new THREE.CubeGeometry(4,4,4);
var cubeMaterial = new THREE.MeshBasicMaterial({color: 0xff0000, transparent:true,opacity:0.3});
var cube = new THREE.Mesh(cubeGeometry, cubeMaterial);
```

Side (сторона): визначає, яка сторона граней відображатиметься – передня, задня або обидві. За замовчуванням FrontSide (рисунок 3.12). Значення: THREE.FrontSide – видно зовні (у напрямку нормалей), THREE.BackSide – видно зсередини (рисунок 3.13), THREE.DoubleSide – видно з обох боків (рисунок 3.14).

```
var cubeGeometry = new THREE.CubeGeometry(4,4,4);
var cubeMaterial = new THREE.MeshBasicMaterial({color: 0xff0000, transparent: true, opacity:0.5, side: THREE.FrontSide });
var cube = new THREE.Mesh(cubeGeometry, cubeMaterial);
```

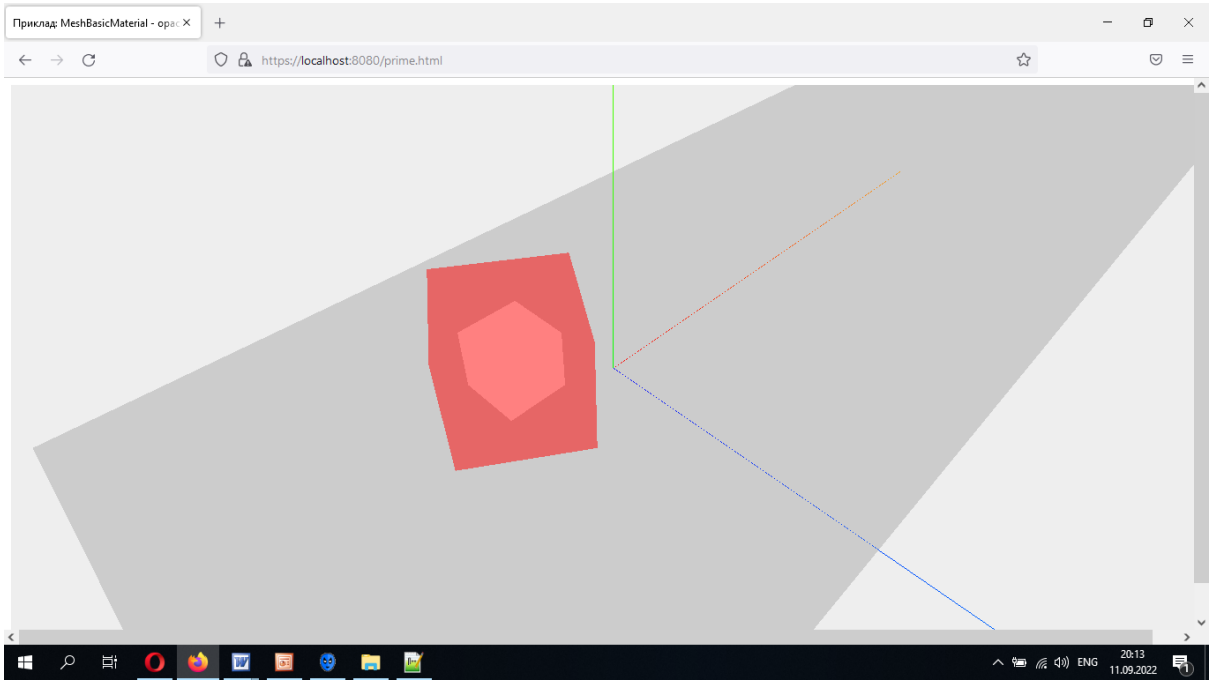


Рисунок 3.12 – Властивість THREE.FrontSide

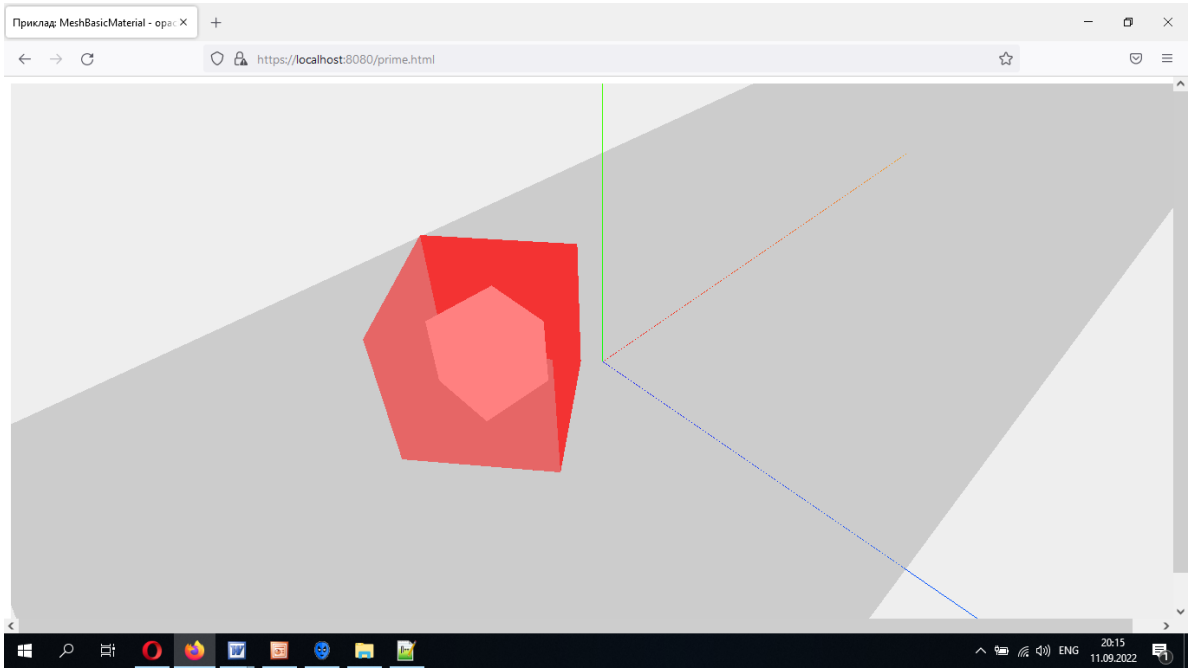


Рисунок 3.13 – Властивість THREE.DoubleSide

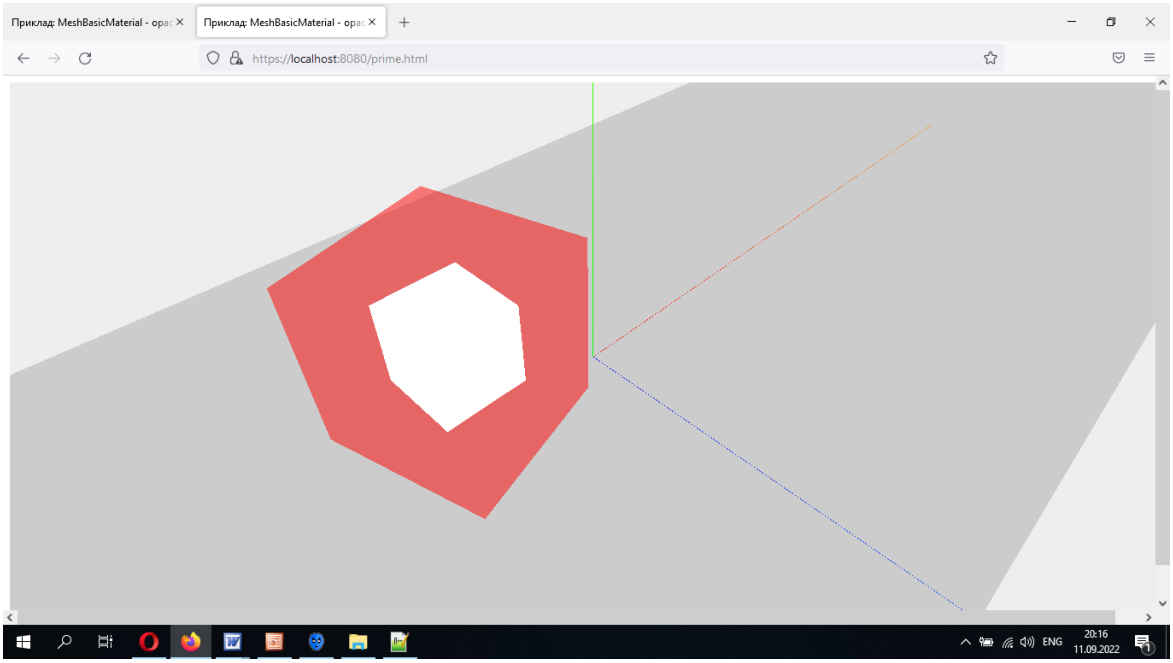


Рисунок 3.14 – Властивість THREE.BackSide

3.4 MeshLambertMaterial

MeshLambertMaterial обчислює освітлення лише у вершинах vs. Матеріал для поверхонь без блиску. Матеріал добре імітує такі поверхні, як необроблене дерево або камінь, але не може імітувати блискучі поверхні із дзеркальними відблисками, такі як лакована деревина.

Через простоту моделей відображення та освітлення продуктивність буде вищою при використанні цього матеріалу замість MeshPhongMaterial, MeshStandardMaterial або MeshPhysicalMaterial.

3.5 MeshPhongMaterial

MeshPhongMaterial, який обчислює освітлення в кожному пікселі, а також підтримує відблиски. Матеріал для блискучих поверхонь із дзеркальними відблисками. MeshPhongMaterial може імітувати блискучі поверхні із дзеркальними відблисками, наприклад, лаковане дерево.

Як правило, продуктивність буде вищою при використанні цього матеріалу замість MeshStandardMaterial або MeshPhysicalMaterial.

Shininess встановлює ***MeshPhongMaterial*** визначаючи блиск від відблисків. Значення за замовчуванням – 30.

3.6 MeshStandardMaterial

MeshStandardMaterial – на основі фізичних властивостей, який використовує робочий процес Metallic-Roughness. Фізичний рендеринг (PBR – physically based rendering) нещодавно став стандартом у багатьох 3D-додатках, таких як *Unity, 3DStudio Max*. На практиці це дає точніший і реалістичніший результат, ніж MeshLambertMaterial або MeshPhongMaterial за рахунок того, що це дещо витратніше в обчислювальному відношенні (додаток Л).

3.7 alphaMap (карта прозорості)

Визначає прозорість фрагментів полігонів.

Для роботи карти прозорості необхідно увімкнути властивість матеріалу transparent (за замовчуванням false).

Карта прозорості працює на зелений канал текстури. FF – повністю непрозорий, 00 – повністю прозорий.

Ви можете використовувати параметр alphaTest. Він приймає значення від 0 (за замовчуванням) до 1. При цьому не будуть рисуватись фрагменти з прозорістю менше величини alphaTest.

Для кращого ефекту включимо відображення із двох сторін (додаток М)

```
var alphamap = New THREE.ImageUtils.loadTexture('./images/1.jpg');
alphamap.wrapS=THREE.RepeatWrapping;
alphamap.wrapT=THREE.RepeatWrapping;
alphamap.repeat.set( 1, 1 );
var cube_material = New THREE.MeshPhongMaterial({transparent:true,
color:0xFF0000, alphaMap:alphamap, alphaTest:0.5, side:
THREE.DoubleSide });
cube_geometry=new THREE.CubeGeometry(4, 4, 4);
cube=new THREE.Mesh(cube_geometry, cube_material);
scene.add(cube);
```

Завдання

На підставі 1 лабораторної роботи (вибраних Вами власних геометричних фігур):

- додати одну текстуру на об’єкт (клас ImageUtils) (додаток Е);
- додати кілька текстур на об’єкт (клас ImageUtils) (додаток Ж);
- додати одну текстуру на об’єкт (клас TextureLoader) (додаток И) ;
- додати кілька текстур на об’єкт (клас TextureLoader) (додаток К);
- додати карту прозорості (додаток М);
- застосувати деякі властивості матеріалу (color, opacity, transparent…) (додаток Л);
- використати матеріали Three.js;
- надіслати звіт та архів усіх файлів (включаючи папку з бібліотеками js, папку images…).

Рекомендована література

1. Dirksen J. Learn Three.js : Program 3D Animations and Visualizations for the Web with JavaScript and WebGl. Packt Publishing, Limited, 2023. 554 p.
2. Фреймворк three.js : офіційний сайт. URL : http://threejs.org (дата звернення : 18.01.2024).

Контрольні запитання

1. Як додати текстуру на поверхню об’єкта в three.js?
2. Як завантажити зображення для використання як текстура в three.js?
3. Як налаштувати параметри текстури, такі як повторення, розмір та спосіб налаштування відображення текстури на об’єкті?
4. Як використовувати текстури з альфа-каналом для створення прозорих об’єктів?
5. Як додати множинні текстури на один об’єкт в three.js?
6. Як змінити або оновити текстуру об’єкта в реальному часі?
7. Як застосувати текстуру на лише певну частину об’єкта, наприклад, на одну зі сторін куба?
8. Що таке матеріал в бібліотеці three.js і яку роль він відіграє в рендерингу сцени?
9. Які основні типи матеріалів підтримуються в three.js і в чому їхні особливості?
10. Як застосувати матеріал до об’єкта в three.js?
11. Як змінити кольори, прозорість, блиск або інші властивості матеріалу в three.js?
12. Як використовувати текстури як частину матеріалу в three.js?
13. Як створити власний матеріал зі змінними властивостями в three.js?
14. Як створити емісійний матеріал, який випромінює світло в three.js?
15. Як створити матеріал з прозорістю або прозорим ефектом в three.js?

4 ОГЛЯД ПОМІЧНИКІВ

Більшість програмістів зазвичай мають труднощі з розумінням тривимірного простору. Також у нас можуть виникнути труднощі з розумінням того, як працює різне світло або навіть як осі розташовані у просторі. Сьогодні ми зможемо розібратися із цими питаннями. У Three.js є все необхідне для цього – помічники. У прикладах ми розглянемо демонстрацію всіх існуючих помічників: ArrowHelper, AxisHelper, BoundingBoxHelper, CameraHelper,DirectionalLightHelper, GridHelper, HemisphereLightHelper, PointLightHelper, SpotLightHelper.

Для початку створимо просту сцену, яка включає наступні елементи: камеру, пряме світло, площину і дві сфери (додаток Н).

4.1 ArrowHelper

Він рисує тривимірну стрілку (починаючи з початку у напрямку dir на певній довжині) у просторі. Це допомагає зрозуміти напрямок вектору в просторі.

Конструктор

ArrowHelper (dir: Vector3, origin:Vector3, length:Number, hex:Number, headLength:Number, headWidth:Number). Тут:

dir – напрямок від вихідної точки. Має бути одиничним вектором;

origin – точка початку стрілки;

length – довжина стрілки. За замовчуванням 1;

hex – шістнадцяткове значення визначення кольору. За замовчанням 0xffff00 (Yellow);

headLength – довжина вістря стрілки. За замовчуванням – 0,2* довжина;

headWidth – ширина вістря стрілки. За замовчуванням – 0,2*headLength.

Щоб додати цього помічника, використовуйте наступний код (додаток П, рисунок 4.1):

Vector3 – Клас, який представляє тривимірний вектор.

```
var directionV3 = New THREE.Vector3(1, 0, 1);
var originV3 = New THREE.Vector3(0, 200, 0);
var arrowHelper = New THREE.ArrowHelper(directionV3, originV3, 100, 0xff0000, 20, 10);
// directionV3 – напрямок від вихідної точки,
// originV3 – точка початку стрілки,
//100 – довжина стрілки,
//колір – червоний,
//20 – довжина вістря стрілки,
// 10 – ширина вістря стрілки.
scene.add(arrowHelper);
```

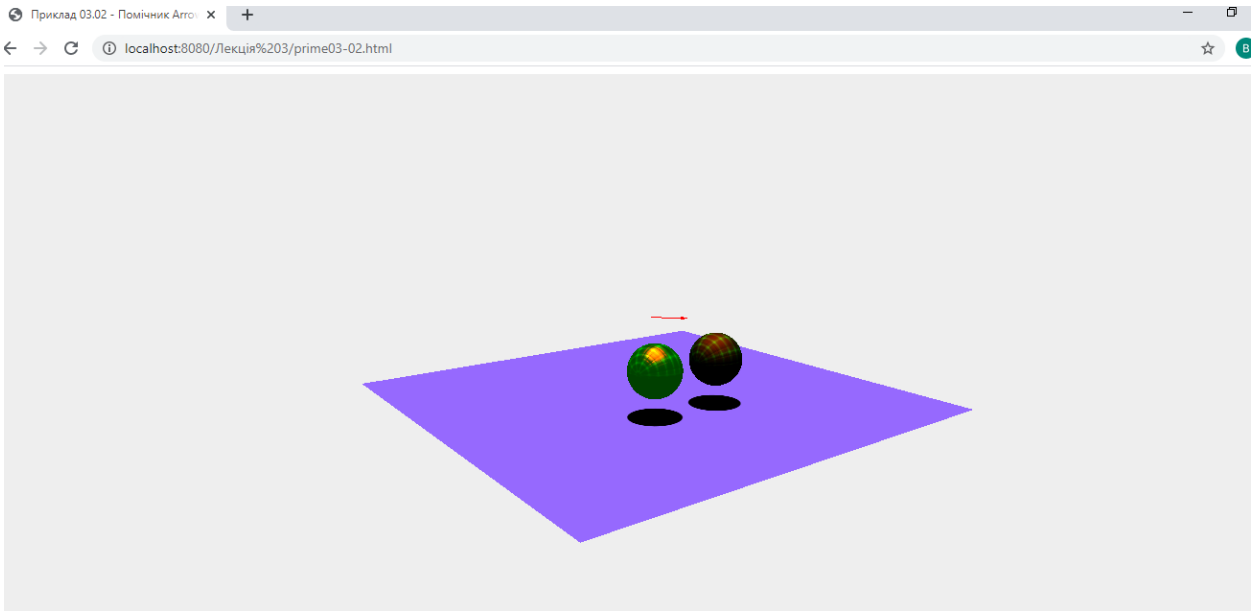


Рисунок 4.1 – ПомічникArrowHelper

4.2 AxisHelper

Він рисує об’єкт осі для простої візуалізації трьох осей. Вісь X червона. Вісь Y – зелена. Вісь Z синього кольору. Це допомагає зрозуміти напрямок всіх трьох осей у просторі.

Конструктор

AxesHelper (розмір: Число). Тут:

size – (необов’язково) розмір ліній, що становлять осі. За замовчуванням 1.

Щоб додати цього помічника, використовуйте наступний код (рисунок 4.2):

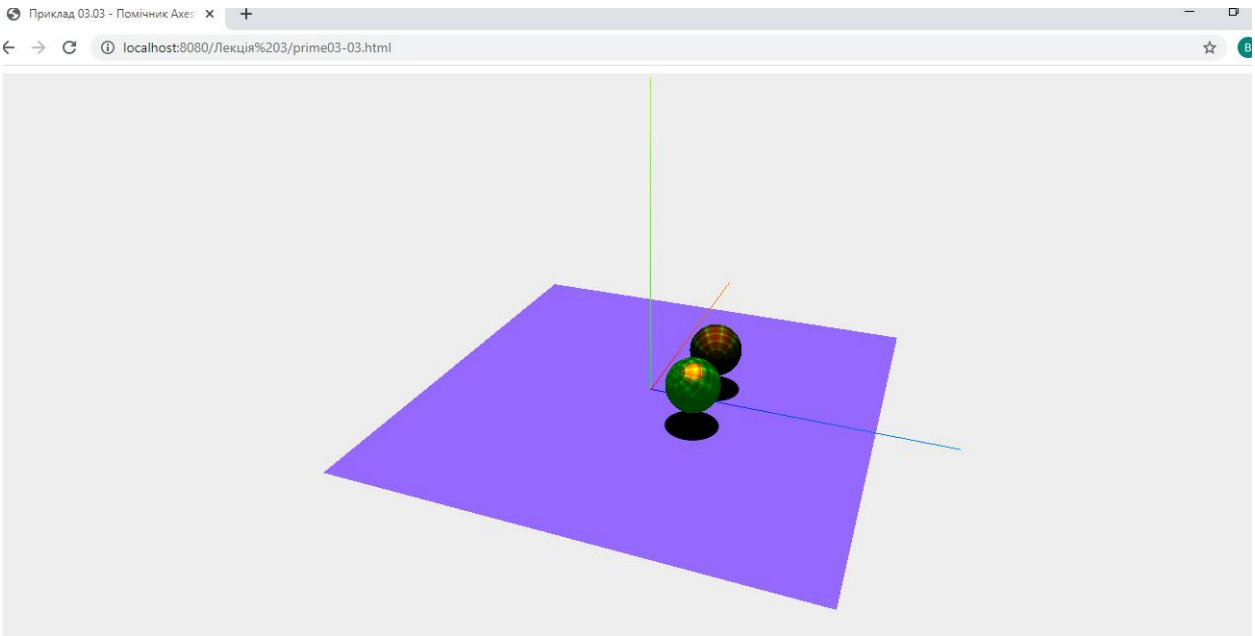


Рисунок 4.2 – Помічник AxesHelper

```
var axisHelper = New THREE.AxisHelper(800); // 800 is size
scene.add(axisHelper);
```

4.3 BoxHelper

Він рисує лінійний об’єкт в рамці будь-якого об’єкта (Object3D), щоб показати обмежувальну рамку, вирівняну по світовій осі для цього об’єкта.

Конструктор

BoxHelper (об’єкт: Object3D, колір: Color). Тут:

object – (необов’язково) object3D для відображення рамки, яка обмежує, вирівняної по світовій осі;

color – (необов’язково) шістнадцяткове значення, яке визначає колір поля. За промовчанням 0xffff00 (жовтий).

Щоб додати цього помічника, використовуйте наступний код:

```
// створити new group (Object3D)
var group = new THREE.Object3D();

// додати дві сфери
var sphere  = drawSphere(-100, 150, new THREE.MeshPhongMaterial({ map: texture, bumpMap: textureBump, color: 0x00ff00, specular: 0xff2200, emissive: 0x004000 }));
var sphere2 = drawSphere( 100, 150, new THREE.MeshPhongMaterial({ map: texture, bumpMap: textureBump, color: 0x00ff00, specular: 0xff2200, shininess: 3 }));

//.. і додати їх до групи
group.add(sphere);
group.add(sphere2);
scene.add(group);
 // BoxHelper
boxHelper = new THREE.BoxHelper( group, 0x999999 ); (сірий).
scene.add(boxHelper);
```

У прикладі ми об’єднали дві сфери в єдиний груповий об’єкт (рисунок 4.3).

4.4 CameraHelper

Він рисує конкретний об’єкт Object3D (виглядає як піраміда) з лінійною геометрією, яка допомагає візуалізувати те, що вказана камера містить у своїй піраміді.

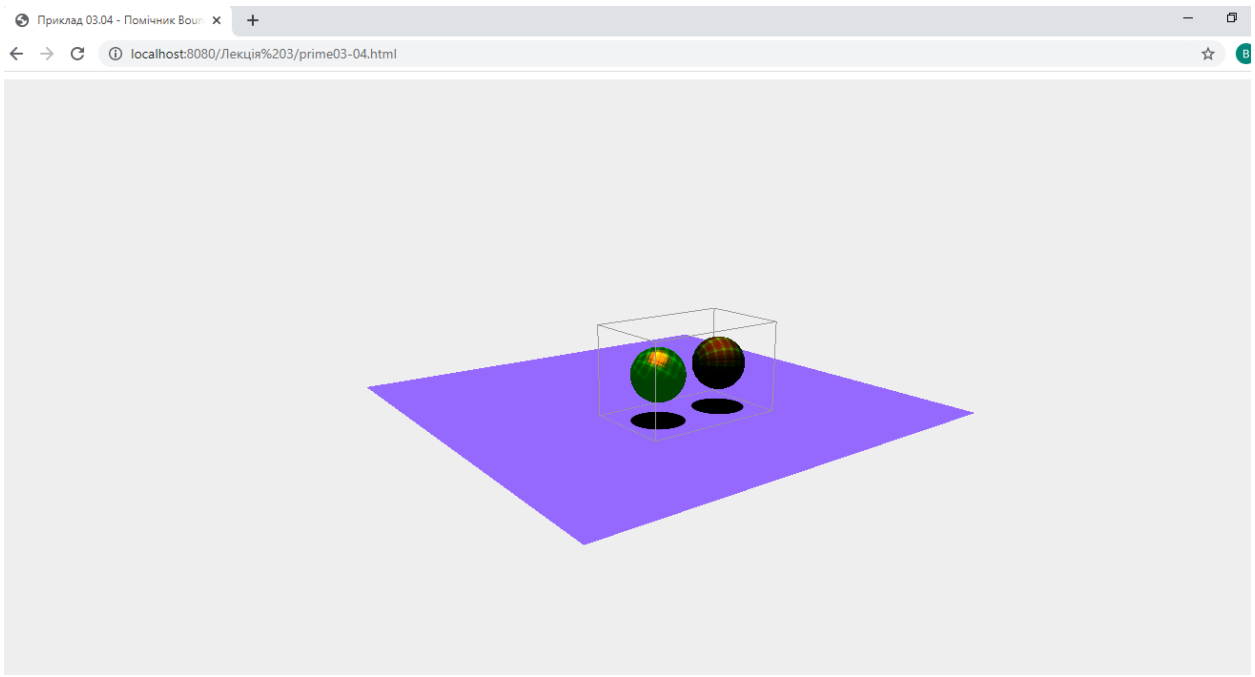


Рисунок 4.3 – Помічник BoxHelper

Конструктор
CameraHelper (camera: Camera). Тут:
Camera – Камера для візуалізації.
Це створить новий CameraHelper для цієї камери.
Щоб додати цього помічника, використовуйте наступний код (рисунок 4.4).

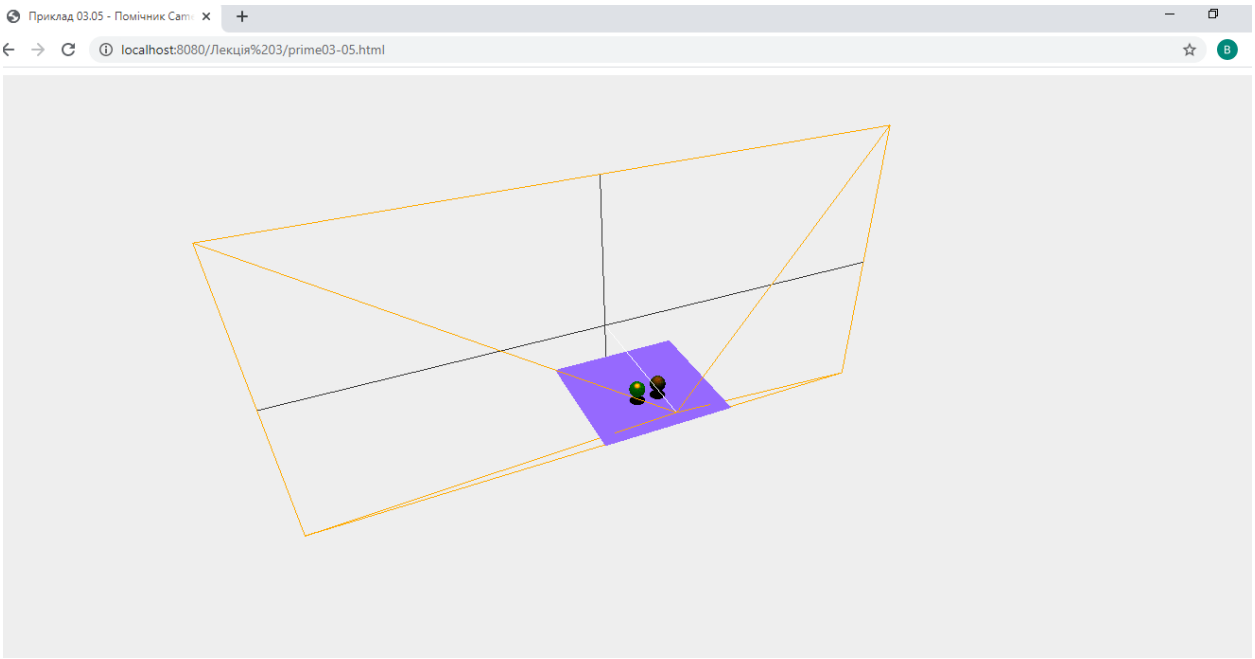


Рисунок 4.4 – Помічник CameraHelper

```
// CameraHelper
var perspectiveCamera = new THREE.PerspectiveCamera(90,
window.innerWidth / window.innerHeight, 0.01, 1500);
```

/ /90 – кут, який можна бачити навколо центру камери;

// window.innerWidth / window.innerHeight – співвідношення ширини до висоти екрана

// 0.01 – мінімальна відстань від камери, яка потрапляє у рендеринг.

// 1500 – максимальна відстань від камери, яка потрапляє у рендеринг.

```
var cameraHelper = new THREE.CameraHelper(perspectiveCamera);
scene.add(cameraHelper);
```

4.5 DirectionalLightHelper

Він рисує лінійний об’єкт, щоб показати напрямок спрямованого світла. Ми поступово переміщатимемо наше джерело світла, щоб ми могли бачити цього помічника.

Конструктор

DirectionalLightHelper(light: DirectionalLight, size: Number, color: Hex). Тут:

light – світло, яке потрібно візуалізувати;

size – (необов’язково) розмір помічника. За замовчуванням 1;

color – (необов’язково), якщо це не той набір, помічник набуде кольору світла.

Щоб додати цього помічника, використовуйте наступний код (рисунок 4.5):

```
// підготувати часи
var clock = New THREE.Clock();
// DirectionalLightHelper
dlightHelper = New THREE.DirectionalLightHelper(dLight, 50); // 50 is helper size
scene.add(dlightHelper);
function renderScene() {
requestAnimationFrame(renderScene);
controls.update(clock.getDelta());
dlightHelper.update();
// плавно рухаємо dLight
var timer = Date.now() * 0.000025;
dLight.position.x = Math.sin(timer * 5) * 300;
dLight.position.z = Math.cos(timer * 5) * 300;
renderer.render (scene, camera);
}
```

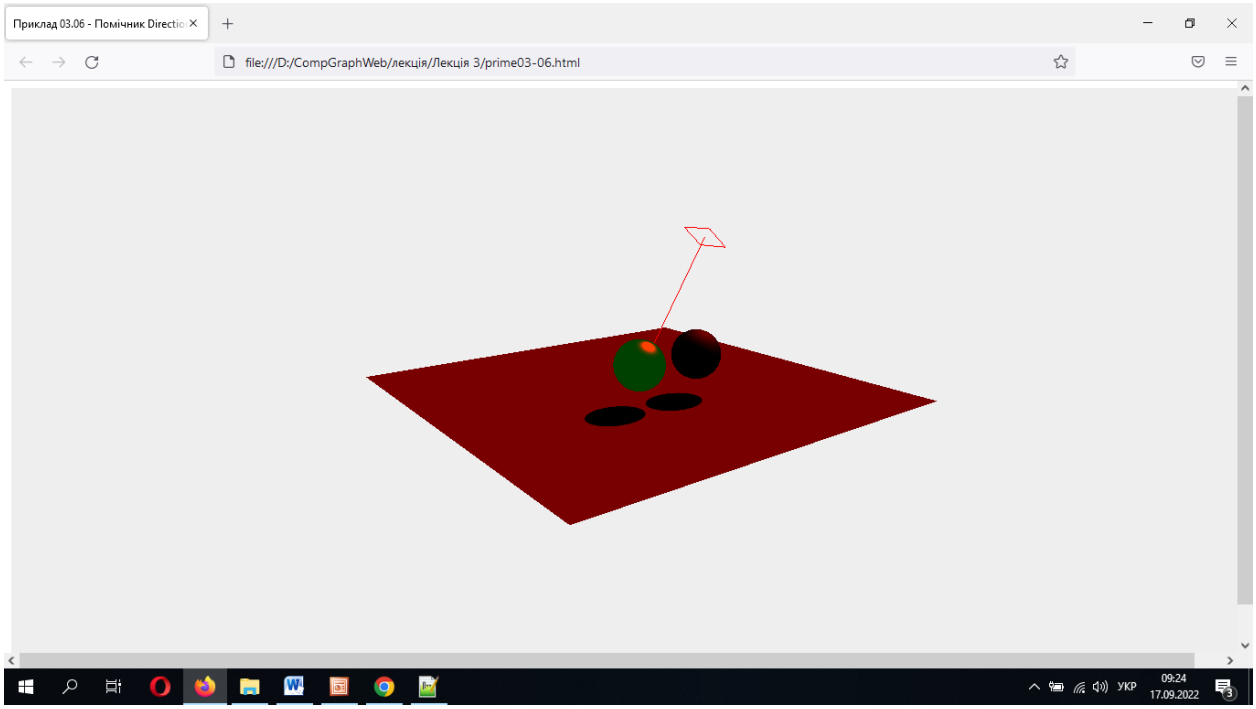


Рисунок 4.5 – Помічник DirectionalLightHelper

4.6 GridHelper

Цей помічник рисує двовимірну сітку ліній.

Конструктор

GridHelper(size: number, divisions: Number, colorCenterLine: Color, colorGrid: Color). Тут:

size – розмір сітки. Типово 10;

divisions – кількість поділів у сітці. Типово 10;

colorCenterLine – колір центральної лінії. Це може бути Color, шістнадцяткове значення та ім'я CSS-Color. Типово 0x444444 (темно сірий);

colorGrid – колір ліній сітки. Це може бути Color, шістнадцяткове значення та ім'я CSS-Color. Типово 0x888888 (світло сірий).

Створює новий GridHelper розміру size і ділиться на сегменти поділу з кожного боку. Кольори не є обов'язковими.

Щоб додати цього помічника, використовуйте наступний код (рисунок 4.6):

```
// GridHelper
var gridHelper = New THREE.GridHelper(500, 40); // 500 is grid size, 20 is grid step
gridHelper.position = New THREE.Vector3(0, 0, 0);
gridHelper.rotation = New THREE.Euler(0, 0, 0);
```

Euler – Клас, що представляє кути Ейлера.

// Кути Ейлера описують перетворення обертання шляхом обертання об’єкта за його різними осями в заданій кількості кожної осі і в певному порядку осей.

scene.add(gridHelper);

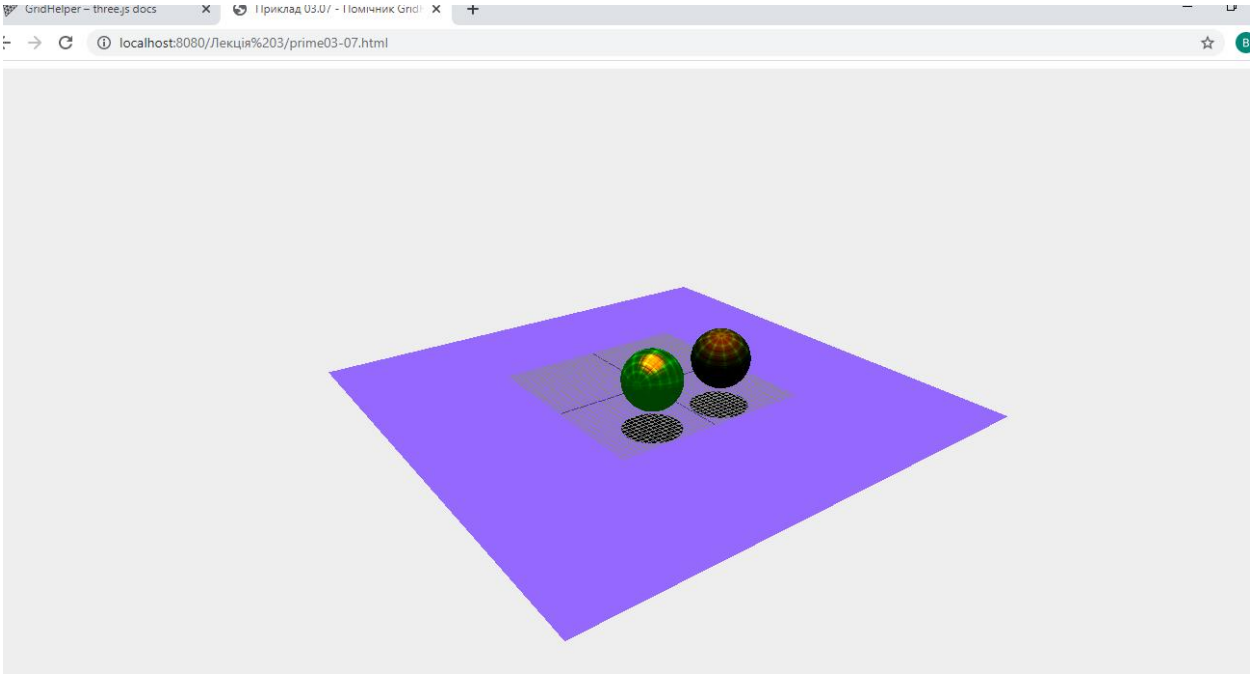


Рисунок 4.6 – Помічник GridHelper

4.7 HemisphereLightHelper

HemisphereLightHelper – призначений для джерела світла HemisphereLight – джерело світла, розташоване прямо над сценою. Це світло не можна використовувати для відкидання тіней.

Конструктор

HemisphereLightHelper(light: HemisphereLight, sphereSize: Number, color: Hex). Тут:

light – світло, яке візуалізується;

size – розмір сітки, яка використовується для візуалізації світла;

color – (необов’язково), якщо це не той набір, помічник набуде кольору світла.

// hemisphere light Джерело світла, розташоване прямо над сценою

Конструктор

HemisphereLight(skyColor: Integer, groundColor: Integer, intensity: Float). Тут:

skyColor – (необов’язково) шістнадцятковий колір піднебіння. За промовчанням 0xffffff. (білий);

groundColor – (необов’язково) шістнадцятковий колір землі. За промовчанням 0xffffff (білий);

intensity – (необов’язково) числове значення сили/інтенсивності світла. За замовчуванням 1.

Щоб додати цього помічника, використовуйте наступний код (рисунок 4.7):

```
var hemiLight = new THREE.HemisphereLight(0x0000ff, 0x00ff00, 0.4);
//синій, зелений
hemiLight.color.setHSL(0.6, 1, 0.6);
// h – значення відтінку від 0,0 до 1,0
// s – значення насиченості від 0,0 до 1,0
// l – значення яскравості від 0,0 до 1,0
hemiLight.groundColor.setHSL(0.095, 1, 0.75);
// Основний колір джерела світла, переданий у конструкторі.
hemiLight.position.set(-200, 400, -200);
scene.add(hemiLight);
//HemisphereLightHelper
var hlightHelper = New THREE.HemisphereLightHelper(hemiLight, 50);
scene.add(hlightHelper);
```

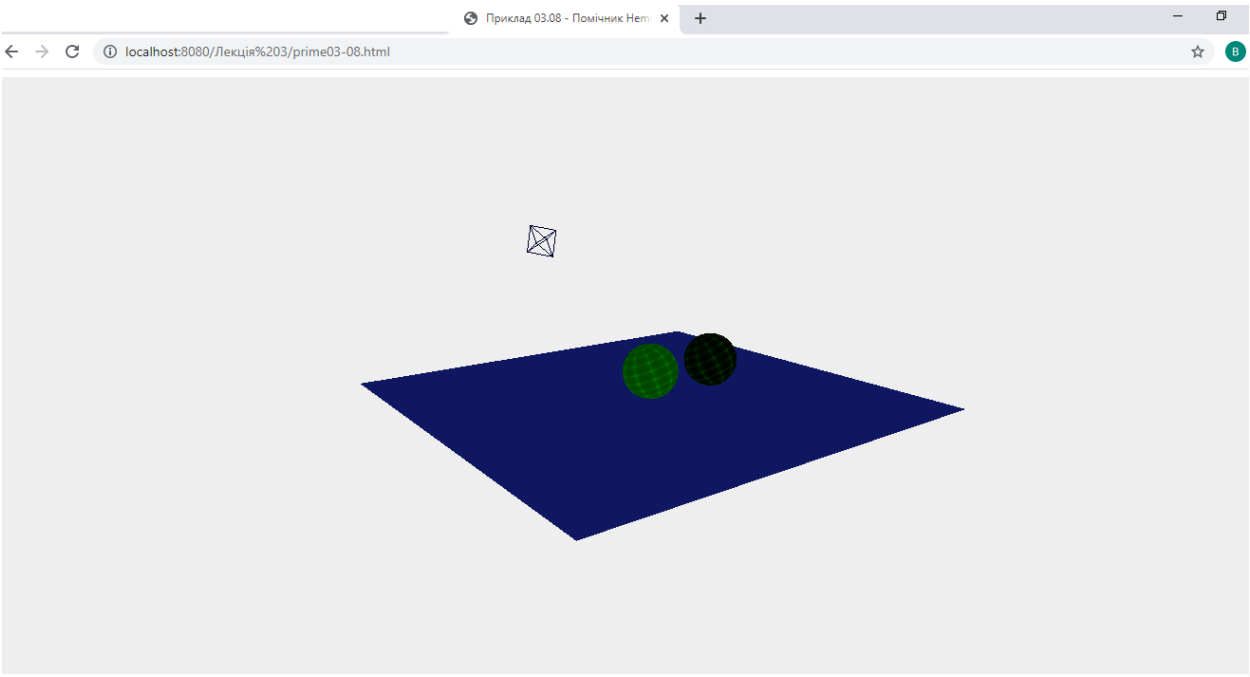


Рисунок 4.7 – ПомічникHemisphereLightHelper

4.8 PointLightHelper

PointLightHelper призначено для джерела світла PointLight. PointLight – світло, яке випромінюється з однієї точки у всіх напрямках. Зазвичай це

використовується для імітації світла, яке випромінюється лампочкою. Це світло може відкидати тіні.

Конструктор

PointLightHelper(light: PointLight, sphereSize: Float, color: Hex). Тут:

light – світло, яке потрібно візуалізувати;

sphereSize – (необов'язково) розмір допоміжної сфери. За замовчуванням 1;

color – (необов'язково), якщо це не той набір, помічник набуде кольору світла.

Щоб додати цього помічника, використовуйте наступний код (рисунок 4.8):

```
// point light – світло, яке випромінюється з однієї точки
// у всіх напрямках. Зазвичай це використовується для імітації
// світла, випромінюваного лампочкою.
var pointLight = New THREE.PointLight(0xffff00, 1.0);
pointLight.position.set(300,300,300);
pointLight.castShadow = true;
pointLight.shadowMapWidth = pointLight.shadowMapHeight = 1000;
var d = 500;
pointLight.shadow.camera.left = – d;
pointLight.shadow.camera.right = d;
pointLight.shadow.camera.top = d;
pointLight.shadow.camera.bottom = – d;
pointLight.shadow.camera.far = 1000;
pointLight.shadow.bias = – 0.0001;
scene.add(pointLight);
// PointLightHelper
var pointLightHelper = New THREE.PointLightHelper(pointLight, 50); // 50 is sphere size
scene.add(pointLightHelper);
```

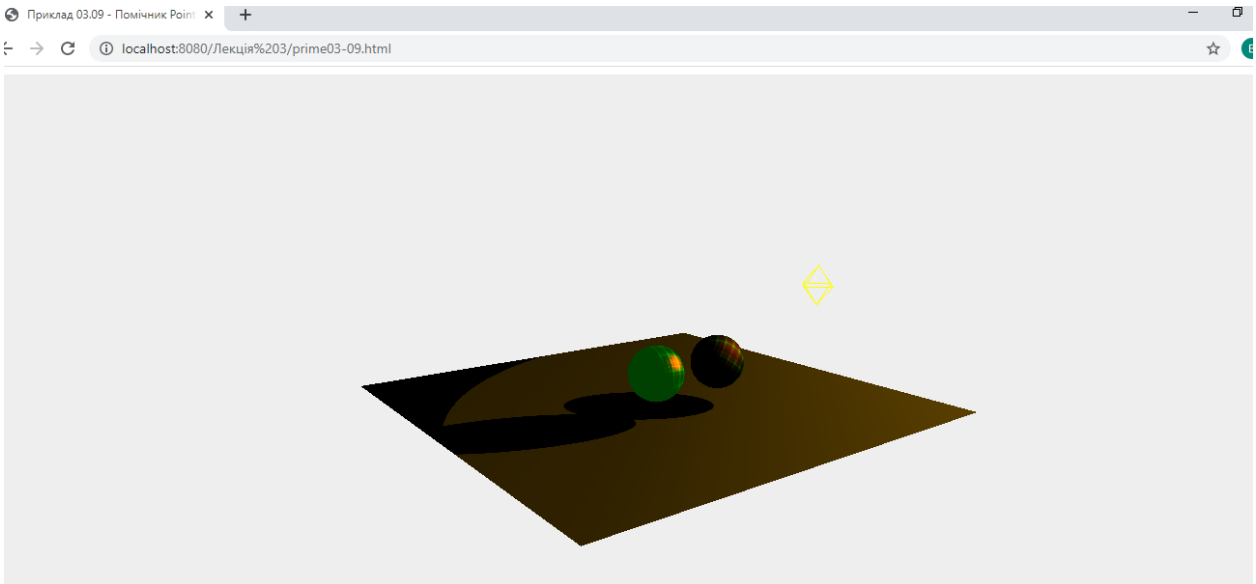


Рисунок 4.8 – ПомічникPointLightHelper

4.9 SpotLightHelper

SpotLightHelper призначений для об’єкта SpotLight. SpotLight – це світло випромінюється з однієї точки в одному напрямку, вздовж конуса, розмір якого збільшується в міру віддалення джерела світла. Це світло може відкидати тіні.

Конструктор

SpotLightHelper(light: SpotLight, color: Hex). Тут:

light – Точне світло, яке потрібно візуалізувати;

color – (необов’язково), якщо це не той набір, помічник набуде кольору світла.

Щоб додати цього помічника, використовуйте наступний код (рисунок 4.9):

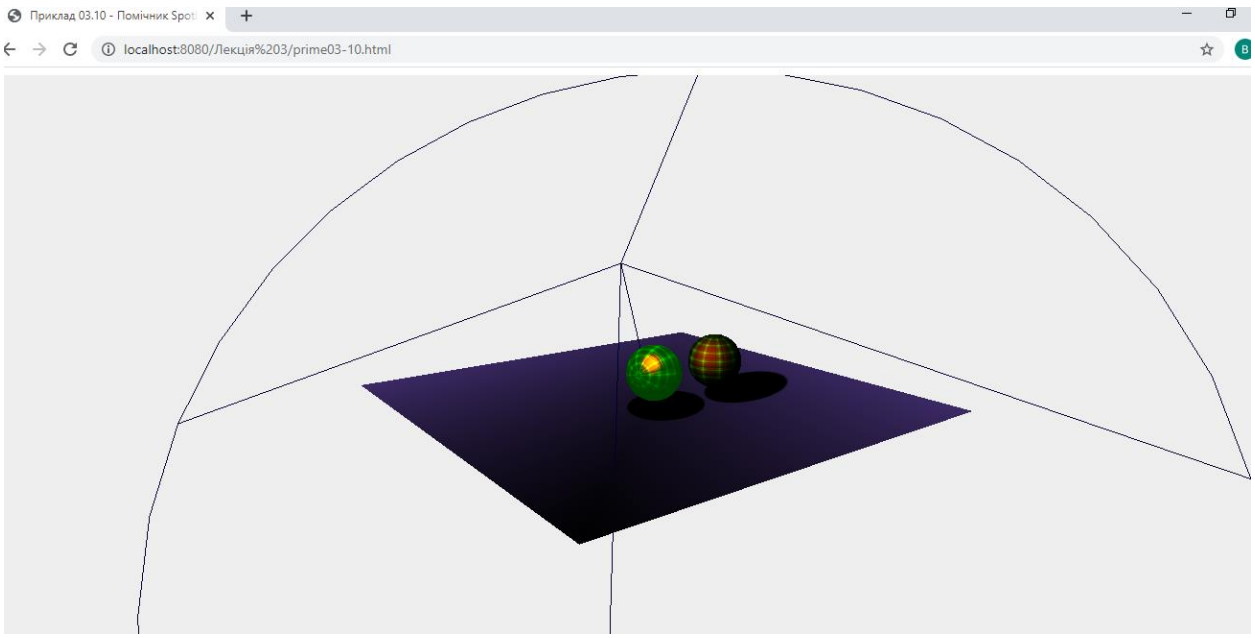


Рисунок 4.9 – Помічник SpotLightHelper

```
// spot light – світло випромінюється з однієї точки в одному напрямку,
// Вздовж конуса, розмір якого збільшується в міру віддалення від джерела світла.
var spotLight = New THREE.SpotLight(0xffffff);
spotLight.position.set(-300,400,300);
spotLight.castShadow = true;
spotLight.shadowCameraFov = 60;
scene.add(spotLight);
// SpotLightHelper
var spotLightHelper = New THREE.SpotLightHelper(spotLight, 50); // 50 is sphere size
scene.add(spotLightHelper);
```

Завдання

На підставі 1 лабораторної роботи (вибраних Вами власних геометричних фігур):

– використати помічники ArrowHelper (додаток П), AxisHelper, BoxHelper, CameraHelper, DirectionalLightHelper, HemisphereLightHelper, PointLightHelper, SpotLightHelper;
– надіслати звіт та архів усіх файлів (включаючи папку з бібліотеками js…

Рекомендована література

1. Dirksen J. Learn Three.js : Program 3D Animations and Visualizations for the Web with JavaScript and WebGl. Packt Publishing, Limited, 2023. 554 p.
2. Фреймворк three.js : офіційний сайт. URL : http://threejs.org (дата звернення : 18.01.2024).

Контрольні запитання

1. Що таке помічники в бібліотеці three.js і яку роль вони відіграють у розробці тривимірних сцен?
2. Які основні типи помічників підтримуються в three.js і для чого вони використовуються?
3. Як додати вісі координат до сцени за допомогою AxisHelper в three.js?
4. Як додати сітку (grid) до сцени за допомогою GridHelper в three.js?
5. Як додати відстань (distance) або вектор (vector) за допомогою помічника DistanceHelper або ArrowHelper в three.js?
6. Як створити камеру, яка слідкує за об’єктом, використовуючи CameraHelper в three.js?
7. Як додати обмежуючий паралелепіпед (bounding box) до об’єкта за допомогою BoxHelper в three.js?
8. Як відображати обмежуючий паралелепіпед (bounding box) об’єкта з певним кольором або матеріалом в three.js?
9. Як налаштувати розмір, колір або інші властивості помічників у three.js?
10. Як додати власний помічник, що відображає додаткову інформацію про сцену або об’єкти, у three.js?

5 ТІНІ. ОСВІТЛЕННЯ У THREE.JS

5.1 Тіні

Рендеринг тіней вимагає великих обчислювальних потужностей, і з цієї причини за замовчуванням тіні відключено в Three.js. Однак увімкнути їх дуже просто.

Для накладання тіней на сцені потрібно попередньо «включити» їх у рендері (додаток Р):

```
renderer.shadowMap.enabled = true;
renderer.shadowMap.type = THREE.PCFSoftShadowMap; // фільтрує карти тіней // з використанням алгоритму процентної фільтрації (PCF) з кращими м’якими
//тінями, особливо при використанні карт тіней з низькою роздільною //здатністю.
```

Далі необхідно увімкнути утворення тіней біля джерела світла

```
directionalLight.castShadow = true;:
var directionalLight = New THREE.DirectionalLight( 0xffffff );
directionalLight.position.set(-10, 100, 0);
directionalLight.castShadow = true; // Якщо встановлено значення true, це //світло відкидає тіні.
directionalLight.shadowCameraNear = 2; // Тіньова камера перспективи // біля параметра. Типово – 50 .
directionalLight.shadowCameraFar = 200; // Далекий параметр усіченої / / Піраміди камери перспективи.
// Визначення усіченої області тіньової камери,
// яка використовується для розрахунку карти тіней джерела світла.
directionalLight.shadowCameraLeft = -50;
directionalLight.shadowCameraRight = 50;
directionalLight.shadowCameraTop = 50;
directionalLight.shadowCameraBottom = -50;
directionalLight.distance = 0;
directionalLight.intensity = 0.5;
directionalLight.shadowMapHeight = 1024; // Висота текстури карти тіней у // пікселях. За замовчуванням – 512 .
directionalLight.shadowMapWidth = 1024; // Ширина текстури карти тіней у // пікселях. За умовчанням – 512.
scene.add(directionalLight);
```

Також властивість castShadow потрібно включити для тіл, від яких потрібно відкидання тіней.

```
var cubeGeometry = New THREE.CubeGeometry(4,4,4);
var cubeMaterial = new THREE.MeshBasicMaterial({color: 0xff0000});
var cube = New THREE.Mesh(cubeGeometry, cubeMaterial);
cube.position.x = -4;
cube.position.y = 10;
cube.position.z = 0;
cube.castShadow = true;
scene.add(cube);
```

Щоб об’єкти сцени сприймали тіні, потрібно включити у них властивість receiveShadow (рисунок 5.1):

```
var planeGeometry_2 = New THREE.PlaneGeometry(60,60,1,1);
var planeMaterial_2 = new THREE.MeshPhongMaterial( { color: 0xff00ff, side: THREE.DoubleSide });
var plane2 = New THREE.Mesh(planeGeometry_2,planeMaterial_2);
plane2.rotation.x= Math.PI/2;;
plane2.position.y = 0;
plane2.position.z = 0;
plane2.receiveShadow = true;
scene.add(plane2);
```

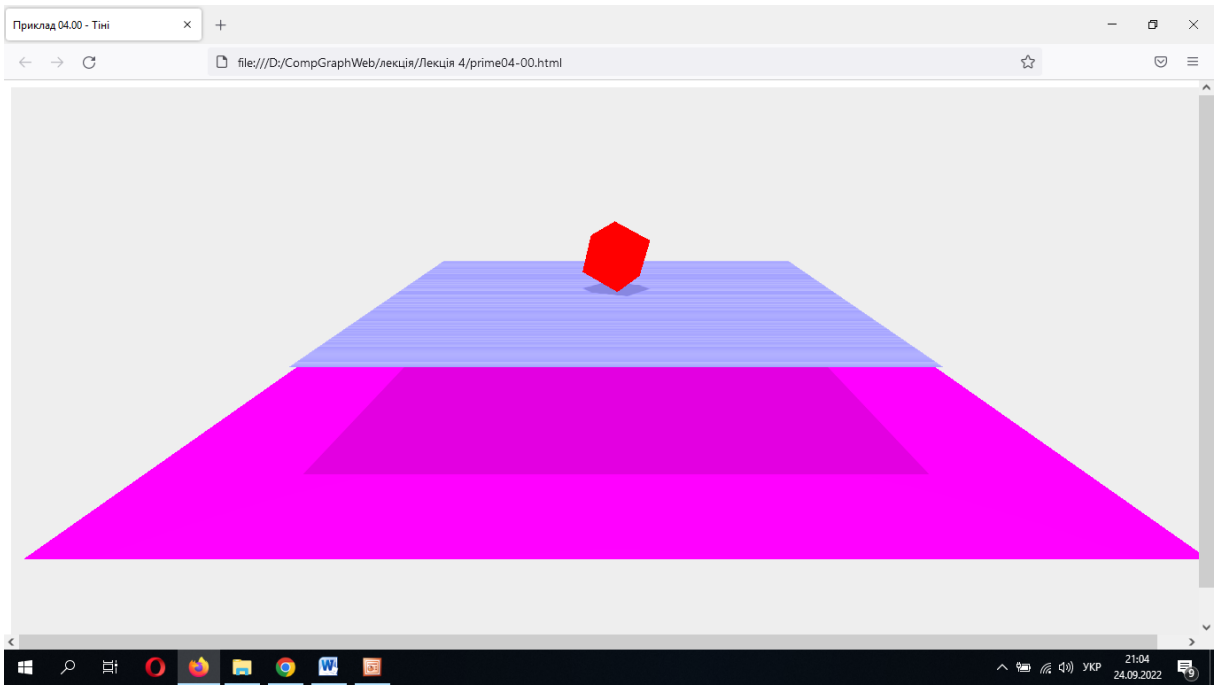


Рисунок 5.1 – Тіні

5.2 AmbientLight

AmbientLight – це освітлення, яке освітлює всю сцену. Воно не має напряму і освітлює кожен об’єкт сцени однаково, незалежно від розташування об’єкта. Відповідно, у цього світла немає позиції на осі координат. Це світло не може використовуватися для створення тіней.

Конструктор

AmbientLight (color, intensity). Тут:

color – число RGB компонентів кольору. Значення за замовчуванням дорівнює 0xffffff (білий).

Додатковий, необов’язковий аргумент:

intensity – чисельне значення сили/інтенсивності світла. Значення за замовчуванням дорівнює 1.

Конструктор створює навколишнє (загальне) освітлення із заданими кольором та інтенсивністю.

Ми не використовуємо AmbientLight як єдине джерело світла в сцені. Ми будемо використовувати його з іншими джерелами світла, такими як SpotLight або DirectionalLight, щоб пом’якшити тіні або додати колір.

У цьому прикладі (додаток С) ми отримали простий інтерфейс користувача за допомогою бібліотеки **DAT.GUI**, який можна використовувати для зміни джерела світла AmbientLight, яке додається до сцени (рисунок 5.2).

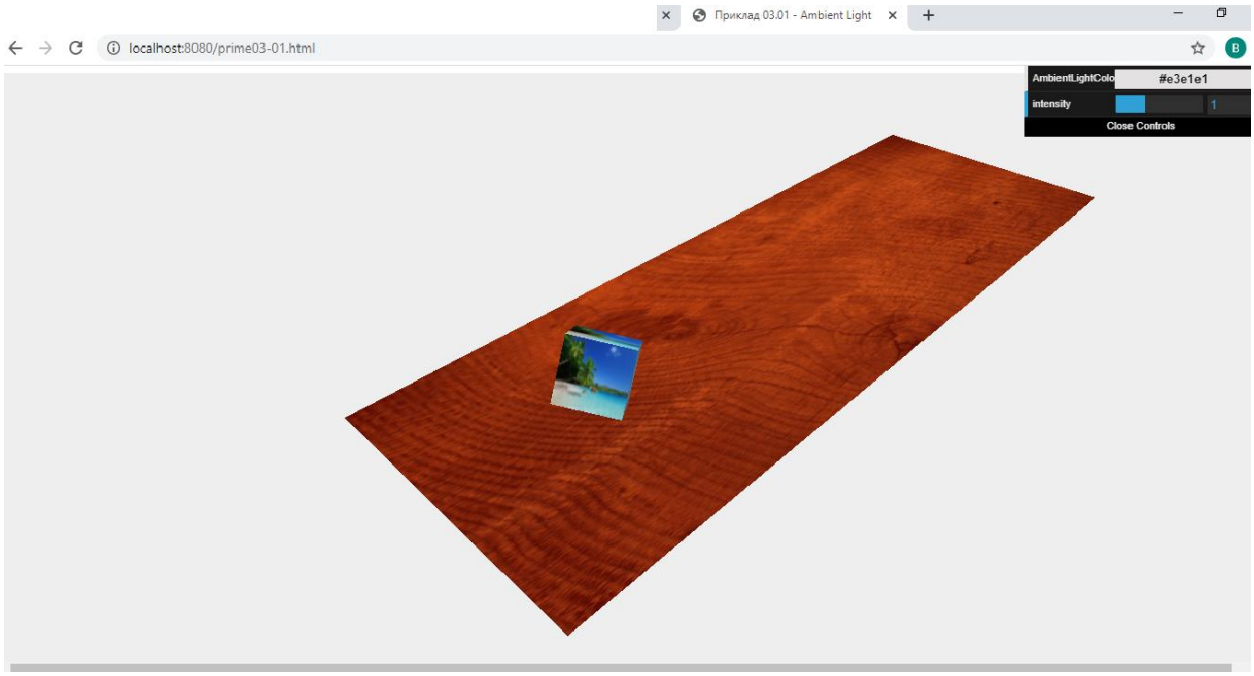


Рисунок 5.2 – AmbientLight

Якщо ми хочемо змінити цей колір на помітніший зелений колір, ми можемо зробити це в меню у правому верхньому кутку; об’єкти тоді будуть мати світло-зелене світло над ними. Це показано на рисунку 5.3.

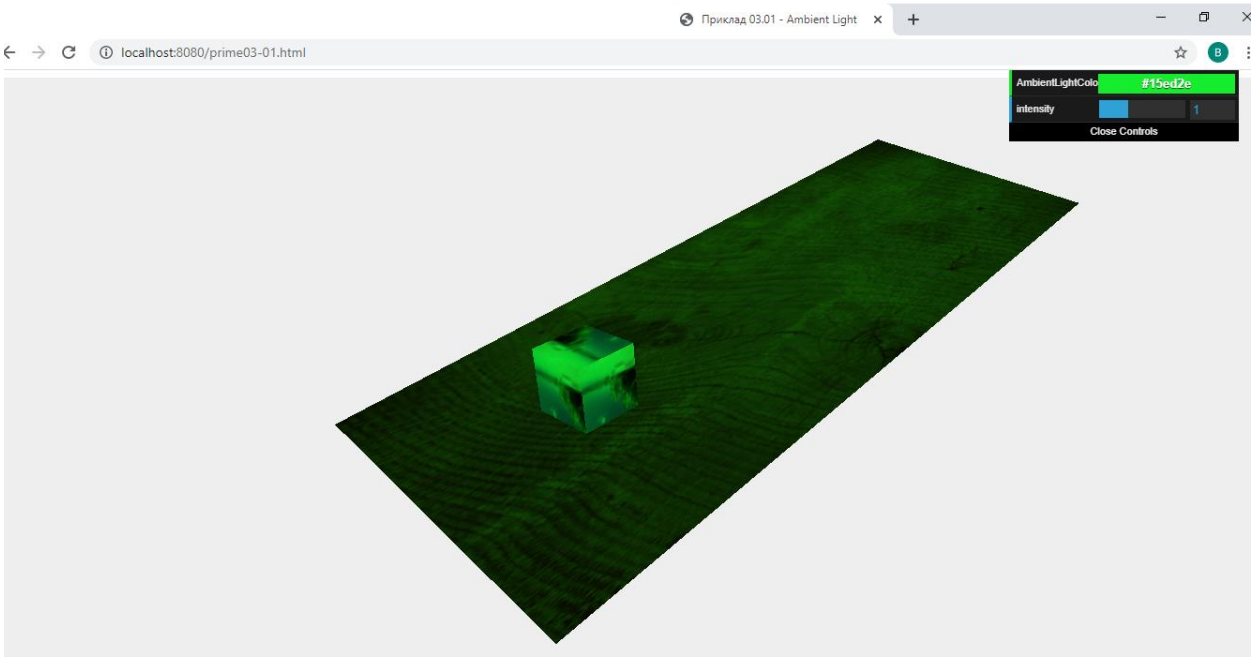


Рисунок 5.3 – Світло-зелене світло

Як показано на скріншоті, зелений колір застосовується до всіх об’єктів та висвітлює всю сцену зеленим свіченням.

Розглянемо, як ми можемо створити і використовувати джерело AmbientLight. Наступна пара рядків коду покаже нам, як створити джерело AmbientLight, а також, як підключити його до меню керування dat.GUI:

```
var ambiColor = "#e3e1e1";
var ambiIntensity = 1;
var ambientLight = New THREE.AmbientLight(ambiColor,ambiIntensity);
scene.add(ambientLight);
```

Додамо наступний JavaScript об’єкт:

```
var controls = new function () {
this.rotationSpeed = 0.02;
this.AmbientLightColor = ambiColor;
this.intensity = 1;
};
```

Далі ми передаємо цей об’єкт у новий об’єкт dat.GUI і визначаємо діапазон цих трьох властивостей, як показано нижче:

```
var gui = new dat.GUI();
gui.addColor(controls, ‘AmbientLightColor’).onChange(function (e) {
ambientLight.color = New THREE.Color(e);
});
gui.add(controls, ‘intensity’, 0, 3).onChange(function (e) {
```

```
ambientLight.intensity = e;
});
```

Створення джерела AmbientLight дуже очевидно. Оскільки джерело AmbientLight не має позиції, нам потрібно лише вказати колір (у шістнадцятковому форматі), використовуючи новий new THREE.AmbientLight(ambiColor,ambiIntensity).

У цьому прикладі зв'язали колір джерела AmbientLight з елементом управління.

У коді ми використовували функцію onChange Data файлу.

5.3 PointLight

PointLight – освітлення, яке виходить з однієї точки у всіх напрямках (це джерело освітлення можна порівняти зі звичайною лампочкою).

Конструктор

PointLight(color, intensity, distance, decay). Тут:

color – шістнадцяткове значення кольору освітлення за замовчуванням 0xffffff (білий). Додатковий, необов'язковий аргумент;

intensity – чисельне значення сили/інтенсивності світла. Значення за замовчуванням дорівнює 1. Додатковий, необов'язковий аргумент;

distance – відстань від джерела світла, при якому інтенсивність світла дорівнює 0. Значення за замовчуванням дорівнює 0;

decay – значення, на яке зменшується інтенсивність світла зі збільшенням відстані від джерела.

Як і у випадку з усіма джерелами світла, ми маємо конкретний приклад, який ми можемо використовувати, щоб розібратися з PointLight. У прикладі розглядається переміщення джерела PointLight навколо простої сцени Three.js (додаток Т, рисунок 5.4).

У цьому прикладі джерело PointLight рухається сценою. Щоб було зрозуміліше, де джерело PointLight: ми переміщаємо маленьку сферу тим самим шляхом.

Коли джерело світла рухатиметься, ми побачимо куб, освітлений цим світлом.

Розглянемо, як створити джерело PointLight:

```
var pointColor = "#ccffcc"; //блакитний
var pointLight = New THREE.PointLight(pointColor);
pointLight.distance = 100;
scene.add(pointLight);
```

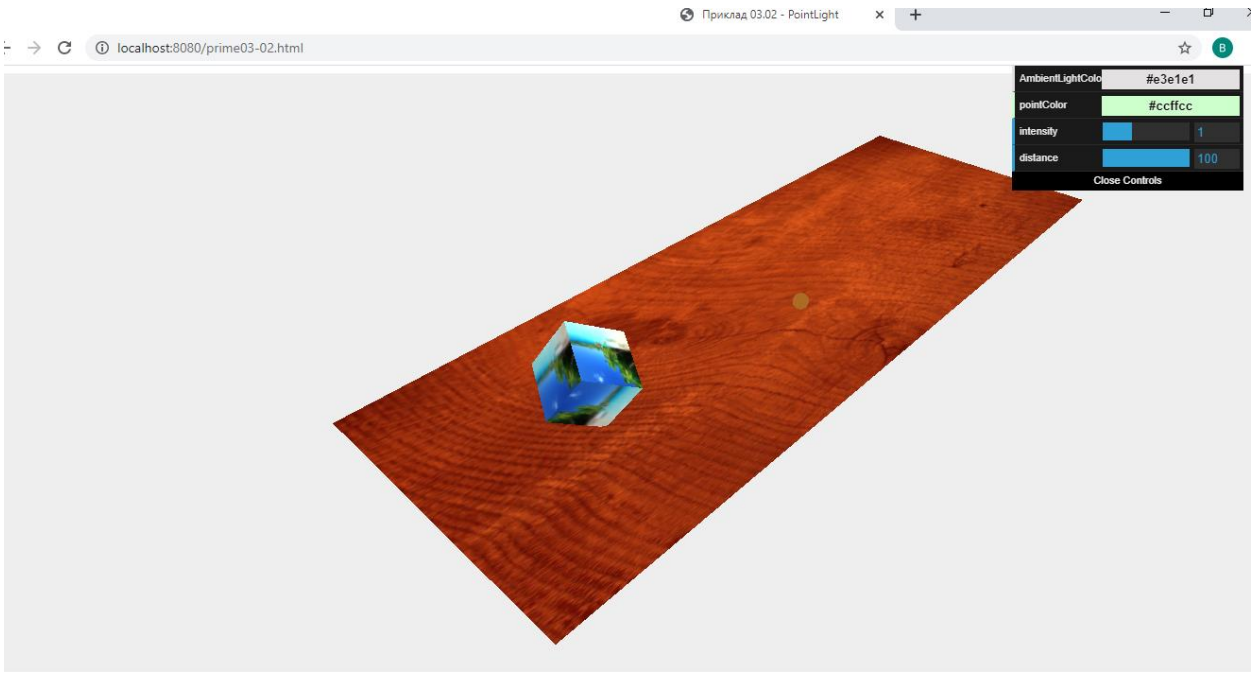


Рисунок 5.4 – PointLight

Тут нічого нового. Ми створили джерело світла певного кольору, встановили його положення і додали його до сцени. Перша властивість, яку ми розглянемо, – це інтенсивність. З цим ви можете встановити, наскільки яскраво світитиме світло. Якщо ми встановимо його на 0, ми не побачимо нічого, встановимо його на 1, і ми маємо яскравість за замовчуванням, а встановимо його на 3, і ми отримаємо світло, яке світитиме втричі яскравіше. На наступному знімку екрана (рисунок 5.5), наприклад, ми встановили властивість інтенсивності світла 3.

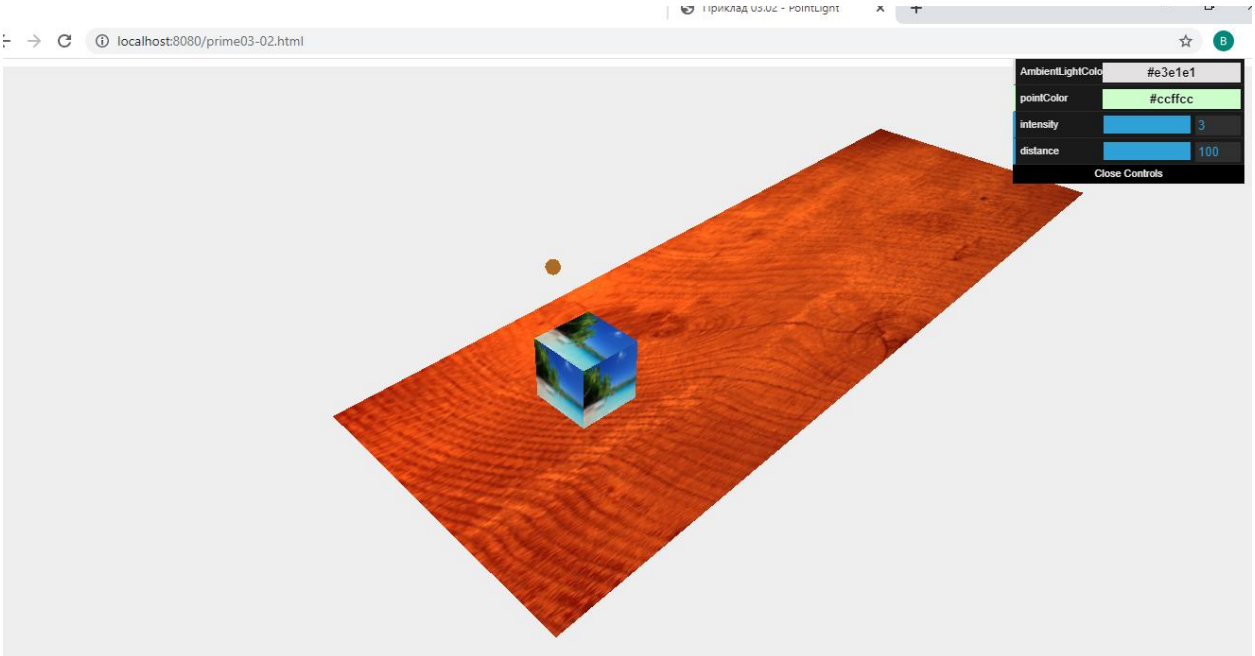


Рисунок 5.5 – Інтенсивность світла яка дорівнюється 3

Все, що вам потрібно зробити, щоб змінити властивість інтенсивності світла, це написати наступний код:

```
pointLight.intensity = 3;
```

Ми також можемо використовувати DAT.GUI, як показано в наступному фрагменті коду:

```
var controls = new function() {
this.intensity = 1; }
var gui = new dat.GUI();
gui.add(controls, ‘intensity’, 0, 3).onChange(function(e) { pointLight.intensity = e; });
```

Остання властивість джерела PointLight – дуже цікава і найкраща, пояснимо на прикладі. На наступному скріншоті (рисунок 5.6) ми побачимо ту саму сцену знову, але цього разу з дуже високою інтенсивністю (оскільки у нас дуже яскраве світло), але з невеликою відстанню.

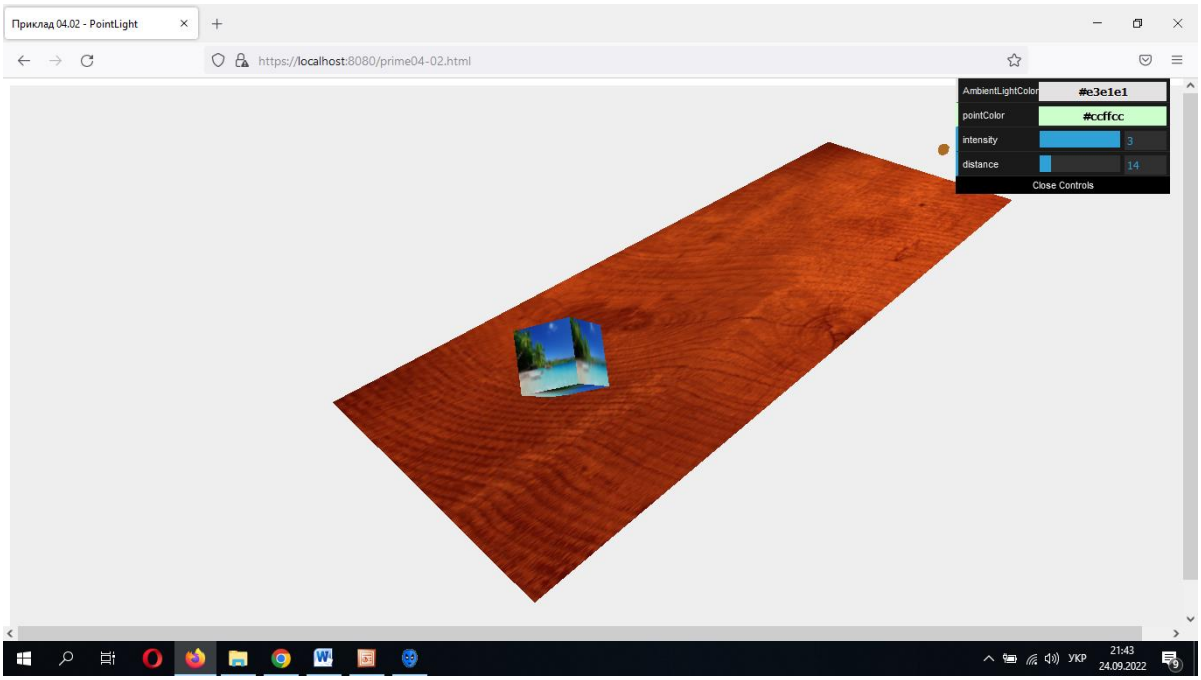


Рисунок 5.6 – Сцена з високою інтенсивністю, але з невеликою відстанню

Властивість distance джерела SpotLight визначає, як далеко буде світло від джерела. Ми можемо встановити властивість так:

```
pointLight.distance = 14;
```

У нашому прикладі яскравість (інтенсивність) світла повільно зменшуватиметься до 0 при відстань 14. Значення за замовчуванням для властивості distance дорівнює 0, яке означає, що світло не гасне на відстані.

5.4 SpotLight

SpotLight – світло з ефектом конуса, є одним із джерел світла, яке ми використовуватимемо найчастіше (особливо якщо будемо використовувати тінь). Це світло має напрямок і кут, під яким воно випромінює світло. Світло PointLight також застосовується до SpotLight. Світло SpotLight також має низку додаткових властивостей, представлених в таблиці 5.1

Таблиця 5.1 – Додаткові властивості світло SpotLight

Властивість	Опис
castShadow	Якщо встановлено значення true, це світло відкидає тіні.
shadowCameraNear	Тінь камери перспективи біля параметра. Типово – 50.
shadowCameraFar	*Далекий* параметр зрізаної піраміди камери перспективи. За замовчуванням – 5000.
shadowCameraFov	Параметр поля зору зрізаної камери перспективи. Типово – 50.
target	Визначає, куди спрямоване світло.
shadowBias	Може використовуватися для усунення позиції візуалізованої тіні.
angle	Наскільки великий промінь від джерела світла. Це вимірюється у радіанах. За замовчуванням Math.PI/3.
exponent	Це значення визначає, наскільки швидко інтенсивність світла зменшується.
тільки Shadow	Якщо встановлено значення true, це світло лише відкидатиме тінь і не буде додано до сцени світло.
shadowCameraVisible	Якщо встановлено значення true, ми можемо побачити, як і де джерело світла відкидає тінь.
shadowDarkness	За замовчуванням 0,5. Визначає, наскільки темним буде рендерінг тіні. Не може бути змінено після рендерингу сцени.
shadowMapWidth	Ширина текстури карти тіні в пікселях. За замовчуванням – 512.
shadowMapHeight	Висота текстури карти тіні в пікселях. За замовчуванням – 512 .

Приклад створення світла SpotLight (рисунок 5.7):

```
var pointColor = "#ffffff";    //білий
var spotLight = New THREE.SpotLight(pointColor);
spotLight.position.set(-40, 60, -10);
spotLight.castShadow = true;
spotLight.shadowCameraNear = 2;
spotLight.shadowCameraFar = 200;
```

```
spotLight.shadowCameraFov = 30; // поле зору
spotLight.target = plane;
spotLight.distance = 0;
spotLight.angle = 0.4;
scene.add(spotLight);
```

Не дуже відрізняється від джерела PointLight. Єдина різниця в тому, що ми встановимо для якості castShadow значення true, тому що нам потрібні тіні.

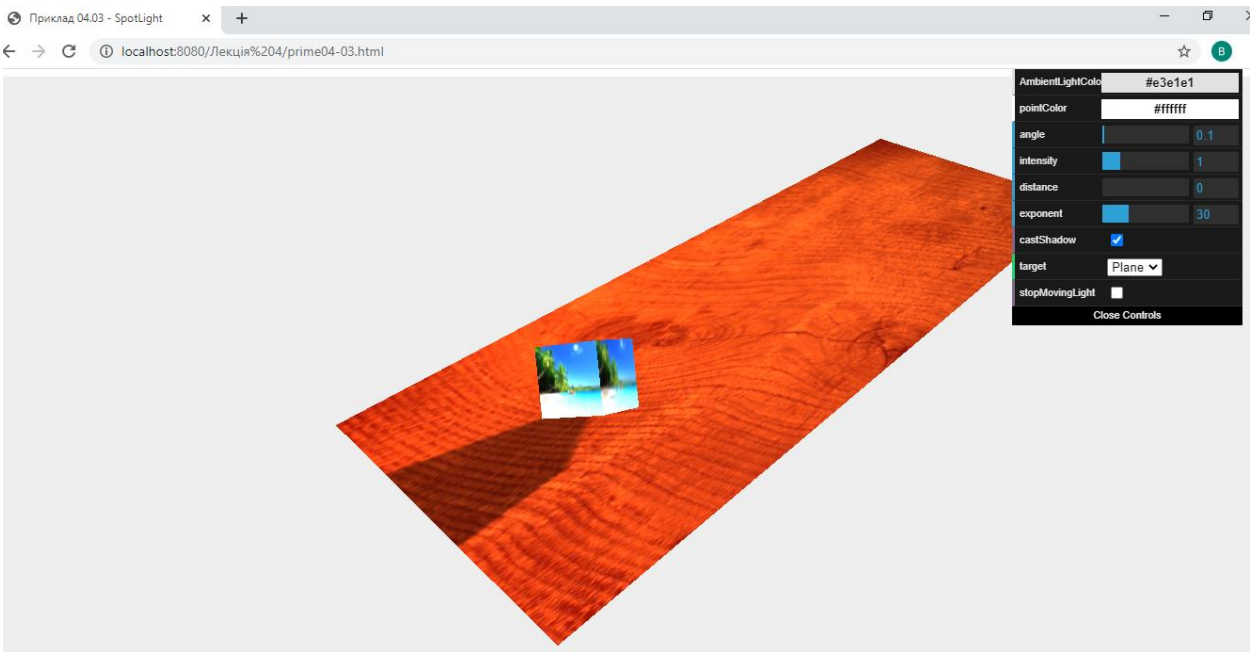


Рисунок 5.7 – SpotLight

У цьому прикладі ми можемо встановити низку властивостей для SpotLight. Одне з них target – визначає, куди спрямоване світло.

У прикладі, ми створили джерело світла та направили його на площину. Ми також можемо направити його на другий об’єкт – куб. Але якщо ми не хочемо направляти світло на конкретний об’єкт, а довільну точку простору, то ми можемо зробити це, створивши порожній екземпляр THREE.Object3D () наступним чином:

```
var target = новий THREE.Object3D();
target.position = новий THREE.Vector3(5, 0, 0);
```

І встановимо якість джерела SpotLight, як показано:

```
spotlight.target = target;
```

У таблиці 5.1 ми розглянули кілька властивостей, які можна використовувати, щоб контролювати, як світло походить від джерела SpotLight. Властивості відстані та кута визначають форму конуса.

Властивість angle визначає ширину конуса та за допомогою властивості відстані ми можемо встановити довжину конуса:

```
var coneLength = light.distance? light.distance: 10000;
var coneWidth = coneLength * Math.tan (light.angle * 0.5) * 2;
```

На наступному рисунку 5.8 показано, як кут та довжина конуса визначають ширину конуса:

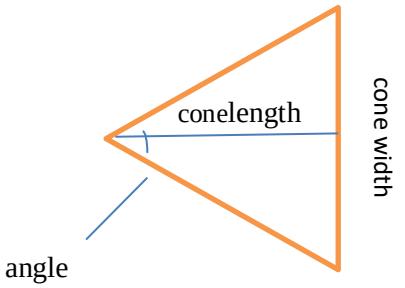


Рисунок 5.8 – Кут та довжина конуса визначають ширину конуса

Зазвичай нам не потрібно встановлювати ці значення, оскільки вони мають розумні значення за замовчуванням, але ми можемо використовувати ці властивості, наприклад для створення джерел SpotLight, які мають дуже вузький промінь або швидке зменшення інтенсивності світла. Остання властивість, яку ми розглянемо – це, щоб змінити сприйняття світла SpotLight, можна використовувати властивість експоненти. За допомогою цієї властивості ми можемо встановити, наскільки швидко зменшуватиметься інтенсивність світла від центру світлового конуса, як показано на рисунку 5.9.

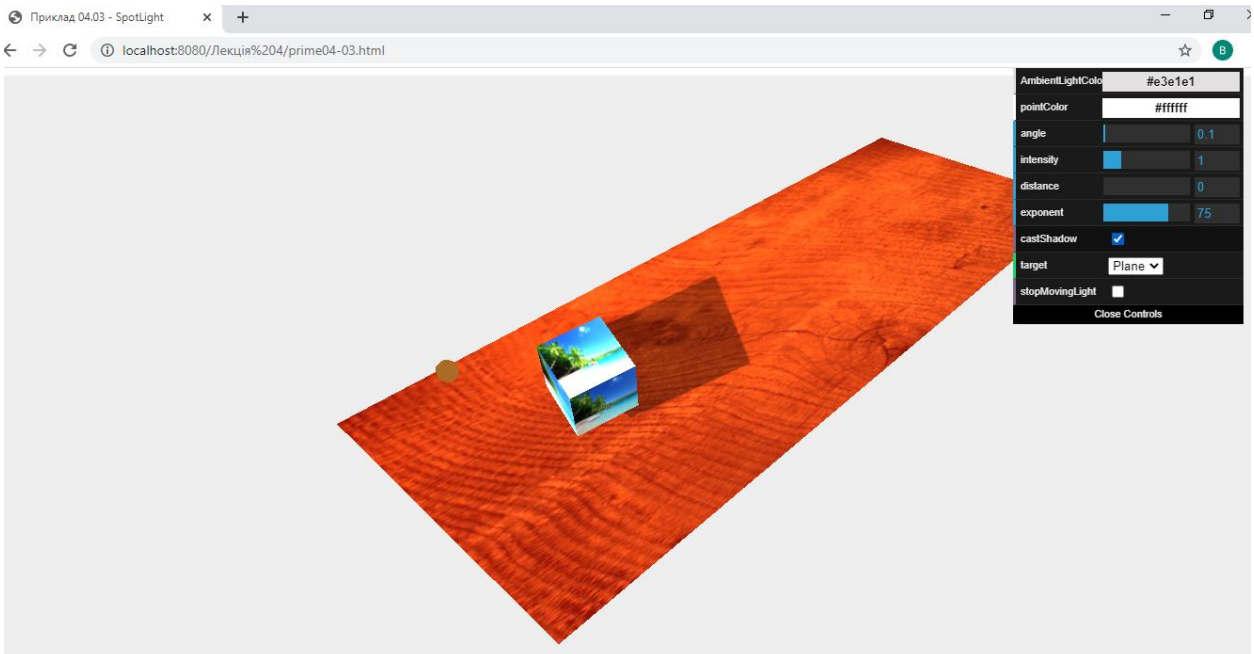


Рисунок 5.9 – Властивість експоненти

У нас дуже яскраве світло (висока інтенсивність), інтенсивність, яка швидко зменшується (високий показник) у міру віддалення від центру. Ми також могли б створити той самий ефект сфокусованого променя за рахунок використання малого значення показника ступеня та малого кута.

Перш ніж перейти до такого джерела світла, ми розглянемо тіні, тобто. властивості тіней доступні джерелу SpotLight. Ми вже розглянули раніше, що ми можемо отримати тіні, встановивши властивість castShadow джерела SpotLight як істина.

Бібліотека Three.js також дозволяє дуже детально контролювати, як рендерує тінь. Це робиться за допомогою декількох властивостей, які ми розглянули в таблиці раніше в цій лекції. За допомогою shadowCameraNear, shadowCameraFar та shadowCameraFov, ми можемо контролювати, як і де це світло буде відкидати тінь.

Це працює так само, як поле зору перспективної камери. Найпростіший спосіб побачити це в дії – встановити для властивості shadowCameraVisible значення true.

5.5 DirectionalLight

Джерело DirectionalLight можна використовувати як джерело світла, яке є дуже далеко. Все світло промені, які він випромінює, є паралельними один одному. Хорошим прикладом є сонце.

Основна відмінність між джерелом DirectionalLight і SpotLight полягає в тому, що це світло не зменшуватиметься в міру віддалення від нього.
Вся область, освітлена джерелом DirectionalLight, отримує однакову інтенсивність світла.

Щоб побачити це у дії, розглянемо приклад (додаток У), який показано на знімку екрана (рисунок 5.10).

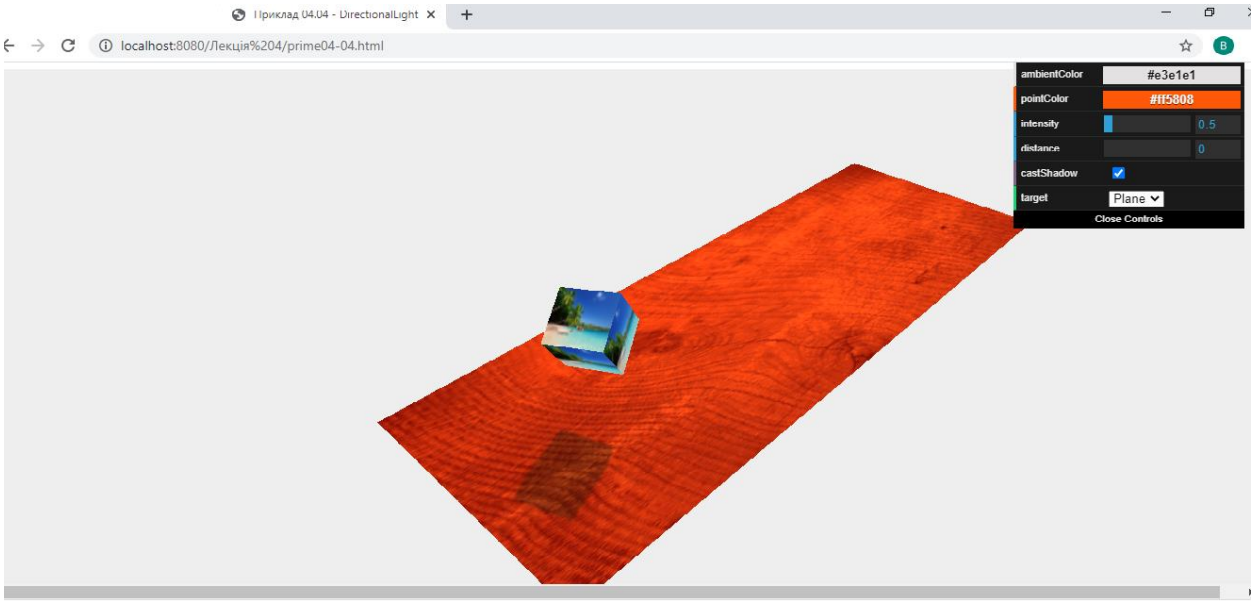


Рисунок 5.10 – DirectionalLight

Як бачите, на сцену не поширюється світловий конус. Все одержує таку ж інтенсивність світла. Як і у випадку з джерелом SpotLight, є кілька властивостей, які ми можемо встановити, контролювати інтенсивність світла та те, як він відкидає тіні. DirectionalLight має безліч властивостей, аналогічних властивостям SpotLight: target, intensity, distance, castShadow, onlyShadow, shadowCameraNear, shadowCameraFar, shadowDarkness, shadowCameraVisible, shadowMapWidth, shadowMapHeight і shadow.

Для джерела DirectionalLight: оскільки всі промені паралельні один одному, у нас замість конуса світла є куб.

Все, що потрапляє в куб, може відкидати та отримувати тіні від світла. Як і у випадку з джерелом SpotLight, чим точніше ми визначимо область навколо об’єктів, тим краще виглядатимуть наші тіні. Ми можемо задати куб, використовуючи такі властивості:

```
directionalLight.shadowCameraNear = 2;
directionalLight.shadowCameraFar = 200;
directionalLight.shadowCameraLeft = -50;
directionalLight.shadowCameraRight = 50;
directionalLight.shadowCameraTop = 50;
directionalLight.shadowCameraBottom = -50;
```

Завдання

На підставі 1 лабораторної роботи (вибраних Вами власних геометричних фігур):

- накласти тіні на сцену (додаток Р);
- використати різні види освітлення three.js: AmbientLight (додаток С), PointLight (додаток Т), SpotLight, DirectionalLight (додаток У);
- надіслати звіт та архів усіх файлів (включаючи папку з бібліотеками js, папку images…).

Рекомендована література

1. Dirksen J. Learn Three.js : Program 3D Animations and Visualizations for the Web with JavaScript and WebGl. Packt Publishing, Limited, 2023. 554 p.

2. Фреймворк three.js : офіційний сайт. URL : http://threejs.org (дата звернення : 18.01.2024).

Контрольні запитання

1. Що таке тіні в бібліотеці three.js і чому вони важливі для створення реалістичних тривимірних сцен?
2. Як включити підтримку тіней у three.js і налаштувати їх глобальні параметри?
3. Як додати тіні до об’єктів на сцені в three.js?

4. Які типи тіней підтримуються в three.js, і як вони відрізняються один від одного?
5. Як налаштувати параметри тіней, такі як якість, м’якість, розмір картки тіні, відстань рендерингу тощо?
6. Як додати тіні від джерела світла (наприклад, сонця або лампи) в three.js?
7. Як додати тіні від камери на сцені в three.js, наприклад, для проектування тіней на площину?
8. Як створити власні картки тіней (shadow maps) для досягнення певних ефектів у three.js?
9. Як впливати на продуктивність рендерингу сцени в залежності від використання тіней в three.js?
10. Що таке освітлення в бібліотеці three.js і чому воно важливе для створення реалістичних тривимірних сцен?
11. Які типи джерел світла підтримуються в three.js і як вони відрізняються один від одного?
12. Як додати джерело світла на сцену в three.js і налаштувати його параметри, такі як інтенсивність, колір та напрямок?
13. Як використовувати різні типи джерел світла, такі як AmbientLight, DirectionalLight, PointLight, SpotLight, HemisphereLight та інші, в three.js?
14. Як створити реалістичні ефекти освітлення, такі як тіні, рефлексії та відблиски, за допомогою джерел світла в three.js?
15. Як додати різні матеріали та текстури на об’єкти і налаштувати їх взаємодію з освітленням в three.js?

6 ВИКОРИСТАННЯ ВІДЕО

6.1 Використання відео як текстури

Сучасні браузери чудово підтримують відтворення відео без будь-яких плагінів. З Three.js ми навіть можемо використовувати відео як вхідні дані для наших текстур. У цьому прикладі ми розглянемо послідовність дій, які необхідно зробити для виведення відео на куб (додаток Ф, рисунок 6.1).

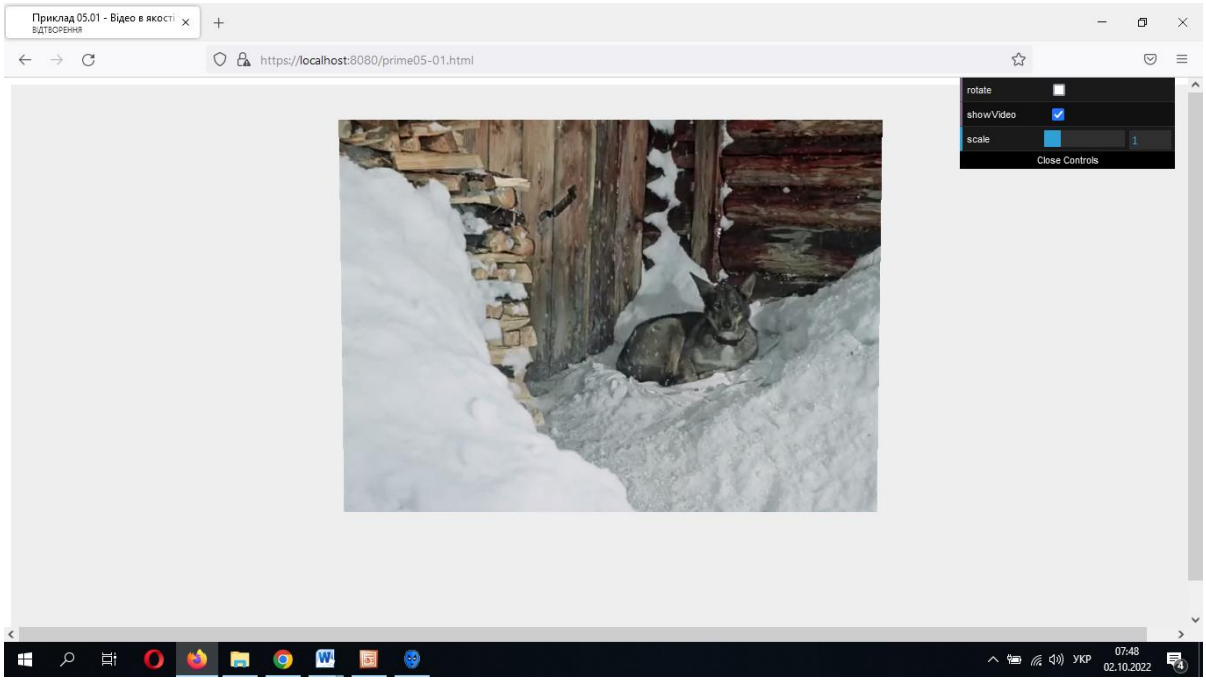


Рисунок 6.1 – Виведення відео на куб

Коли ми запустимо цей приклад, побачимо, що відео відтворюється і продовжує оновлюватися, навіть коли куб обертається.

Щоб досягти цього ефекту, нам необхідно визначити елемент відео HTML як джерело нашої текстури. Для цього виконаємо такі дії:

а) перше, що нам потрібно, це спосіб відтворити відео, для цього ми додаємо наступний HTML-код в основний елемент сторінки:

```
<video id="video"
    style="display:none; position: absolute; left: 15px; top: 75px;"
    src="./movies/NewVideo.mp4">
</video>
```

За допомогою цього фрагмента HTML ми завантажимо відео та відтворимо його після завантаження. Оскільки ми встановили display: none, цей елемент відео не відображається на сторінці. Встановивши властивість loop, завантажимо відео та відтворимо його в циклі після завантаження.

```
<video id="video" loop
    style="display:none; position: absolute; left: 15px; top: 75px;"
    src="./movies/NewVideo.mp4">
</video>
```

б) тепер, коли відтворюється відео, ми можемо отримати посилання на цей елемент і використовувати його для створення текстури:

```
var video = document.getElementById('video');
//відеоелемент для використання
// як текстури.
videoTexture = New THREE.Texture(video);
// minFilter – як вибирається текстура, коли тексел (англ.Texel, скорочення від Texture
// element«Текстурний елемент») – мінімальна одиниця текстури тривимірного об'єкта,
// Піксел текстури) покриває менше одного пікселя.
// LinearFilter -Значення за замовчуванням – THREE.LinearFilter, яке бере чотири
// Найближчих текселя і виконує білінійну інтерполяцію між ними.

videoTexture.minFilter = THREE.LinearFilter;
videoTexture.magFilter = THREE.LinearFilter;
videoTexture.format = THREE.RGBFormat; // Формат пікселів

videoTexture.generateMipmaps = false; //Чи створювати mipmaps – картки (якщо можливо) для
// текстури. Істина за замовчуванням. Встановіть false, якщо ви створюєте mipmaps – карти вручну.
```

Використання властивостей minFilter, magFilter, format і generateMipmaps тут забезпечують кращу якість і продуктивність при використанні відео як текстури.

в) на даний момент у нас є текстура, яку ми можемо використовувати як будь-яку іншу текстуру:

```
materialArray.push(new THREE.MeshBasicMaterial({map: videoTexture}));
```

Тут ми встановлюємо властивість карти матеріалу для відеотекстури. Отже, будь-який створений нами об'єкт THREE.Mesh, який використовує цей матеріал, показуватиме відео.

г) Three.js зазвичай кешує текстури, оскільки вони зазвичай не змінюються так часто. У цьому прикладі текстура постійно

змінюється. Щоб повідомити Three.js, що текстура змінилася, нам необхідно додати до циклу рендерингу наступне:

```
function renderScene()
{
videoTexture.needsUpdate = true;
requestAnimationFrame(renderScene);
renderer.render (scene, camera);
}
```

WebGL, який використовується Three.js для рендерингу сцен, має вбудовану підтримку для використання відео. Three.js просто передає елемент відео на WebGL і не вимагає попередньої обробки. У WebGL поточне зображення, показане відео перетворюється на текстуру. Щоразу, коли ми встановлюємо для videoTexture.needsUpdate значення true, текстура оновлюється в WebGL.

При роботі з елементом відео пам’ятайте, що різні браузери мають різну підтримку відеоформатів.

6.2 Додавання 3D тексту

Чудова особливість бібліотеки Three.js у тому, що вона дає можливість використовувати 3D-текст. Ми можемо використовувати будь-який текст, навіть за допомогою шрифтів, як тривимірний об’єкт у нашій сцені.

Для створення тривимірного тексту призначено клас **TextGeometry**.

Конструктор

TextGeometry(text, parameters). Тут:

text– текст, який потрібно показати.

parameters включають:

- size: розмір тексту (за замовчуванням 100);
- height: Товщина для видавлювання тексту (за замовчуванням 50);
- curveSegments: кількість точок (сегментів) кривої під час рисування літери (за замовчуванням 12);
- font: назва шрифту;
- weight: тип шрифту ("normal", "bold");
- style: стиль шрифту ("normal", "italics");

Далі можна включити фаску (тобто додати «скоси»):

- bevelEnabled: включення фаски (за замовчуванням – false);
- bevelThickness: глибина фаски (за замовчуванням 10);
- bevelSize: ширина фаски (за замовчуванням 8);
- bevelOffset – на відстані від контуру тексту починається фаска (за замовчуванням 0);
- bevelSegments- Кількість сегментів фаски (за замовчуванням 3).

Тривимірний текст виходить із плоского «екструзією»

(«видавлюванням») на величину height (товщина для видавлювання тексту). Можна використовувати різні матеріали для передньої (задньої) частини тексту та отриманої екструзії бічної частини.

Потрібно створити масив матеріалів та вказати:

– material:індекс матеріалу для переднього та заднього планів;
– extrudeMaterial: індекс матеріалу для плану екструзії та плану фаски.

При цьому обов’язково потрібно завантажити шрифт. На сайті https://ukrfonts.com/info/index.php?v=19&id=5575 представлені безкоштовні українські шрифти з розширенням ttf.

Щоб отримати файл шрифту формату json скористаємося конвертером TypeFace.js (http://gero3.github.io/facetype.js/).

Тепер можна задати шрифт, текст та його геометрію:

```
var fontLoader = new THREE.FontLoader();
fontLoader.load("./fonts/Academia Libera_Regular.json",function(tex){
var text_Geometry = new THREE.TextGeometry("Комп’ютерна графіка", {
size: 20,              // розмір тексту
height: 3,             // Товщина для видавлювання тексту
curveSegments: 6, // кількість точок (сегментів) кривої під час рисування літери
bevelEnabled: true,        // включення фаски
bevelThickness: 2, // глибина фаски
bevelSize: 1,          // ширина фаски
font: tex});           // назва шрифту
```

Підготуємо матеріал:

```
var text_Material = new THREE.MeshBasicMaterial({ color: 0x62254a });
```

Тепер створимо, власне, тривимірний об’єкт – «меш» із побудованими геометрією та матеріалом:

```
var text = new THREE.Mesh(text_Geometry, text_Material);
```

Якщо хочеться помістити текст по центру екрана:

```
text_Geometry.center();
scene.add(text);
```

Результат представлений на рисунку 6.2 (додаток Х).

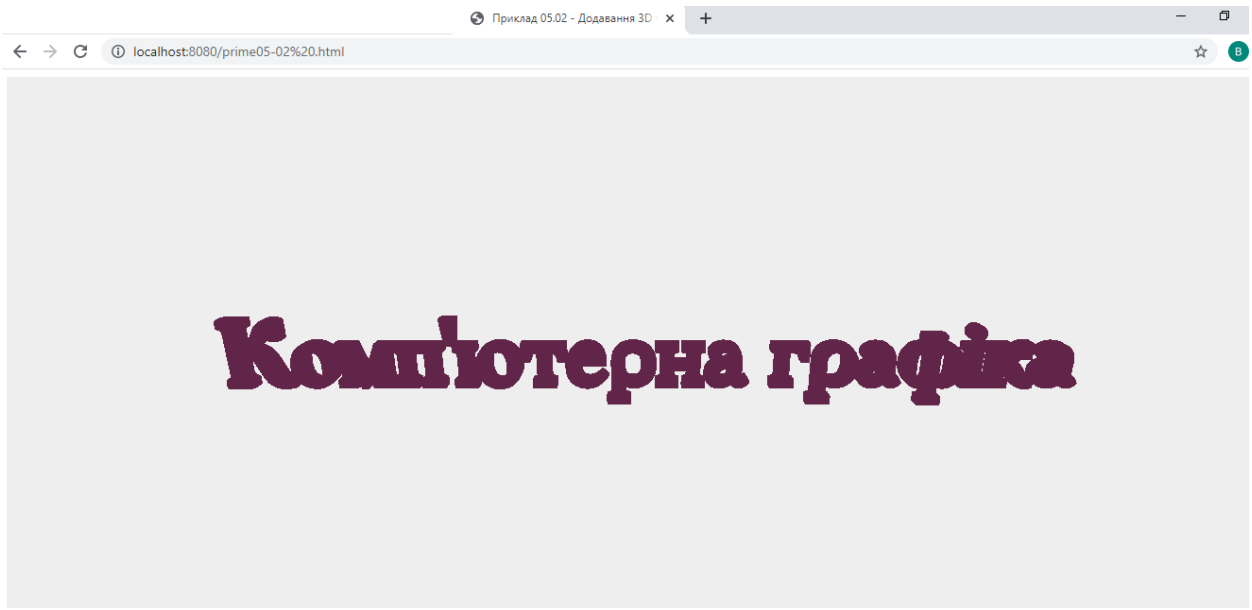


Рисунок 6.2 – Відображення тексту “Комп’ютерна графіка”

Зробимо цей приклад, але вже з накладенням текстури. Завантажимо довільну безкоштовну текстуру (файл fire.jpg), і розмножимо її:

```
var Texture = THREE.ImageUtils.loadTexture( ‘images/fire.jpg’);
Texture.wrapS = Texture.wrapT = THREE.RepeatWrapping;      // текстура просто повторюватиметься до нескінченності
Texture.repeat.set(0.05, 0.05);
var text_Material = new THREE.MeshBasicMaterial( { map: Texture } );
var text = new THREE.Mesh(text_Geometry, text_Material);
text_Geometry.center();
scene.add(text);
```

З цим матеріалом текст буде виглядати як (додаток Ц, рисунок 6.3).



Рисунок 6.3 – 3D текст з використанням однієї текстури

І, нарешті, розглянемо ще один приклад, уже із двома матеріалами. Перший матеріал буде призначений для переднього та заднього планів, а інший для фаски – «бічної» частини літер. Підготуємо картинки – файли fire.jpg і sky.jpg.

На їх основі підготуємо дві текстури Texture і ExtrTexture, потім, відповідно, два матеріали, Material і extrMaterial, і створимо масив materials:

```
var Texture = THREE.ImageUtils.loadTexture( ‘images/fire.jpg’);
Texture.wrapS = Texture.wrapT = THREE.RepeatWrapping;
Texture.repeat.set(0.05, 0.05);
var Material = new THREE.MeshBasicMaterial( { map: Texture } );
var ExtrTexture = THREE.ImageUtils.loadTexture( ‘images/sky.jpg’);
ExtrTexture.wrapS = ExtrTexture.wrapT = THREE.RepeatWrapping;
ExtrTexture.offset.set( 0, 0.7 ); //Зміщення текстури
ExtrTexture.repeat.set( 0, 0.13 );
var extrMaterial = new THREE.MeshBasicMaterial( { map: ExtrTexture } );
var materials = [ Material, extrMaterial ];
```

Тепер оголошуємо матеріал тексту:

```
var text = new THREE.Mesh(text_Geometry, materials);
```

та геометрію (зі змістом майбутнього тексту):

```
var fontLoader = new  THREE.FontLoader();
fontLoader.load("./fonts/Academia Libera_Regular.json",function(tex){
var text_Geometry = new THREE.TextGeometry("Комп’ютерна графіка", {
size: 20,                    // розмір тексту
height: 3,                   // товщина для видавлювання тексту
curveSegments: 6,                  // кількість точок (сегментів) кривої під час рисування літери
bevelEnabled: true,                // включення фаски
bevelThickness: 2,                 // глибина фаски
bevelSize: 1,                      // ширина фаски
Material: 0,                       //
ExtrudeMaterial: 1,                //
font: tex});                       // назва шрифту
```

Тут додалися два нові рядки. Вказуємо, що матеріал Material передніх і задніх частин розташовується в масиві materials першим, а матеріал бічної частини та фаски extrMaterial – другим (нумерація, природно, з нуля). Підсумок на рисунку 6.4 (додаток Ш).

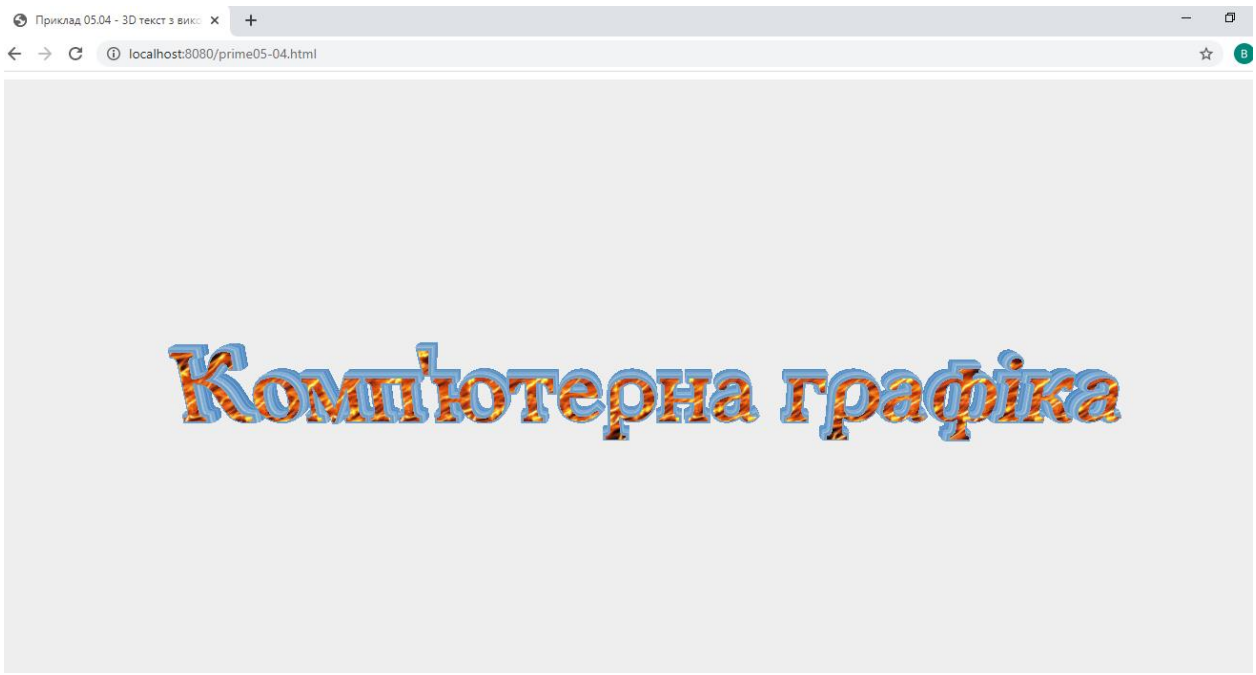


Рисунок 6.4 – 3D текст з використанням двох текстур

6.3 Таймер Clock

Таймер Three.js дозволяє відстежувати час, пройдений з моменту тієї чи іншої події, і має різноманітне застосування. Для додавання таймера напишемо:

```
var clock = new THREE.Clock( );
```

Конструктор
Clock(autoStart :Boolean)
Властивості:

- .autoStart: Boolean. Якщо встановлено, годинник запускається автоматично під час першого оновлення. Типово true;
- .startTime: Float Містить час, коли востаннє було викликано метод запуску годинника;
- .oldTime: Float Містить час, коли востаннє були викликані методи start, getElapsedTime або getDelta;
- .elapsedTime: Float Відстежує загальний час роботи годинника;
- .running: Boolean Годинник йде чи ні.

Клас має такі методи:

- .start():null Запускається годинник. Також встановлює startTimeioldTime на поточний час, встановлює elasedTime на 0 та running на true;
- .stop():null Зупиняє годинник та встановлює oldTime поточний час;

- .getElapsedTime() – повертає кількість секунд, які пройшли з моменту запуску годинника, і встановлює для oldTime поточний час. Якщо autoStart має значення true і годинник не працює, також запускається годинник;
- .getDelta() – повертає кількість мілісекунд (тип Float), які пройшли з попереднього звернення до таймера.

Розглянемо приклад із кубом, який змінює свій колір випадковим чином із заданим періодом. Оголосимо глобальні змінні (додаток Щ):

```
var clock = new THREE.Clock( );
var period = 1;
```

Додамо куб:

```
var cubeGeometry = New THREE.CubeGeometry(4,4,4);
var cubeMaterial = new THREE.MeshPhongMaterial( { color: 0x00ff00 } );
var cube = New THREE.Mesh(cubeGeometry, cubeMaterial);
cube.position.x = -4;
cube.position.y = 3;
cube.position.z = 0;
scene.add(cube);
```

Запускаємо таймер:

```
clock.start();
```

Тепер усередині renderScene() додамо:

```
if ( clock.running )
    {
      var time = clock.getElapsedTime(); if (time >
      period)
        {
          material.color.setHex( Math.random() * 0xffffff );
           clock.elapsedTime = 0; // Обнулюємо лічильник таймера
        }
    }
```

Тепер колір куба періодично змінюється (рисунок 6.5).

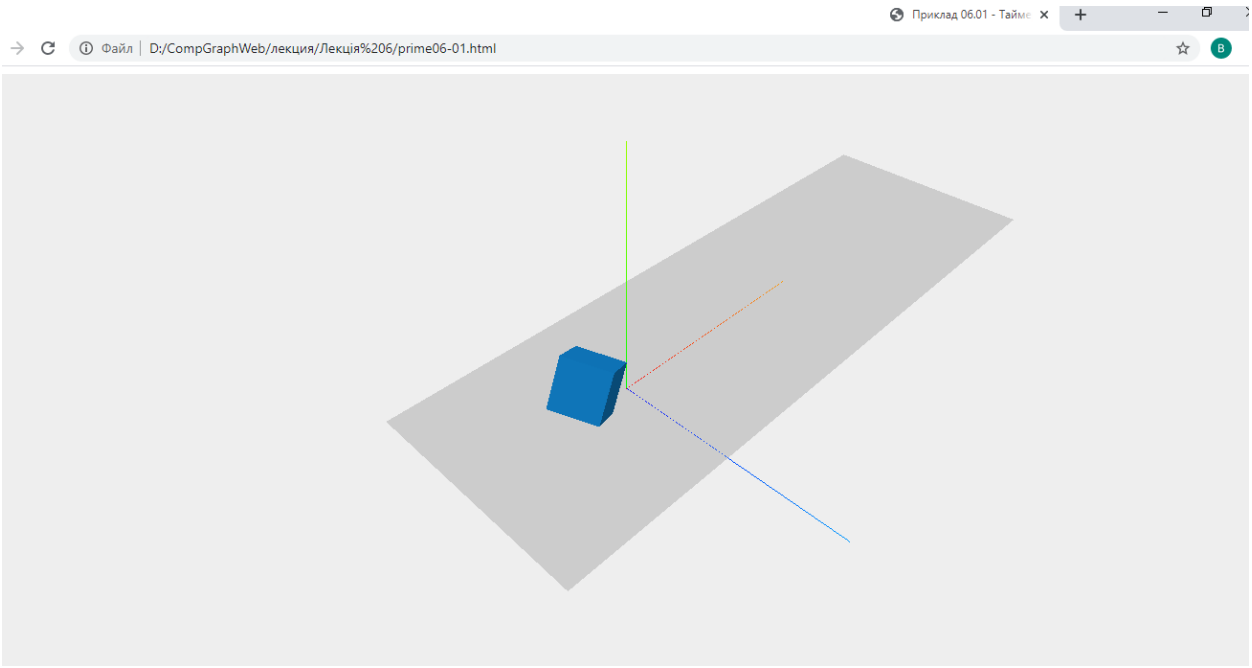


Рисунок 6.5 – Таймер Clock

6.4 Створення структурних об’єктів

Іноді доводиться створювати складну фігуру із кількох простих об’єктів. Тоді, за ідеєю, щоб вони рухалися, як єдине ціле, потрібно кожному об’єкту задавати те саме, загальне для всіх, правило руху. Звичайно, це було б досить стомлюючим. Постає питання, чи можна згрупувати кілька об’єктів в одну групу? Щоб вони, наприклад, рухалися як єдине ціле.

Для вирішення цієї проблеми можна використати клас тривимірних об’єктів Object3D. При створенні складної фігури всі її запчастини додаються не на сцену, а всередину об’єкта, і лише потім об’єкт виводиться на сцену.

Створимо, наприклад, модель «Капітошка» (рисунок 6.6) з кількох простих фігур (додаток Ю). Для цього спочатку оголосимо глобальний об’єкт:

```
var kapitoshka = новий THREE.Object3D();
```

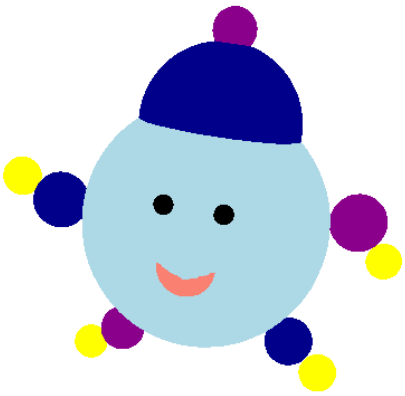


Рисунок 6.6 – Модель «Капітошка»

Як видно з рисунку, основу капітошки складає сфера:

```
var material = new THREE.MeshBasicMaterial({color: 0xADD8E6});
var geometry = new THREE.SphereGeometry(70, 36, 36);
var sphere1 = новий THREE.Mesh(geometry, матеріал);
sphere1.position.set(0, 60, 0);
```

Тепер додаємо цю сферу, але не на сцену, а до нашого об’єкту Object3D:

```
kapitoshka.add(sphere1);
```

Додамо шапочку з помпоном

```
material = new THREE.MeshBasicMaterial({color: 0x00008B});
geometry = new THREE.SphereGeometry(45, 36, 36);
var sphere2 = New THREE.Mesh(geometry, material);
sphere2.position.set(0, 115, 0);
kapitoshka.add(sphere2);

material = new THREE.MeshBasicMaterial({color: 0x8B008B});
geometry = new THREE.SphereGeometry(12, 16, 16);
var sphere3 = New THREE.Mesh(geometry, material);
sphere3.position.set(0, 166, 0);
kapitoshka.add(sphere3);
```

Оскільки «руки» та «ноги» однакові, ліву руку та ногу ми створили «клонуванням» правої руки та ноги.

```
material = new THREE.MeshBasicMaterial({color: 0x00008B});
geometry = new THREE.SphereGeometry(16, 16, 16);
var sphere4 = New THREE.Mesh(geometry, material);
sphere4.position.set(-85, 60, 0);
kapitoshka.add(sphere4);

material = new THREE.MeshBasicMaterial({color: 0x8B008B});
geometry = new THREE.SphereGeometry(16, 16, 16);
var sphere5 = новий THREE.Mesh(geometry, матеріал);
sphere5.position.set(84, 75, 0);
kapitoshka.add(sphere5);

material = new THREE.MeshBasicMaterial({color: 0x8B008B});
geometry = new THREE.SphereGeometry(13, 16, 16);
var sphere6 = новий THREE.Mesh(geometry, матеріал);
```

```
sphere6.position.set(-40, -10, 0);
kapitoshka.add(sphere6);

material = new THREE.MeshBasicMaterial({color: 0x00008B});
geometry = new THREE.SphereGeometry(14, 16, 16);
var sphere7 = новий THREE.Mesh(geometry, матеріал);
sphere7.position.set(58, 0, 0);
kapitoshka.add(sphere7);

material = new THREE.MeshBasicMaterial({color: 0xFFFF00});
geometry = new THREE.SphereGeometry(11, 16, 16);
var finger1 = новий THREE.Mesh(geometry, матеріал);
finger1.position.set(-109, 70, 0);
kapitoshka.add(finger1);

material = new THREE.MeshBasicMaterial({color: 0xFFFF00});
geometry = new THREE.SphereGeometry(10, 16, 16);
var finger2 = новий THREE.Mesh(geometry, матеріал);
finger2.position.set(102, 56, 0);
kapitoshka.add(finger2);

material = new THREE.MeshBasicMaterial({color: 0xFFFF00});
geometry = new THREE.SphereGeometry(10, 16, 16);
var finger3 = новий THREE.Mesh(geometry, матеріал);
finger3.position.set(-58, -22, 0);
kapitoshka.add(finger3);

material = new THREE.MeshBasicMaterial({color: 0xFFFF00});
geometry = new THREE.SphereGeometry(11, 16, 16);
var finger4 = новий THREE.Mesh(geometry, матеріал);
finger4.position.set(78, -16, 0);
kapitoshka.add(finger4);
```

Два ока:

```
material = new THREE.MeshLambertMaterial({color: 0x000000});
geometry = new THREE.SphereGeometry(5, 50, 11);
var eye1 = New THREE.Mesh(geometry, material);
eye1.position.set(-15, 80, 70);
kapitoshka.add(eye1);

material = new THREE.MeshLambertMaterial({color: 0x000000});
geometry = new THREE.SphereGeometry(5, 50, 11);
var eye2 = New THREE.Mesh(geometry, material);
eye2.position.set(15, 80, 70);
```

```
kapitoshka.add(eye2);
```

І, нарешті, рот:

```
material = new THREE.MeshBasicMaterial({color: 0xFA8072});
geometry = new THREE.CircleGeometry(15, 10, 0, Math.PI);
var mouse = new THREE.Mesh(geometry, material);
mouse.position.set(0, 50, 68);
mouse.rotation.z=Math.PI;
kapitoshka.add(mouse);
```

Вкажемо позицію капітошки і додамо його на сцену. Об’єкт Object3D будується від нульової координати, тобто, при вказівці координат позиції об’єкта в зазначеному місці буде не центр об’єкта, (як, наприклад, у куба), а його найнижча частина:

```
kapitoshka.position.set(0, 0, 0);
scene.add(kapitoshka);
```

Якщо тепер у рендері вказати

```
kapitoshka.rotation.y + = 0.02;
kapitoshka.rotation.z + = 0.02;
```

то наш капітошка буде обертатися як єдине ціле.

Завдання

- додати 3D – текст на сцену за допомогою свого українського шрифту (формат json);
- додати 3D – текст на сцену за допомогою однієї текстури;
- додати 3D – текст на сцену за допомогою двох текстур;
- використовуючи бібліотеку dat.GUI створити компоненти інтерфейсу користувача зміни розміру тексту;
- змініть колір випадково з заданим періодом;
- створіть складну фігуру із кількох простих об’єктів (використовуйте клас Object3D);
- використати відео як текстуру (додаток Ф);
- надіслати звіт та архів усіх файлів (включаючи папку з бібліотеками js, папку movies…)

Рекомендована література

1. Dirksen J. Learn Three.js : Program 3D Animations and Visualizations for the Web with JavaScript and WebGl. Packt Publishing, Limited, 2023. 554 p.

2. Фреймворк three.js : офіційний сайт. URL : http://threejs.org (дата звернення : 18.01.2024).

Контрольні запитання

1. Як додати 3D текст до сцени в бібліотеці three.js?
2. Як створити 3D текст з використанням певного шрифту в three.js?
3. Як налаштувати параметри 3D тексту, такі як розмір, колір, шрифт тощо, в three.js?
4. Як додати ефекти до 3D тексту, такі як відблиск, тінь або текстурні ефекти, в three.js?
5. Як динамічно оновлювати вміст 3D тексту, наприклад, змінювати текст або інші параметри, в реальному часі в three.js?
6. Як використовувати текстури на 3D тексті, наприклад, для нанесення зображень або текстур на букви, в three.js?
1. Як додати анімацію до 3D тексту, наприклад, змінювати його позицію, розмір або вигляд в залежності від подій або часу, в three.js?
2. Що таке таймер Clock в бібліотеці three.js і яку роль він відіграє в анімаційному движку?
3. Як створити екземпляр класу Clock в three.js та розпочати відлік часу?
4. Які методи доступні для таймера Clock в three.js і як вони використовуються для управління часом?
5. Як використовувати таймер Clock для створення анімацій у three.js?
6. Як зупинити, паузувати або скинути таймер Clock в three.js?
7. Як використовувати час, отриманий від таймера Clock, для синхронізації руху об’єктів або запуску подій в three.js?
8. Як налаштувати точність таймера Clock в three.js і як це впливає на продуктивність та точність анімацій?
9. Що таке структурні об’єкти в бібліотеці three.js і як вони використовуються для організації складних сцен?
10. Як створити базові структурні об’єкти, такі як групи (Group) або об’єкти (Object3D), в three.js?
11. Як додати об’єкти до структурних об’єктів в three.js і як управляти їх позицією, обертанням та масштабом?
12. Як організувати ієрархію об’єктів, вкладаючи один об’єкт у інший, в three.js?
13. Як використовувати структурні об’єкти для групування пов’язаних об’єктів разом для спільного управління?
14. Як використовувати структурні об’єкти для створення анімаційних ефектів, таких як рух групи об’єктів або їхніх трансформацій, в three.js?
15. Як використати відео як текстуру на поверхні об’єкта в three.js?
16. Як завантажити відео для використання як текстура в three.js та налаштувати його параметри?

17. Як відтворити відео на поверхні об’єкта в real-time в three.js?
18. Як налаштувати параметри відтворення відео, такі як автовідтворення, зациклення та контроль швидкості відтворення в three.js?
19. Як динамічно оновлювати відео текстури, наприклад, змінювати вміст або параметри відтворення, в реальному часі в three.js?
20. Як використовувати текстури з відео як частину матеріалів об’єктів в three.js?
21. Як додати відео як фонову текстуру для сцени в three.js?

7 СТВОРЕННЯ АНІМАЦІЇ ЗА ДОПОМОГОЮ TWEEN.JS

7.1 Побудова плоского кола

Клас CircleGeometry дозволяє будувати плоске коло у просторі (або його частину). Вказуються радіус, кількість сегментів, початковий кут та величина кута. Звіт іде проти годинникової стрілки.

Конструктор

CircleGeometry(radius:Float, segments: Integer, thetaStart: Float, thetaLength: Float). Тут:

Radius – радіус кола, за замовчуванням = 1;

segments – кількість сегментів (трикутників), мінімум = 3, за замовчуванням = 8;

thetaStart – початковий кут першого сегмента, за умовчанням = 0 (положення три години);

thetaLength – центральний кут, який часто називається тета, кругового сектора. За замовчуванням 2* Pi, що становить повне коло.

Наприклад, намалюємо логотип Запорізького національного університету (додаток Я, рисунок 7.1):

```
var circle_geometry = new THREE.CircleGeometry( 6, 100, 0, 2*Math.PI );
var circle_texture = New THREE.TextureLoader().load( ‘images/logo.jpg’ );//load texture
var circle_material = New THREE.MeshLambertMaterial({map: circle_texture});

var circle = new THREE.Mesh(circle_geometry, circle_material);
circle.castShadow = true;
circle.position.x=-60;
circle.position.y=45;
scene.add(circle);
```

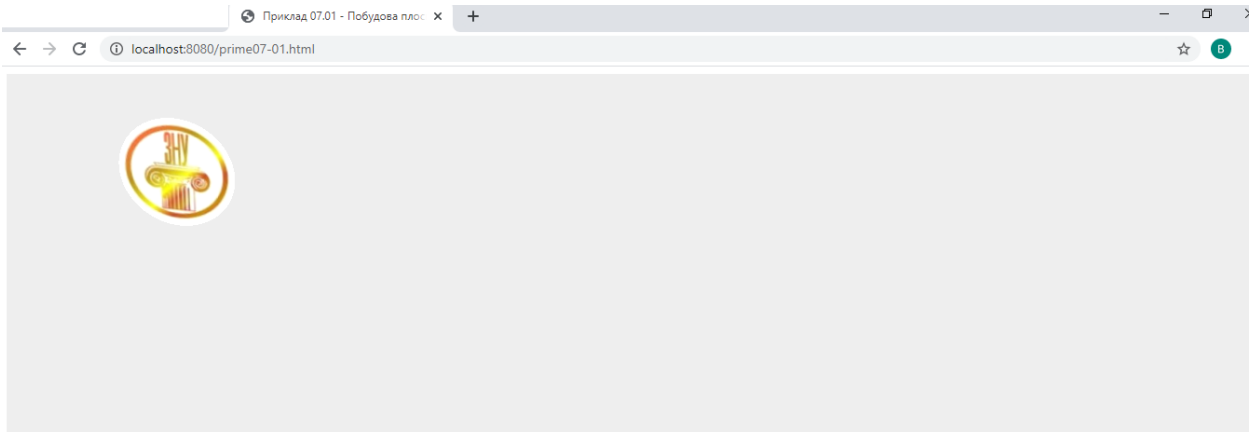


Рисунок 7.1 – Плоске коло у просторі

При цьому фігура буде видна лише з одного боку. Змінити це можна за допомогою параметра матеріалу side:

```
circle_material.side = THREE.DoubleSide;
```

Тепер наша фігура видна з обох боків.

7.2 Створення анімації за допомогою tween.js

У попередніх прикладах ми розглядали як створювати прості анімації. Коли у нас багато складних анімацій, код може швидко стати складним для підтримки чи розуміння. У цьому розділі ми розглянемо, як можна використовувати зовнішню бібліотеку JavaScript, яка спрощує створення анімації. Ми скористаємося для цього бібліотекою Tween.js (https://github.com/sole/tween.js/).

Оскільки це зовнішня бібліотека, нам спочатку потрібно переконатися, що вона включена до нашої HTML-сторінки. Для цього спочатку додамо елемент заголовка своєї сторінки наступне:

```
<script src="../js/tween.js"></script>
```

Анімація руху – це концепція, яка дозволяє плавно змінювати значення властивостей об’єкта. Ми просто вказуємо йому, які властивості хочемо змінити, які остаточні значення вони повинні мати після завершення роботи проміжного кадру і скільки часу це займе, а механізм анімації подбає про пошук проміжних значень від початкової до кінцевої точки. Наприклад, припустимо, що у нас є position об’єкт x і y координати:

```
var circle_position = {x: -300, y: 0};
```

Якщо ми хочемо змінити значення x з -300 на -60, зробимо наступне:

```
var circle_target = {};
circle_target.x = -60;
circle_target.y = 45;

// Створюємо анімацію руху
var circle_tween = New TWEEN.Tween (circle_position).
// Потім говоримо анімації руху, що ми хочемо анімувати властивість x протягом 4000 мілісекунд
to(circle_target, 4000);
```

Насправді це поки що нічого не дасть. Анімація руху створена, але не активна. Для цього потрібно:

```
// І встановлюємо запуск
circle_tween.start();
```

```
var circle_tween = New TWEEN.Tween(circle_position).to(circle_target, 4000);
circle_tween.start();
```

```
circle_tween.onUpdate(function(){
circle.position.x = circle_position.x;
circle.position.y = circle_position.y;
circle.rotation.x += 0.02;
circle.rotation.y+=0.02;
circle.rotation.z+=0.02;

});
```

Нарешті, ми маємо викликати TWEEN.update функцію в тому самому основному циклі, який ми використовуємо для анімації. Зазвичай це виглядає так (додаток 1):

```
function renderScene() {
TWEEN.update();
requestAnimationFrame(renderScene);
controls.update();
renderer.render (scene, camera);}
```

7.3 Зміна функції плавності

Tween.js буде виконувати інтерполяцію між значеннями лінійним чином, тому зміна буде прямо пропорційно минулому часу. Це передбачувано, Проте візуально досить нецікаво. Це можна легко змінити за допомогою easing методу. Наприклад:

```
circle_tween.easing(TWEEN.Easing.Quadratic.In);
```

Це призведе до того, що анімація руху почне повільно наближатися до кінцевого значення, прискорюючись до середини, а потім швидко досягає кінцевого значення.TWEEN.Easing.Quadratic.Out почне швидко змінюватися

у бік значення, але потім сповільниться при наближенні до остаточного значення (додаток 2).

У tween.js є кілька функцій плавності (рисунок 7.2). Вони згруповані за типом рівняння, яке вони представляють: Лінійне, Квадратичне, Кубічне і т.д., а потім за типом плавності: Вхід, Вихід, Вхід та Вихід.

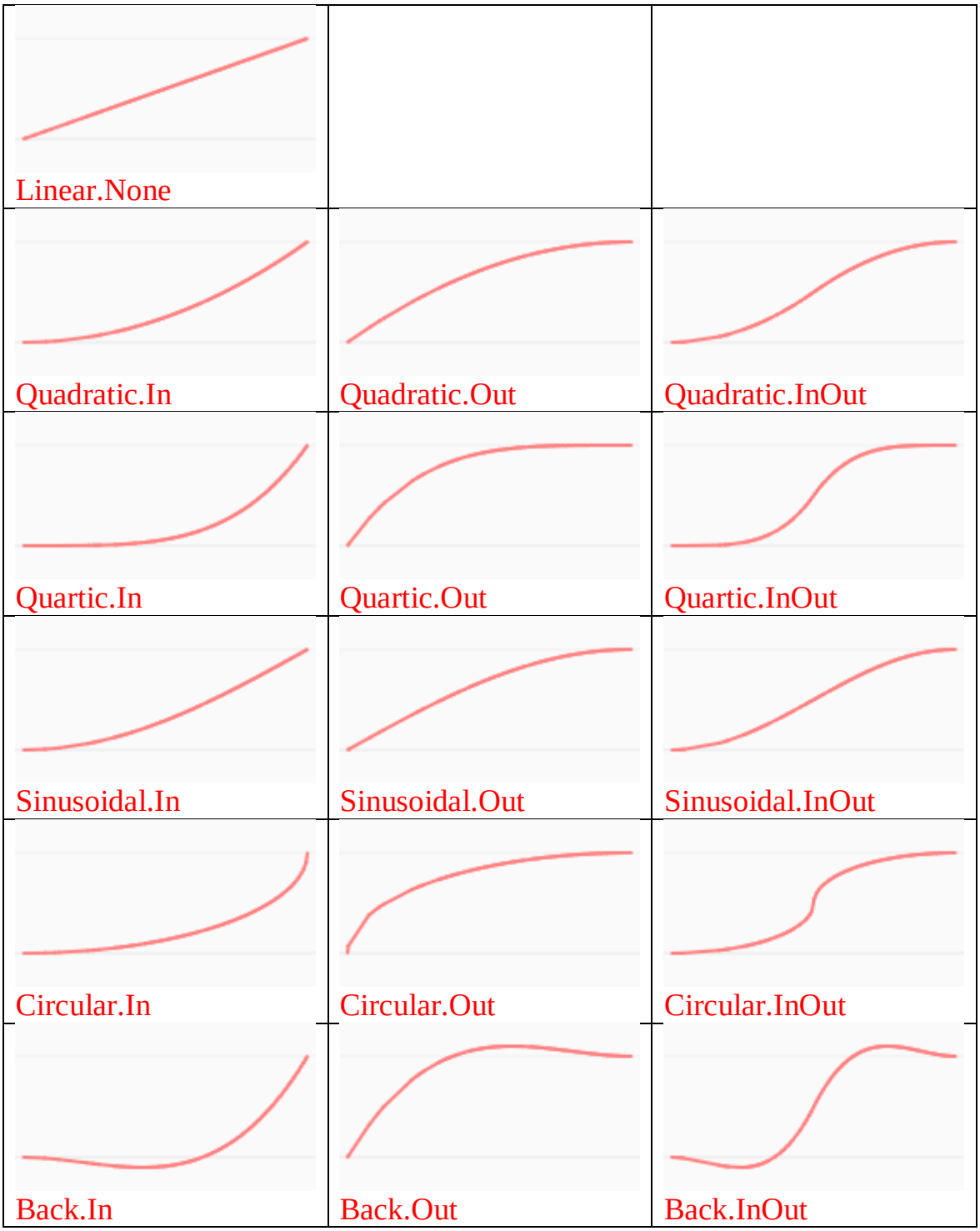


Рисунок 7.2 – Функції плавності, аркуш 1

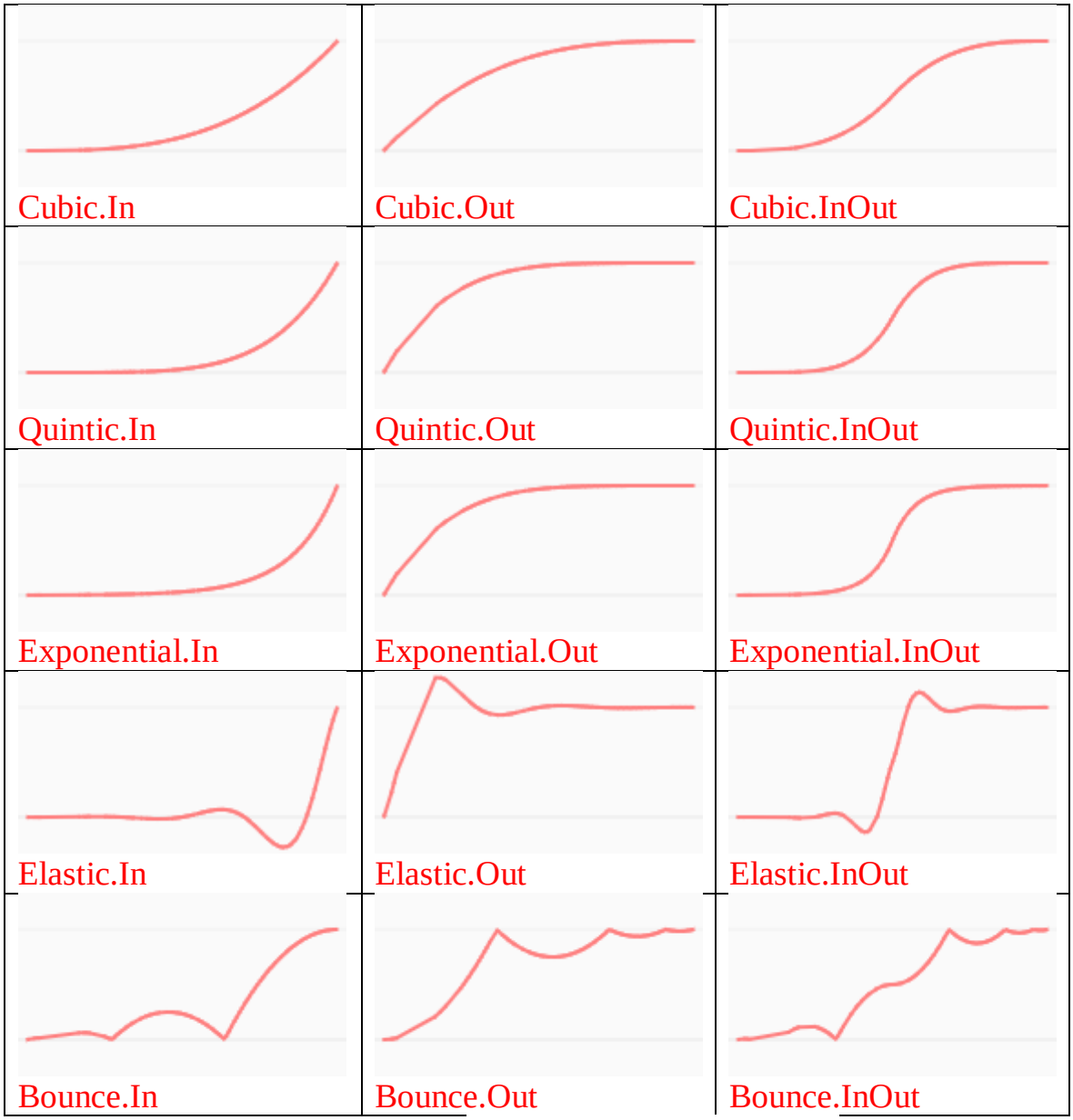


Рисунок 7.2 – Функції плавності, аркуш 2

7.4 Управління анімацією

Раніше ми роздивились Tween.start метод, але є й інші методи. Ймовірно, найважливішим з них є stop аналог start. Якщо ми хочемо скасувати анімацію руху, просто викликаємо цей метод для окремої анімації руху:

```
circle_tween.stop();
```

Якщо ми хочемо, щоб анімація повторювалася, для цього використовуємо метод repeat. Він приймає параметр, який описує скільки повторень ми хочемо після завершення першої анімації руху:

circle_tween_1.repeat(2); // повторюється 2 рази після першої анімації та зупинки

circle_tween.repeat(Infinity); //повторюється нескінченно

Загальна кількість анімацій руху буде параметром повтору плюс один для початкової анімації руху.

7.5 Функція yoyo

Ця функція діє лише при використанні разом із repeat. В активному стані анімація поводитиметься як yoyo, тобто вона переходитиме від початкового і кінцевого значень і назад, а не просто повторюватиме одну і ту ж послідовність з початку.

circle_tween.yoyo(true);

7.6 Функція delay

Більш складні схеми можуть зажадати затримки анімації до того, як вона дійсно почне працювати. Це ми можемо зробити за допомогою delay методу:

circle_tween.delay(5000);

Ще одна потужна функція – це можливість запускати власні функції у певні моменти життєвого циклу кожної анімації. Зазвичай це потрібно, коли недостатньо змінити властивості.

7.7 Функція onUpdate

Виконується щоразу під час оновлення анімації руху після фактичного оновлення значень.

```
circle_tween.onUpdate(function(){
circle.position.x = circle_position.x;
circle.position.y = circle_position.y;
circle.rotation.x += 0.02;
circle.rotation.y+=0.02;
circle.rotation.z+=0.02;
});
```

Анімований об’єкт передається як перший параметр.

7.8 Функція onComplete

Виконується, коли анімацію руху завершено нормально (тобто не зупинено).

Анімований об’єкт передається як перший параметр.

```
circle_tween.onComplete(function() {
var circle_tween_1 = новий TWEEN.Tween(circle_position).to (circle_target, 2000);
circle_tween_1.repeat(Infinity);
circle_tween_1.start();

circle_tween_1.onUpdate(function(){
circle.rotation.x += 0.02;
circle.rotation.y+=0.02;
circle.rotation.z+=0.02;
});
});
```

На рисунку 7.3 показано приклад використання бібліотеки Tween.js (додаток 3).

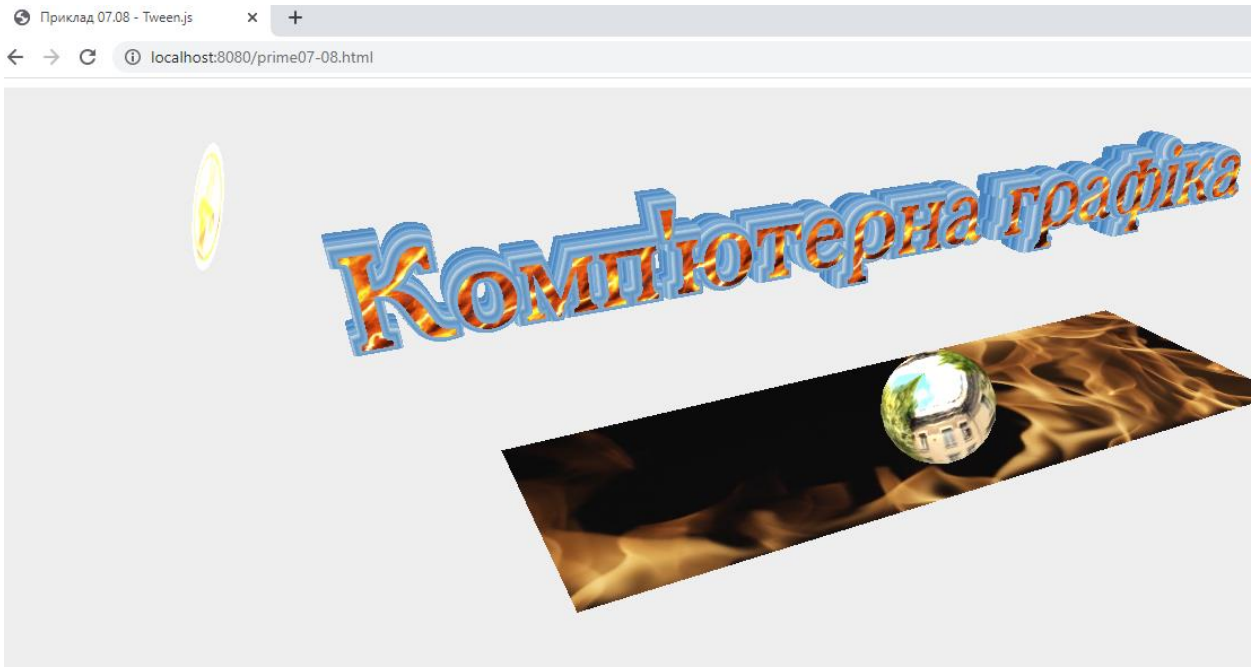


Рисунок 7.3 – Приклад використання бібліотеки Tween.js

7.9 Рисування ліній та площин

Для рисування фігур, які створені з ліній у просторі, використовується метод Line(geometry, material). Перший аргумент є екземпляром класу

Geometry та містить набір вершин фігури. Матеріал фігури матеріалу вибирається або LineBasicMaterial – для суцільних ліній, або LineDashedMaterial – для пунктирних ліній.

Зобразимо, наприклад, правильний шестикутник у просторі. Кожна вершина такого шестикутника лежить на стороні кута з величиною, кратною 60 градусів (радіани Math.PI/3).

Оголосимо нову геометрію та матеріал:

```
var geometry = new THREE.Geometry();
var material = new THREE.LineBasicMaterial( { color: 0xffffff } );
```

і в циклі додамо всі вершини:

```
for (var i = 0; i <= 6; i++){
var a = new THREE.Vector3(20*Math.cos((Math.PI/3)*i),
20*Math.sin((Math.PI/3)*i), 0);
geometry.vertices.push(a);
}
```

Для замикання кривої ми додали сім вершин, де сьома збігається з першою. Тепер створюємо фігуру line та додаємо на сцену:

```
var line = new THREE.Line(geometry, material);
scene.add(line);
```

Результат відображено на рисунку 7.4 (додаток 8).

Для зображення прямокутної частини поверхні використовується клас PlaneGeometry. Вказуємо ширину, висоту та кількість сегментів:

```
PlaneGeometry( width, height, widthSegments, heightSegments );
```

Допустимо ситуацію, що часто зустрічається, з перетином прямої і площини. Зобразимо площину:

```
var material = new THREE.MeshBasicMaterial({ wireframe: true, color:
0x21f344, side: THREE.DoubleSide });
var plane = new THREE.PlaneGeometry(80, 60, 1, 1);
var pl = new THREE.Mesh(plane, material);
pl.rotation.x = Math.PI/2;
pl.position.set(50, 0, 50);
scene.add(pl);
```

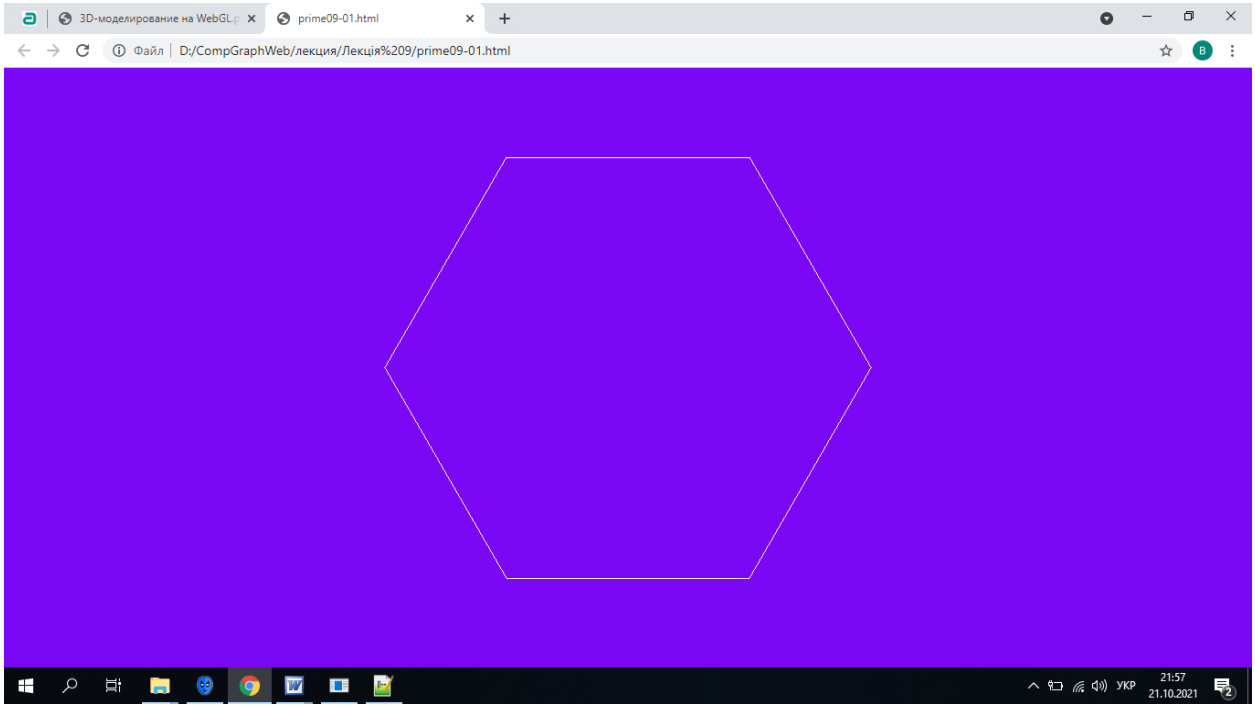


Рисунок 7.4 – Плоский шестикутник

Тут ми, щоб залишити лише межі площини, використовували параметр wireframe, який взяли рівним true. Тепер зобразимо пряму із двох шматків, нижню частину якої намалюємо пунктиром. При проведенні прямої потрібно спочатку вказати два вектори, кінці яких лежать на даній прямій, і додати їх до об’єкта класу THREE.Geometry:

```
var lineGeom = new THREE.Geometry();
var a = new THREE.Vector3(50, 60, 50);
var b = new THREE.Vector3(50, 0, 50);
lineGeom.vertices.push(a, b);
```

Тепер вкажемо матеріал прямій і створимо, власне, пряму:

```
var material = new THREE.LineBasicMaterial({ color: 0x0000ff });
var line = new THREE.Line(lineGeom, material);
scene.add(line);
```

Аналогічно збудуємо другу пряму.

```
var lineGeom = new THREE.Geometry();
var a = new THREE.Vector3(50, 0, 50);
var b = new THREE.Vector3(50, -60, 50);
lineGeom.vertices.push(a, b);
```

Оскільки ми її робитимемо пунктирною, обов’язково потрібно викликати відповідний метод:

lineGeometry.computeLineDistances();

У матеріалі прямої вкажемо довжину пунктиру та відстань між пунктирами:

var material = new THREE.LineDashedMaterial({ color: 0x0000ff, dashSize: 2, gapSize: 1 });
var line = new THREE.Line(lineGeom, material);
line.computeLineDistances();
scene.add(line);

Результат відображено на рисунку 7.5 (додаток 9).

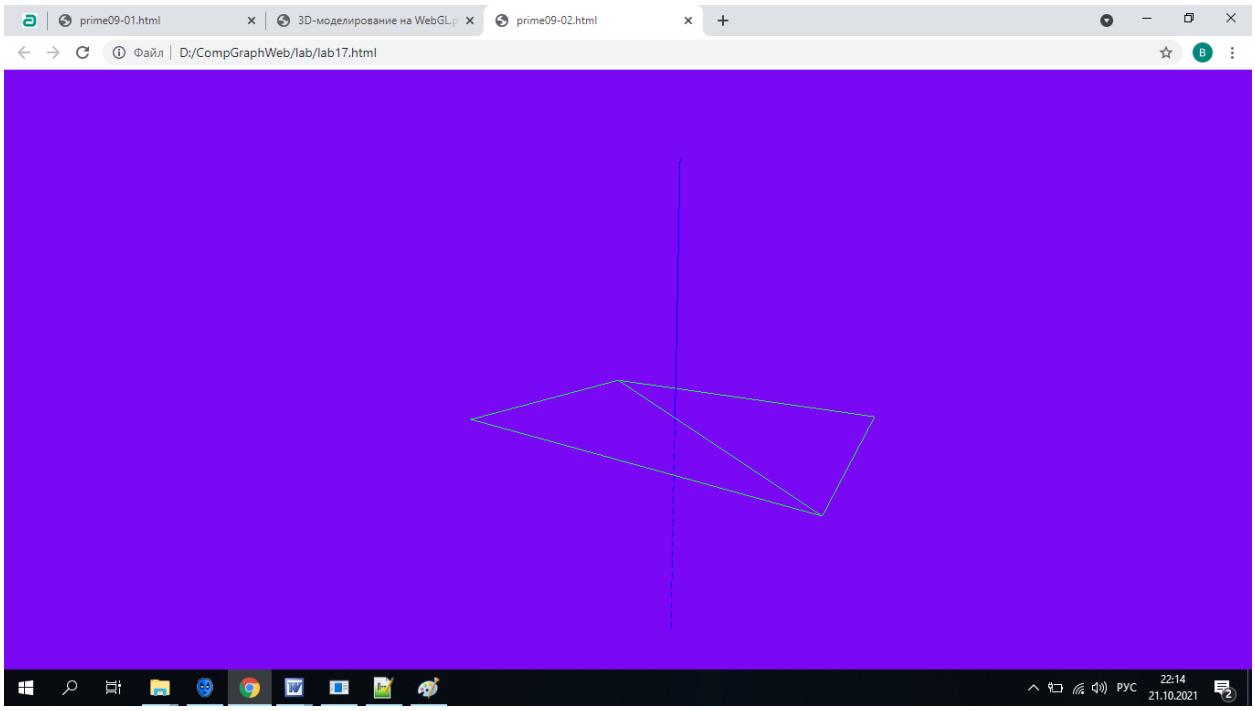


Рисунок 7.5 – Пряма та площина

7.10 Побудова параметричних кривих

Найпростіший спосіб побудови кривої – за допомогою класу Line практично так само, як ми будували. Зобразимо, наприклад, ланцюгову лінію. Параметрично ланцюгову лінію можна задати, наприклад, так:

$$\begin{cases} x = acost, \\ y = asint, \\ z = bt. \end{cases}$$

Оголосимо параметри кривої, геометрію та матеріал:

```
var a = 20, b = 4;
var geometry = new THREE.Geometry();
var material = new THREE.LineBasicMaterial({ color: 0x21f344 });
```

Залишилося заповнити геометрію вершинами:

```
for (var i =0; i <= 30; i++ ) {
var vec = new THREE.Vector3(a*Math.cos(i), a*Math.sin(i), b*i);
geometry.vertices.push(vec);
}
```

Нарешті, створюємо лінію та додаємо на сцену (рисунок 7.6, додаток 10):

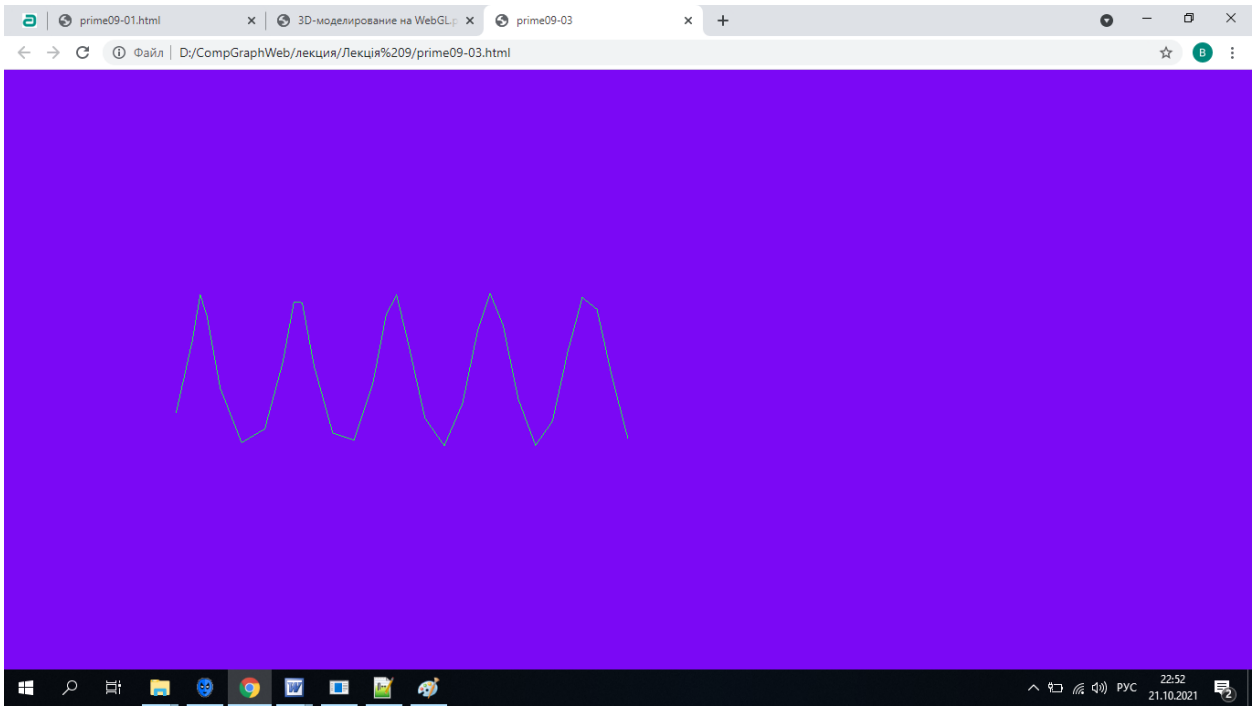


Рисунок 7.6 – Побудова параметричних кривих

```
var line = new THREE.Line(geometry, material);
scene.add(line);
```

Товщина лінії завжди дорівнює одиниці.

Завдання

- створити фігуру, яка створена з ліній у просторі, використовуючи метод Line (додаток 8);
- відобразити сцену з перетином прямої і площини (додаток 9);
- побудувати параметричну криву за допомогою класу Line (додаток 10);

- відобразіть логотип факультету за допомогою анімації (використовувати функцію плавності);
- відобразіть назву кафедри за допомогою анімації;
- відобразіть будь-який геометричний об’єкт (текстура довільний відеофільм), який обертається по колу;
- створіть плоске коло, в якості текстури використати логотип Запорізького національного університету. Анімувати плоске коло;
- надіслати звіт та архів усіх файлів (включаючи папку з бібліотеками js, папку movies…).

Рекомендована література

1. Dirksen J. Learn Three.js : Program 3D Animations and Visualizations for the Web with JavaScript and WebGl. Packt Publishing, Limited, 2023. 554 p.
2. Фреймворк three.js : офіційний сайт. URL : http://threejs.org (дата звернення : 18.01.2024).

Контрольні запитання

1. Як створити просту лінію в бібліотеці Three.js та додати її на сцену?
2. Які параметри можна налаштувати для лінії в Three.js, такі як колір, товщина та матеріал?
3. Як створити складніші лінії, наприклад, з кривими або вигнутими сегментами, в Three.js?
4. Як реалізувати анімацію руху ліній у бібліотеці Three.js?
5. Як створити площину в Three.js та додати її на сцену?
6. Які параметри можна налаштувати для площини в Three.js, такі як колір, текстура, матеріал тощо?
7. Як створити різні форми площини, такі як квадрати, прямокутники або полігональні форми, в Three.js?
8. Як додати текстуру на площину в Three.js та налаштувати її відображення?
9. Як використовувати площини в якості фону для сцени чи для відображення 2D зображень?
10. Що таке параметрична крива і які особливості вона має у відношенні до звичайних кривих?
11. Які види параметричних кривих підтримуються в бібліотеці Three.js?
12. Як створити просту параметричну криву, таку як кардіоїду або спіраль, у бібліотеці Three.js?
13. Які параметри можна налаштувати для параметричних кривих в Three.js, такі як радіус, кутова швидкість тощо?
14. Які методи доступні для анімації параметричних кривих, наприклад, зміна їхньої форми або позиції в часі?

15. Як використовувати параметричні криві для створення складних форм, наприклад, анімованих об’єктів або траєкторій руху?

16. Як додати текстуру на параметричну криву в Three.js та налаштувати її відображення?

17. Що таке бібліотека Tween.js і як вона використовується для створення анімацій в веб-розробці?

18. Як підключити бібліотеку Tween.js до проєкту та почати використовувати її функціонал?

19. Як створити анімаційний об’єкт за допомогою Tween.js та визначити його початкові та кінцеві стани?

20. Як додати різні типи анімацій, такі як рух, зміна розміру, зміна прозорості тощо, за допомогою Tween.js?

21. Як створити послідовні анімації, які відбуваються одна за одною, за допомогою Tween.js?

22. Як додати затримку перед початком анімації або після її завершення в Tween.js?

23. Як додати зворотній виклик (callback) після завершення анімації в Tween.js?

24. Як налаштувати різні параметри анімації, такі як тривалість, затухання (easing), затримка тощо, в Tween.js?

25. Як створити анімацію лінійної або кубічної інтерполяції між початковим та кінцевим станами об’єкта в Tween.js?

26. Як створити плоске коло в бібліотеці three.js?

27. Які параметри можна налаштувати при створенні плоского кола, такі як радіус, сегменти, матеріал тощо?

28. Як додати плоске коло на сцену в three.js?

29. Як створити текстуру для плоского кола та застосувати її?

30. Як анімувати плоске коло, наприклад, змінювати його розмір або позицію в часі?

8 КОНСТРУКТИВНА ТВЕРДОТІЛЬНА ГЕОМЕТРІЯ

Конструктивна твердотільна геометрія (CSG) – це метод моделювання, який використовує логічні операції, такі як об’єднання та перетин для об’єднання тривимірних тіл.

8.1 Логічні операції над моделлю за допомогою ThreeCSG

Розширена бібліотека може реалізувати математичні логічні операції з моделлю, по суті це рекомбінація і поділ точок для відновлення геометричних об’єктів і моделей сітки.

Метод Intersect (перетин) – використовуйте цей метод, щоб отримати загальну частину двох геометрій.

Метод union – використовуючи цей метод, можна об’єднати дві геометрії.

Метод subtract – використовуйте цей метод для видалення іншої геометрії і частини цієї геометрії, що повторюється, з однієї геометрії.

Рассмотрим пример куба и цилиндра (додаток 4, рисунок 8.1).

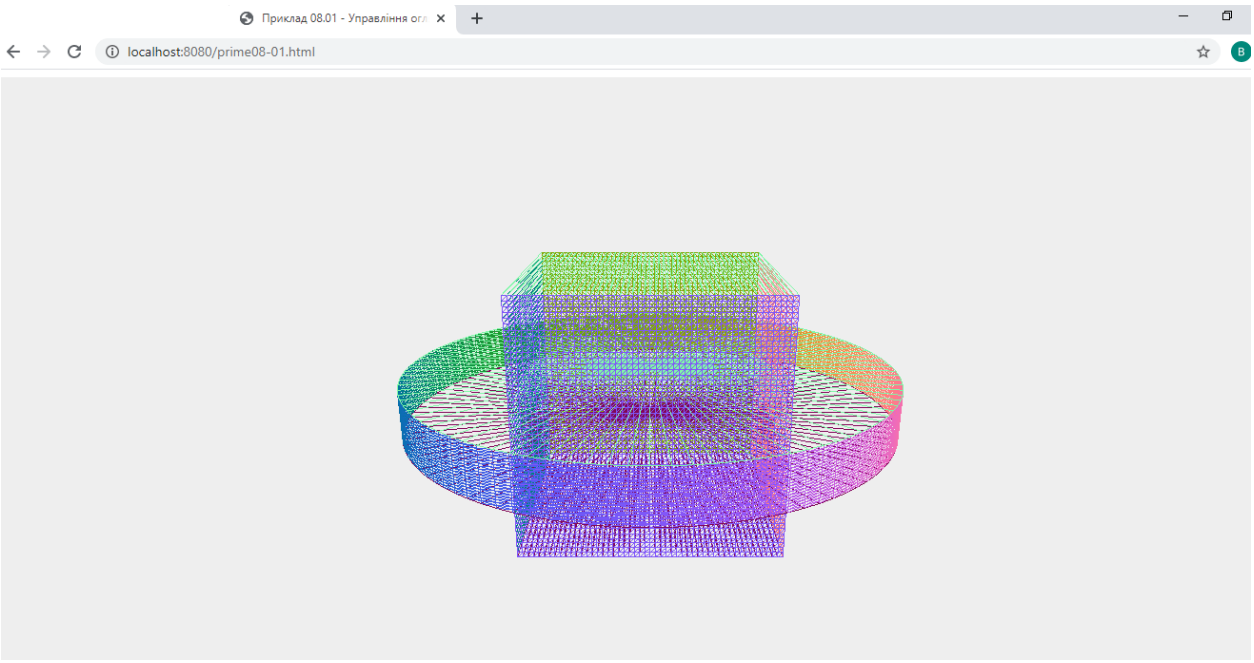


Рисунок 8.1 – Приклад використання бібліотеки ThreeCSG

– імпорт файлів бібліотеки

```
<script src="./js/ThreeCSG.js"></script>
```

– об’єкт перетворення

```
var cylinderBSP = new ThreeBSP(cylinder);
var boxBSP = new ThreeBSP(box);
```

Приклад знаходження точки перетину циліндра та куба (interset) (додаток 5, рисунок 8.2).

```
// Перетин двох об’єктів BSP
var newBSP = cylinderBSP.intersect(boxBSP);

// Перетворення об’єкта BSP на об’єкт Mesh
var newMesh = newBSP.toMesh(materials1);
scene.add( newMesh );
```

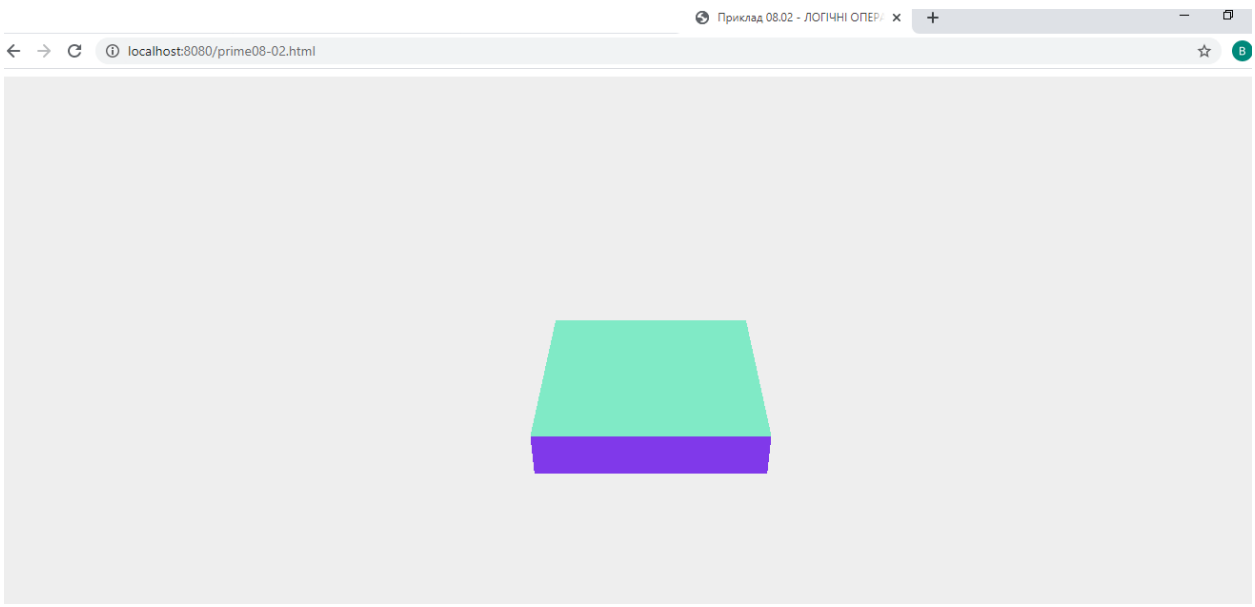


Рисунок 8.2 – Приклад знаходження точки перетину циліндра та куба (interset)

Приклад знаходження об’єднання циліндра та куба (union) (додаток 6, рисунок 8.3)

```
// Об’єднання двох об’єктів BSP
var newBSP = cylinderBSP.union(boxBSP);

// Перетворення об’єкта BSP на об’єкт Mesh
var newMesh = newBSP.toMesh(materials1);
scene.add( newMesh );
```

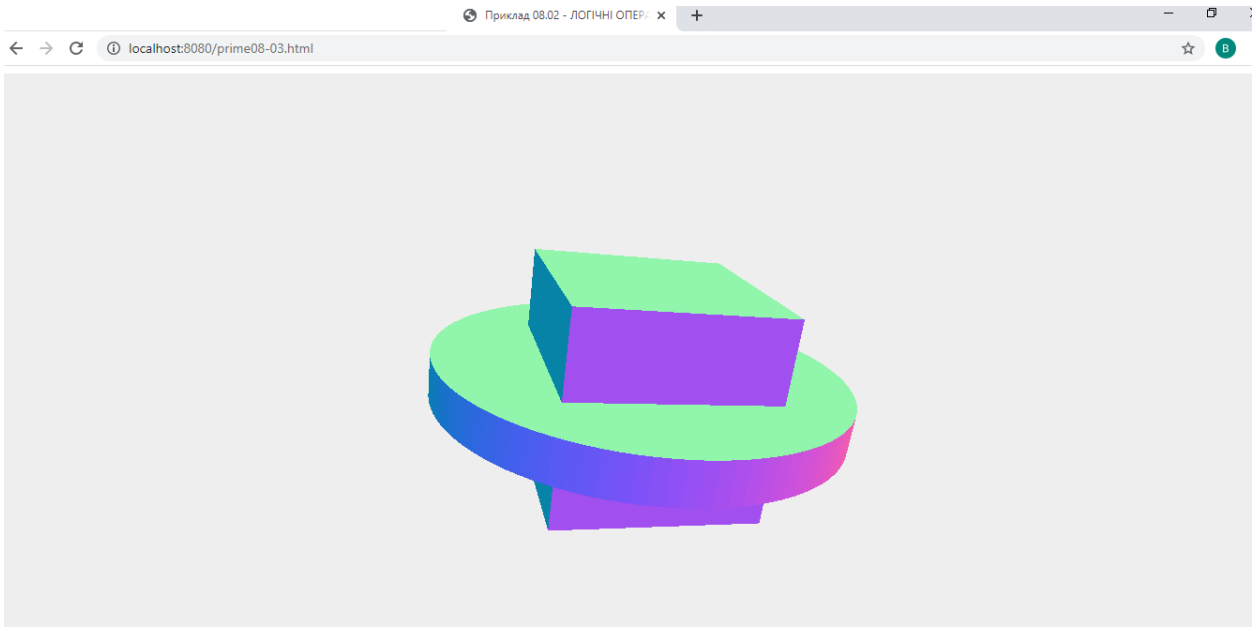


Рисунок 8.3 – Приклад знаходження об’єднання циліндра та куба (union)

Приклад різниці між циліндром та кубом (subtract) (додаток 7, рисунок 8.4)

```
// Різниця між двома об’єктами BSP
var newBSP = cylinderBSP.subtract(boxBSP);

// Перетворення об’єкта BSP на об’єкт Mesh
var newMesh = newBSP.toMesh(materials1);
scene.add( newMesh );
```

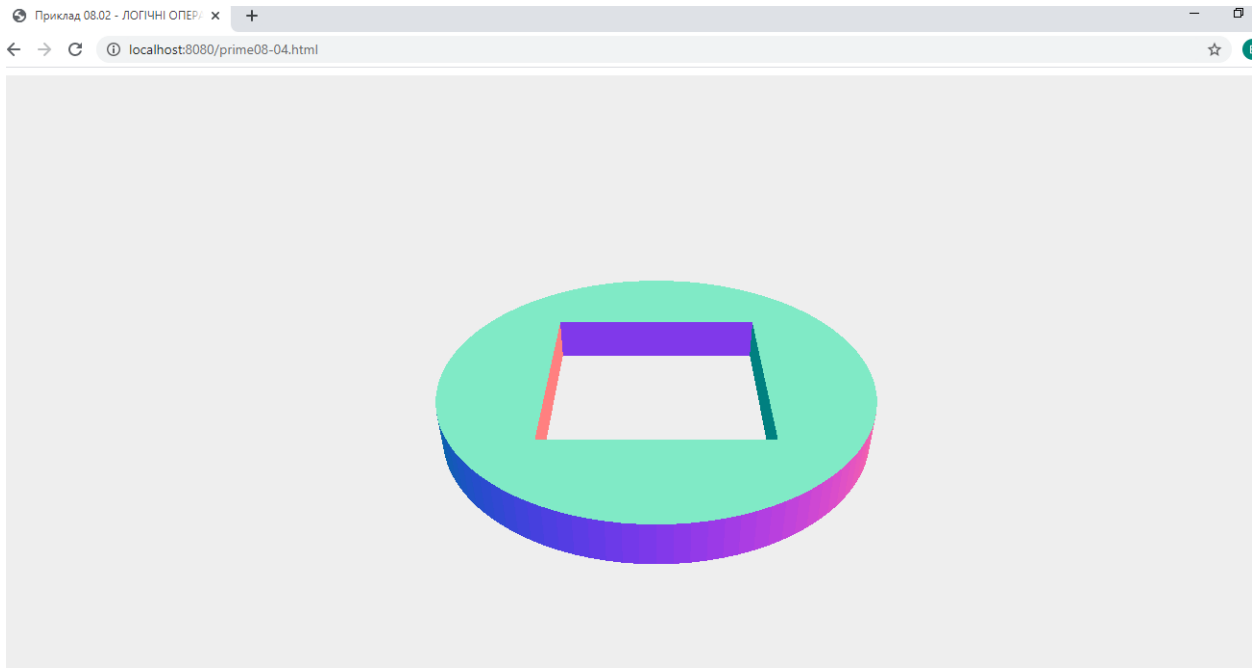


Рисунок 8.4 – Приклад різниці між циліндром та кубом (subtract)

8.2 Поверхні обертання

Тіло обертання виходить обертанням кривої навколо осі Y. Для створення поверхні обертання використовується клас LatheGeometry.

Конструктор

LatheGeometry(points, segments, phiStart, phiLength);

Вказується безліч точок points для побудови кривої, яка обертається, які просто з'єднуються точками; кількість сегментів радіального кола (за умовчанням дорівнює 12). Також можна вказати початковий кут phiStart та величину кута обертання phiLength. Нарисуємо, наприклад, шахівницю. Для побудови вигину за основу візьмемо функцію $10 + 18e^{-e^2}$.

Деталі королеви збиратимемо в тривимірному об'єкті queen. Опишемо також матеріал об'єкта:

```
var queen = new THREE.Object3D();
var material = new THREE.MeshPhongMaterial ({color: 0xdaa520, specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending, depthTest:true});
```

Тепер заповнюємо масив вершин:

```
var points = [];
for (var i=-2.4; i<0.7; i=i+0.1) {
 points.push(new THREE.Vector2(10+18*Math.exp(-i*i),34*i));
}
```

Додаємо в об'єкт та повертаємо

```
var geometry = new THREE.LatheGeometry(points,32);
object1=new THREE.Mesh(geometry,material);
object1.position.set(0,26,0);
object1.rotation.x=Math.PI;
queen.add(object1);
```

Результат наведений на рисунках 8.5 – 8.6 (додатки 11 – 12).

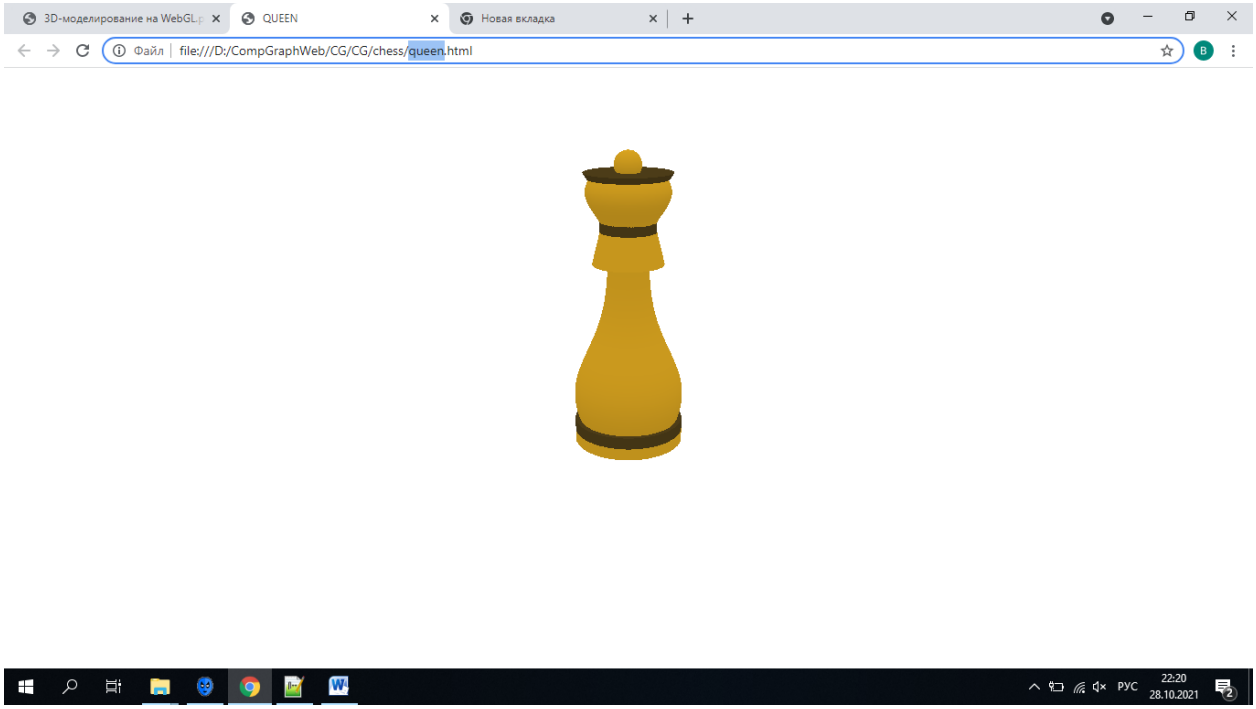


Рисунок 8.5 – Приклад рисування шахової фігури – королева

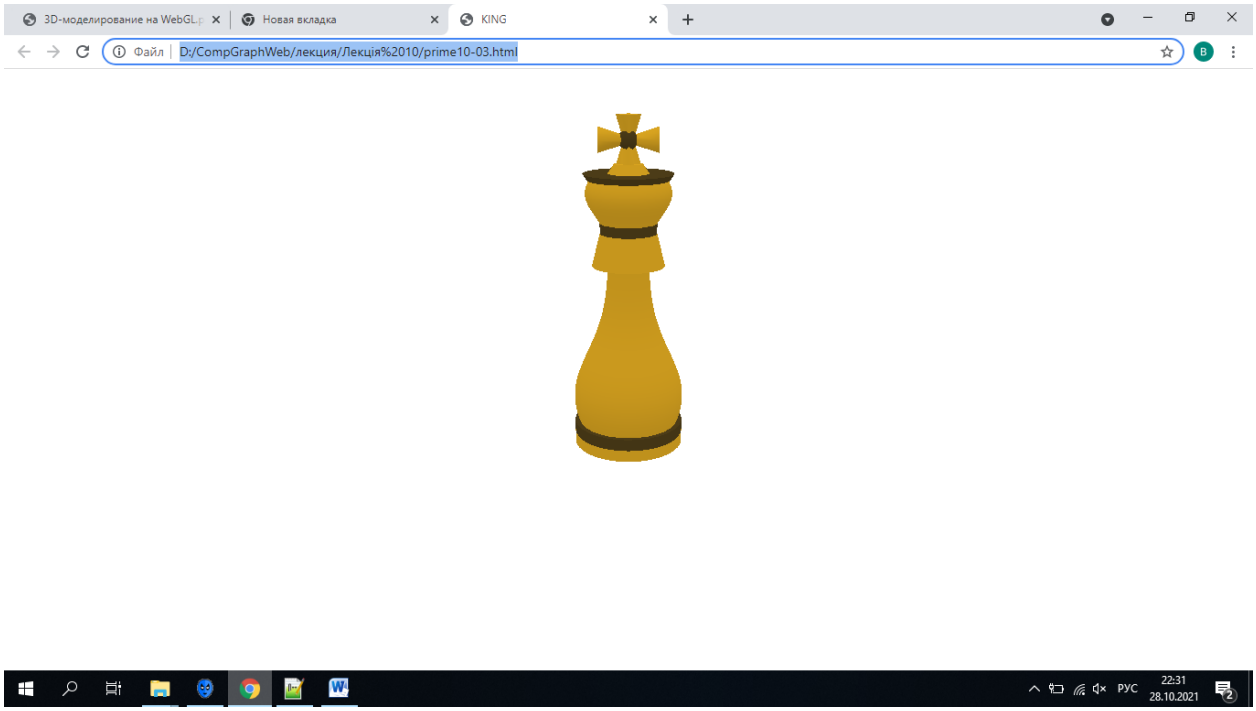


Рисунок 8.6 – Приклад рисування шахової фігури – король

8.3 Криволінійні циліндри з ExtrudeGeometry

Клас ExtrudeGeometry дозволяє рисувати поверхні, отримані «видавлюванням» будь-якої плоскої фігури вздовж прямої. З погляду математики, така фігура називається криволінійним циліндром. Щоправда, цей клас також дозволяє додавати фаску.

Синтаксис класу:

ExtrudeGeometry(shapes, options);

Фігура shapes утворюватиме підстави отриманого циліндра. Опції класу options багато в чому аналогічні опціям розглянутого раніше класу TextGeometry. Це й не дивно, оскільки останній, як було зазначено вище, є його спадкоємцем. Розглянемо приклад рисування тривимірної стрілки. Спочатку підготуємо креслення (рисунок 8.7).

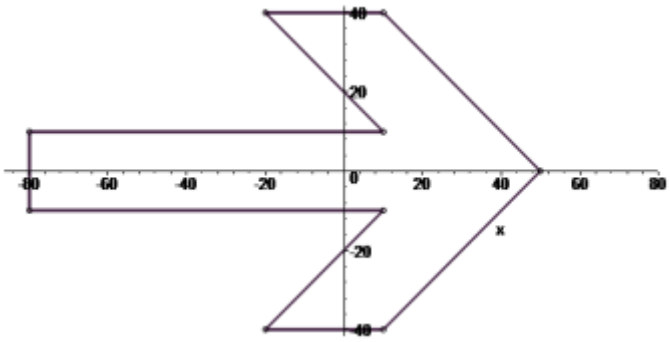


Рисунок 8.7 – Схема стрілки

Відповідно до креслення побудуємо фігуру arrowShape (додаток 13):

```
var arrowShape = new THREE.Shape();
( function roundedRect( ctx )
{ ctx.moveTo( -80, 10 );
ctx.lineTo( 10, 10 );
ctx.lineTo( -20, 40 );
 ctx.lineTo( 10, 40 );
ctx.lineTo( 50, 0 );
ctx.lineTo( 10, -40 );
ctx.lineTo( -20, -40 );
ctx.lineTo( 10, -10 );
ctx.lineTo( -80, -10 );
ctx.lineTo( -80, 10 ); } )( arrowShape);
```

Вкажемо параметри геометрії:

```
var size = 8;
var extrudeSettings = { amount: size, bevelSegments: 8, curveSegments: 32
};
```

Параметр amount відповідає за висоту екструзії. Оголосимо геометрію:

```
var geometry = new THREE.ExtrudeGeometry( arrowShape, extrudeSettings );
```

та матеріал:

```
var material = new THREE.MeshPhongMaterial({ color: 0x708090,
specular: 0x708090, ambient: 0x777777, shininess: 50 });
```

Нарешті, створюємо нашу тривимірну стрілку:

var arrow_right = new THREE.Mesh(geometry, material); arrow_right.position.set(100, 0, – size/2); scene.add(arrow_right);

Завдання

Точка А є центром основ одночасно циліндра та конуса. Основа циліндра лежить у площині xOy та має радіус *r*1; *h*2 і *h*3 – висоти конуса та усіченого конуса відповідно. Радіус основи конуса дорівнює *r*2.

Коло, яке є паралельним перерізом конуса, є також основою усіченого конуса і має радіус, удвічі менший за *r*2. Виходячи з отриманої інформації, побудуйте даний геометричний об’єкт (рисунок 8.8).

Щоб змінити налаштування, використовуйте dat.gui.

- створити поверхню обертання використовується клас LatheGeometry (додаток 11);
- створити тривимірну стрілку використовуючи клас ExtrudeGeometry (додаток 13);
- надіслати звіт та архів усіх файлів (включаючи папку з бібліотеками js, папку movies…).

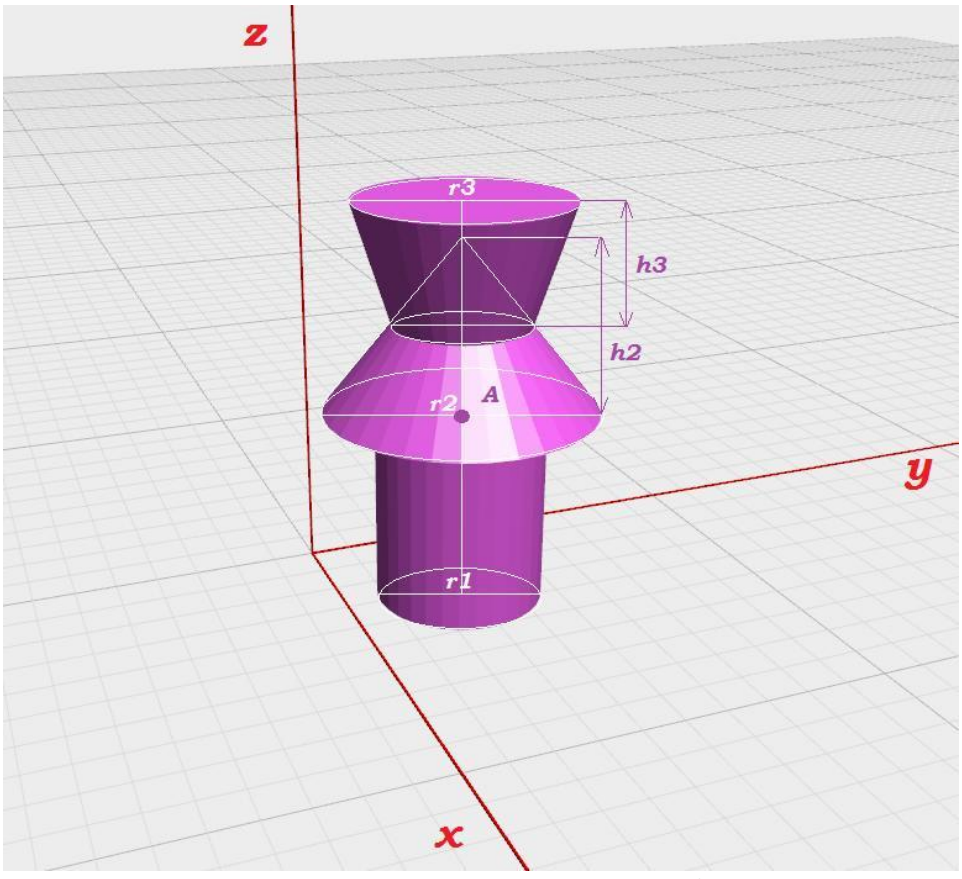


Рисунок 8.8 – Геометричний об’єкт

Рекомендована література

1. Dirksen J. Learn Three.js : Program 3D Animations and Visualizations for the Web with JavaScript and WebGl. Packt Publishing, Limited, 2023. 554 p.
2. Фреймворк three.js : офіційний сайт. URL : http://threejs.org (дата звернення : 18.01.2024).

Контрольні запитання

1. Що таке конструктивна твердотільна геометрія (CSG) і які вона має особливості?
2. Які базові операції доступні в CSG, такі як об’єднання, віднімання та перетин об’єктів?
3. Які переваги має використання CSG у порівнянні з традиційним методом моделювання об’єктів?
4. Які бібліотеки чи програмні засоби доступні для створення CSG моделей?
5. Які можливості надає CSG для створення складних геометричних об’єктів, таких як складні форми або вирізані деталі?
6. Які алгоритми лежать в основі операцій об’єднання, віднімання та перетину в CSG?
7. Які можливості надає CSG для створення анімованих об’єктів або об’єктів змінної форми?
8. Що таке поверхня обертання і як вона визначається у математичному та візуальному контексті?
9. Які види поверхонь обертання підтримуються в бібліотеці Three.js, наприклад, сфери тощо?
10. Як створити просту поверхню обертання, наприклад, сферу чи тор у бібліотеці Three.js?
11. Які параметри можна налаштувати для поверхонь обертання в Three.js, такі як радіус, кількість сегментів тощо?
12. Як реалізувати анімацію обертання поверхонь, наприклад, зміна їхнього положення чи швидкості обертання?
13. Як використовувати поверхні обертання для створення складних форм, наприклад, об’єктів змінної форми або анімованих об’єктів?
14. Як додати текстуру на поверхню обертання в Three.js та налаштувати її відображення?
15. Що таке криволінійний циліндр та в якому випадку його використання може бути корисним у тривимірній візуалізації?
16. Як використовувати ExtrudeGeometry в бібліотеці Three.js для створення криволінійних циліндрів?
17. Які параметри можна налаштувати під час створення криволінійних циліндрів з ExtrudeGeometry, такі як профіль, шлях, висота, радіус тощо?
18. Як створити складні форми криволінійних циліндрів, наприклад, з використанням складних профілів чи шляхів?

19. Як додати матеріали та текстури на криволінійні циліндри в Three.js, щоб надати їм реалістичний вигляд?

20. Як анімувати криволінійні циліндри, наприклад, змінювати їхню форму, положення або орієнтацію в просторі?

21. Як додати світло та тіні на криволінійні циліндри, щоб покращити їхню візуалізацію?

ВИКОРИСТАНА ЛІТЕРАТУРА

1. Dirksen J. Learn Three.js : Program 3D Animations and Visualizations for the Web with JavaScript and WebGl. Packt Publishing, 2023. 554 p.
2. Dirksen J. Learning Three.js : Cookbook. Birmingham; Mumbai : Packt Publishing, 2015. 300 p.
3. McGrath M. HTML, CSS & JavaScript In Easy Steps. Special Edition. Easy Steps Limite, 2020. 1275 p.
4. Noel Rappin. Modern CSS with Tailwind : Flexible Styling Without the Fuss. Pragmatic Bookshelf, 2022. 104 p.
5. Алекс В. Вайт. Основи графічного дизайну. Третє видання. Київ : ArtHuss, 2023. 232 с.
6. Молчанов В. П. Основи проектування WEB-видань : конспект лекцій. Харків : видавництво ХНЕУ, 2008. 168 с.
7. 3D-моделювання в браузері за допомогою three.js : Frontender Magazine. URL : http://frontender.info/building-3d-in-the-browser-with-three-js/#3dmodelirovanievbrauzerespomoshtyyuthreejs (дата звернення : 27.12.2023).
8. Learning three.js : iDigic. URL : http://learningthreejs.com/ (дата звернення : 12.01.2024).
9. Наукова бібліотека Запорізького національного університету : офіційний сайт. URL : http://library.znu.edu.ua/ (дата звернення : 14.11.2023).
10. Онлайн-підручник з HTML : W3Schools. URL : http://www.w3schools. com/html/ (дата звернення : 24.01.2024).
11. Онлайн-підручник з JavaScript : W3Schools. URL : http://www. w3schools.com/js/ (дата звернення : 24.01.2024).
12. Фреймворк three.js : офіційний сайт. URL : http://threejs.org (дата звернення : 18.01.2024).
13. Система електронного забезпечення навчання ЗНУ. URL : https://moodle.znu.edu.ua/ (дата звернення : 21.11.2023).

ДОДАТОК А

Приклад 01-01 структури HTML

```
<!DOCTYPE html>
<html>
<head>
	<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
	<title>Приклад 01-01 – Пуста сторінка </title>
	<script src="./js/three.js"></script>
</head>
<body>
	<script>
		init();
		function init()
		{
		};
	</script>
</body>
</html>
```

ДОДАТОК Б

Приклад 01-02 створення об’єктів на сцені

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 01-02 – Створення об’єктів на сцені </title>
<script src="./js/three.js"></script>
</head>
<body>
<script>
        init();
        function init()
        {
        // створюємо рендер і задаємо розмір
        var renderer =  new THREE.WebGLRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var axes = new THREE.AxisHelper( 20 );
        scene.add(axes);
        var planeGeometry = new THREE.PlaneGeometry(60,20,1,1);
        var planeMaterial = new THREE.MeshBasicMaterial({color: 0xcccccc});
        var plane = new THREE.Mesh(planeGeometry,planeMaterial);
        plane.rotation.x=-0.5*Math.PI;
        plane.position.x = 15;
        plane.position.y = 0;
        plane.position.z = 0;
        scene.add(plane);
        var cubeGeometry = new THREE.CubeGeometry(4,4,4);
        var cubeMaterial = new THREE.MeshBasicMaterial({color: 0xff0000,
wireframe: true});
        var cube = new THREE.Mesh(cubeGeometry, cubeMaterial);
        cube.position.x = -4;
        cube.position.y = 3;
```

```
        cube.position.z = 0;
        scene.add(cube);
        camera.position.x = -30;
        camera.position.y = 40;
        camera.position.z = 30;
        camera.lookAt(scene.position);
renderer.render( scene, camera );
};
</script>
</body>
</html>
```

ДОДАТОК В

Приклад 01-03 анімація

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 01-03 – Анімація </title>
<script src="./js/three.js"></script>
</head>
<body>
<script>
	init();
	function init() {
	// створюємо рендер і задаємо розмір
	var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
	renderer.setClearColor(new THREE.Color(0xEEEEEE));
	renderer.setSize( window.innerWidth, window.innerHeight );
	// додаємо вивід візуалізатора до html-елементу
	document.body.appendChild( renderer.domElement );
	// створюємо камеру
	var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
	// створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
	var scene = new THREE.Scene();
	var axes = new THREE.AxisHelper( 20 );
	scene.add(axes);
	var planeGeometry = new THREE.PlaneGeometry(60,20,1,1);
	var planeMaterial = new THREE.MeshBasicMaterial({color: 0xcccccc});
	var plane = new THREE.Mesh(planeGeometry,planeMaterial);
	plane.rotation.x=-0.5*Math.PI;
	plane.position.x = 15;
	plane.position.y = 0;
	plane.position.z = 0;
	scene.add(plane);
	var cubeGeometry = new THREE.CubeGeometry(4,4,4);
	var cubeMaterial = new THREE.MeshBasicMaterial({color: 0xff0000,
wireframe: true});
	var cube = new THREE.Mesh(cubeGeometry, cubeMaterial);
	cube.position.x = -4;
	cube.position.y = 3;
```

```
        cube.position.z = 0;
        scene.add(cube);
        camera.position.x = -30;
        camera.position.y = 40;
        camera.position.z = 30;
        camera.lookAt(scene.position);
        renderScene();
        function renderScene()    {
        cube.rotation.x += 0.02;
        cube.rotation.y += 0.02;
        cube.rotation.z += 0.02;
        requestAnimationFrame( renderScene );
        renderer.render( scene, camera );}};
</script>
</body>
</html>
```

ДОДАТОК Г

Приклад 01-04 Керування оглядом сцени

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 01-04 – Керування оглядом сцени </title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
</head>
<body>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var controls = new THREE.TrackballControls( camera );
        var axes = new THREE.AxisHelper( 20 );
        scene.add(axes);
        var planeGeometry = new THREE.PlaneGeometry(60,20,1,1);
        var planeMaterial = new THREE.MeshBasicMaterial({color: 0xcccccc});
        var plane = new THREE.Mesh(planeGeometry,planeMaterial);
        plane.rotation.x=-0.5*Math.PI;  plane.position.x = 15; plane.position.y = 0;
        plane.position.z = 0; scene.add(plane);
        var cubeGeometry = new THREE.CylinderGeometry( 5, 5, 20, 4 );
        var cubeMaterial = new THREE.MeshBasicMaterial({color: 0xff0000,
wireframe: true});
        var cube = new THREE.Mesh(cubeGeometry, cubeMaterial);
        cube.position.x = -4;
        cube.position.y = 3;
        cube.position.z = 0;
```

```
        scene.add(cube);
        camera.position.x = -30;
        camera.position.y = 40;
        camera.position.z = 30;
        camera.lookAt(scene.position);
        renderScene();
        function renderScene()    {
        cube.rotation.x += 0.02;
        cube.rotation.y += 0.02;
        cube.rotation.z += 0.02;
        requestAnimationFrame( renderScene );
        controls.update();
        renderer.render( scene, camera );}};
</script>
</body>
</html>
```

ДОДАТОК Д

Приклад 02-01 Використання бібліотеки dat gui

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 02.01 – Використання бібліотеки dat gui</title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
<script src="./js/dat.gui.js"></script>
</head>
<body>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
        THREE.WebGLRenderer({antialias:true}) : new
        THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var trackballcontrols = new THREE.TrackballControls( camera,
renderer.domElement );
        var axes = new THREE.AxisHelper( 20 );
        scene.add(axes);
        var planeGeometry = new THREE.PlaneGeometry(60,20,1,1);
        var planeMaterial = new THREE.MeshBasicMaterial({color: 0xcccccc});
        var plane = new THREE.Mesh(planeGeometry,planeMaterial);
        plane.rotation.x=-0.5*Math.PI;
        plane.position.x = 15;
        plane.position.y = 0;
        plane.position.z = 0;
        scene.add(plane);
        var cubeGeometry = new THREE.CubeGeometry(4,4,4);
```

```
        var cubeMaterial = new THREE.MeshBasicMaterial({color: 0xff0000,
wireframe: true});
        var cube = new THREE.Mesh(cubeGeometry, cubeMaterial);
        cube.position.x = -4;
        cube.position.y = 3;
        cube.position.z = 0;
        scene.add(cube);
        camera.position.x = -30;
        camera.position.y = 40;
        camera.position.z = 30;
        camera.lookAt(scene.position);
        var controls = new function() {
        this.y = 0.05;
        this.x = 0.02; }
        var gui = new dat.GUI();
        gui.add(controls, 'x',0,0.5);
        gui.add(controls, 'y',0,5);
        renderScene();
        function renderScene() {
        cube.rotation.x += controls.x;
        cube.rotation.y += controls.y;
        cube.rotation.z += controls.x;
        requestAnimationFrame( renderScene );
        trackballcontrols.update();
        renderer.render( scene, camera );
        }
};
</script>
</body>
</html>
```

ДОДАТОК Е

Приклад 02-02 – Додавання однієї текстури на об’єкт (клас ImageUtils)

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 02 – 02 – Додавання однієї текстури на об’єкт (клас
ImageUtils) </title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
<script src="./js/dat.gui.js"></script>
</head>
<body>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var trackballcontrols = new THREE.TrackballControls( camera,
renderer.domElement );
        var axes = new THREE.AxisHelper( 20 );
        scene.add(axes);
        var planeGeometry = new THREE.PlaneGeometry(60,20,1,1);
        var planeMaterial = new THREE.MeshBasicMaterial({color: 0xcccccc});
        var plane = new THREE.Mesh(planeGeometry,planeMaterial);
        plane.rotation.x=-0.5*Math.PI;
        plane.position.x = 15;
        plane.position.y = 0;
        plane.position.z = 0;
        scene.add(plane);
        var cube_geometry=new THREE.CubeGeometry(4, 4, 4);
```

```
        //Накладення 1 текстури
        var cube_material=new
THREE.MeshBasicMaterial({map:THREE.ImageUtils.loadTexture('./images/1.jp
g'),overdraw:true});
        cube=new THREE.Mesh(cube_geometry, cube_material);
        scene.add(cube);
        cube.position.x = -4;
        cube.position.y = 3;
        cube.position.z = 0;
        scene.add(cube);
        camera.position.x = -30;
        camera.position.y = 40;
        camera.position.z = 30;
        camera.lookAt(scene.position);
        var controls = new function() {
        this.rotationSpeed = 0.02; }
        var gui = new dat.GUI();
        gui.add(controls, 'rotationSpeed',0,0.5);
        renderScene();
        function renderScene() {
        cube.rotation.x += controls.rotationSpeed;
        cube.rotation.y += controls.rotationSpeed;
        cube.rotation.z += controls.rotationSpeed
        requestAnimationFrame( renderScene );
        trackballcontrols.update();
        renderer.render( scene, camera ); }
};
</script>
</body>
</html>
```

ДОДАТОК Ж

Приклад 02-03 – Декілька текстур на один об’єкт (клас ImageUtils)

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 02-03 – Накладення текстур (декілька текстур на один об’єкт) (клас ImageUtils) </title>
<script src="./js/three.js"></script>
<script src="./js/TrackballControls.js"></script>
<script src="./js/dat.gui.js"></script>
</head>
<body>
<script>
	init();
	function init(){
	// створюємо рендер і задаємо розмір
	var renderer = window.WebGLRenderingContext ? new THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
	renderer.setClearColor(new THREE.Color(0xEEEEEE));
	renderer.setSize( window.innerWidth, window.innerHeight );
	// додаємо вивід візуалізатора до html-елементу
	document.body.appendChild( renderer.domElement );
	// створюємо камеру
	var camera = new THREE.PerspectiveCamera(45,window.innerWidth / window.innerHeight, 0.1,1000);
	// створюємо сцену, яка буде містити всі наші елементи, такі як об’єкти, камери і джерела світла
	var scene = new THREE.Scene();
	var trackballcontrols = new THREE.TrackballControls( camera, renderer.domElement );
	var axes = new THREE.AxisHelper( 20 );
	scene.add(axes);
	var planeGeometry = new THREE.PlaneGeometry(60,20,1,1);
	var planeMaterial = new THREE.MeshBasicMaterial({color: 0xcccccc});
	var plane = new THREE.Mesh(planeGeometry,planeMaterial);
	plane.rotation.x=-0.5*Math.PI;
	plane.position.x = 15;
	plane.position.y = 0;
	plane.position.z = 0;
	scene.add(plane);
	//Накладення текстур (декілька текстур на один об’єкт)
```

```
        var materials = [];
        for (i=1; i<=6; i++) {
        var Texture = new THREE.ImageUtils.loadTexture(‘./images/’ + String(i) +
‘.jpg’ );
        var Material = new THREE.MeshBasicMaterial({ map: Texture, color:
0x00dcff} );
        materials.push( Material );}
        var material = new THREE.MeshFaceMaterial( materials ) ;
        var geometry = new THREE.BoxGeometry( 4, 4, 4);
        var cube = new THREE.Mesh( geometry, material );
        cube.position.x = -4;
        cube.position.y = 3;
        cube.position.z = 0;
        scene.add(cube);
        camera.position.x = -30;
        camera.position.y = 40;
        camera.position.z = 30;
        camera.lookAt(scene.position);
        var controls = new function() {
        this.rotationSpeed = 0.02; }
        var gui = new dat.GUI();
        gui.add(controls, ‘rotationSpeed’,0,0.5);
        renderScene();
        function renderScene() {
        cube.rotation.x += controls.rotationSpeed;
        cube.rotation.y += controls.rotationSpeed;
        cube.rotation.z += controls.rotationSpeed
        requestAnimationFrame( renderScene );
        trackballcontrols.update();
        renderer.render( scene, camera ); } };
</script>
</body>
</html>
```

ДОДАТОК И

Приклад 02-04 – Додавання однієї текстури на об’єкт (клас TextureLoader)

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 02-04 – Додавання однієї текстури на об’єкт (клас TextureLoader)</title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
<script src="./js/dat.gui.js"></script>
</head>
<body>
<script>
	init();
	function init() {
	// створюємо рендер і задаємо розмір
	var renderer = window.WebGLRenderingContext ? new THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
	renderer.setClearColor(new THREE.Color(0xEEEEEE));
	renderer.setSize( window.innerWidth, window.innerHeight );
	// додаємо вивід візуалізатора до html-елементу
	document.body.appendChild( renderer.domElement );
	// створюємо камеру
	var camera = new THREE.PerspectiveCamera(45,window.innerWidth / window.innerHeight, 0.1,1000);
	// створюємо сцену, яка буде містити всі наші елементи, такі як об’єкти, камери і джерела світла
	var scene = new THREE.Scene();
	var trackballcontrols = new THREE.TrackballControls( camera, renderer.domElement );
	var axes = new THREE.AxisHelper( 20 );
	scene.add(axes);
	var planeGeometry = new THREE.PlaneGeometry(60,20,1,1);
	var planeMaterial = new THREE.MeshBasicMaterial({color: 0xcccccc});
	var plane = new THREE.Mesh(planeGeometry,planeMaterial);
	plane.rotation.x=-0.5*Math.PI;
	plane.position.x = 15;
	plane.position.y = 0;
	plane.position.z = 0;
	scene.add(plane);
```

```
        var cube_geometry = new THREE.CubeGeometry(4, 4, 4);
        var cube_texture = new THREE.TextureLoader().load( ‘./images/1.jpg’
);//load texture
        var cube_material = new THREE.MeshBasicMaterial({map: cube_texture});
        var cube = new THREE.Mesh(cube_geometry, cube_material);
        cube.position.x = -4;
        cube.position.y = 3;
        cube.position.z = 0;
        scene.add(cube);
        camera.position.x = -30;
        camera.position.y = 40;
        camera.position.z = 30;
        camera.lookAt(scene.position);
        var controls = new function() {
        this.rotationSpeed = 0.02; }
        var gui = new dat.GUI();
        gui.add(controls, ‘rotationSpeed’,0,0.5);
        renderScene();
        function renderScene() {
        cube.rotation.x += controls.rotationSpeed;
        cube.rotation.y += controls.rotationSpeed;
        cube.rotation.z += controls.rotationSpeed
        requestAnimationFrame( renderScene );
        trackballcontrols.update();
        renderer.render( scene, camera ); }
};
</script>
</body>
</html>
```

ДОДАТОК К

Приклад 02-05 – Додавання декілька текстур на об’єкт (клас TextureLoader)

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 02.05 – Накладення текстур (декілька текстур на один об’єкт) (клас TextureLoader)</title>
<script src="./js/three.js"></script>
<script src="./js/TrackballControls.js"></script>
<script src="./js/dat.gui.js"></script>
</head>
<body>
<script>
	init();
	function init() {
	// створюємо рендер і задаємо розмір
	var renderer = window.WebGLRenderingContext ? new THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
	renderer.setClearColor(new THREE.Color(0xEEEEEE));
	renderer.setSize( window.innerWidth, window.innerHeight );
	// додаємо вивід візуалізатора до html-елементу
	document.body.appendChild( renderer.domElement );
	// створюємо камеру
	var camera = new THREE.PerspectiveCamera(45,window.innerWidth / window.innerHeight, 0.1,1000);
	// створюємо сцену, яка буде містити всі наші елементи, такі як об’єкти, камери і джерела світла
	var scene = new THREE.Scene();
	var trackballcontrols = new THREE.TrackballControls( camera, renderer.domElement );
	var axes = new THREE.AxisHelper( 20 );
	scene.add(axes);
	var planeGeometry = new THREE.PlaneGeometry(60,20,1,1);
	var planeMaterial = new THREE.MeshBasicMaterial({color: 0xcccccc});
	var plane = new THREE.Mesh(planeGeometry,planeMaterial);
	plane.rotation.x=-0.5*Math.PI;
	plane.position.x = 15;
	plane.position.y = 0;
	plane.position.z = 0;
	scene.add(plane);
```

```
        //Накладення текстур (декілька текстур на один об’єкт)
        var materials = [];
        for (i=1; i<=6; i++) {
        var Texture = new THREE.TextureLoader().load(‘./images/’ + String(i) +
‘.jpg’ );
        var Material = new THREE.MeshBasicMaterial({ map: Texture, color:
0x00dcff} );
        materials.push( Material ); }
        var material = new THREE.MeshFaceMaterial( materials ) ;
        var geometry = new THREE.BoxGeometry( 4, 4, 4);
        var cube = new THREE.Mesh( geometry, material );
        cube.position.x = -4;
        cube.position.y = 3;
        cube.position.z = 0;
        scene.add(cube);
        camera.position.x = -30;
        camera.position.y = 40;
        camera.position.z = 30;
        camera.lookAt(scene.position);
        var controls = new function() {
        this.rotationSpeed = 0.02; }
        var gui = new dat.GUI();
        gui.add(controls, ‘rotationSpeed’,0,0.5);
        renderScene();
        function renderScene() {
        cube.rotation.x += controls.rotationSpeed;
        cube.rotation.y += controls.rotationSpeed;
        cube.rotation.z += controls.rotationSpeed
        requestAnimationFrame( renderScene );
        trackballcontrols.update();
        renderer.render( scene, camera ); }
};
</script>
</body>
</html>
```

ДОДАТОК Л

Приклад 02-06 – Матеріали Three.js

```
<!DOCTYPE html>
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 02.03 – Матеріали  Three.js </title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
 <script src="./js/dat.gui.js"></script>
</head>
<body>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
renderer.setClearColor(new THREE.Color(0xEEEEEE));
renderer.setSize( window.innerWidth, window.innerHeight );
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        camera.position.set(-30, 40, 30);
        camera.lookAt(scene.position);
        var controls = new THREE.TrackballControls( camera,
renderer.domElement);
        var light = new THREE.DirectionalLight(0xffffff, 0.5);
        light.position.setScalar(1);
        scene.add(light);
        scene.add(new THREE.AmbientLight(0xffffff, 0.5));
        var axes = new THREE.AxisHelper( 20 );
        scene.add(axes);
        var planeGeometry = new THREE.PlaneGeometry(60,20,1,1);
        var planeMaterial = new THREE.MeshBasicMaterial({color: 0xcccccc});
        var plane = new THREE.Mesh(planeGeometry,planeMaterial);
        plane.rotation.x=-0.5*Math.PI;
```

```
        plane.position.x = 15;
        plane.position.y = 0;
        plane.position.z = 0;
        scene.add(plane);
        var materials = {
        choose: "LambertMaterial",
        BasicMaterial: new THREE.MeshBasicMaterial({color: "red"}),
        LambertMaterial: new THREE.MeshLambertMaterial({color: "green"}),
        PhongMaterial: new THREE.MeshPhongMaterial({color: "blue"}),
        StandardMaterial:new THREE.MeshStandardMaterial({color: "yellow"})
        }
        var Geometry = new THREE.SphereGeometry(4, 32,32);
        var mesh = new THREE.Mesh(Geometry, materials[materials.choose]);
        mesh.position.x = -4;
        mesh.position.y = 3;
        mesh.position.z = 0;
        scene.add(mesh);
        var gui = new dat.GUI();
        gui.add(materials, "choose", ["BasicMaterial", "LambertMaterial",
"PhongMaterial", "StandardMaterial"]).onChange((val) => {console.log(val);
mesh.material = materials[val];});
        renderScene();
        function renderScene() {
        requestAnimationFrame( renderScene );
        controls.update();
        renderer.render( scene, camera );}
}
</script>
</body>
</html>
```

ДОДАТОК М

Приклад 02-07 – ALPHAMAP (карта прозорості)

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 02.08 – ALPHAMAP (карта прозорості)</title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
<script src="./js/dat.gui.js"></script>
</head>
<body>
<script>
	init();
	function init() {
	// створюємо рендер і задаємо розмір
	var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
	renderer.setClearColor(new THREE.Color(0xEEEEEE));
	renderer.setSize( window.innerWidth, window.innerHeight );
	// додаємо вивід візуалізатора до html-елементу
	document.body.appendChild( renderer.domElement );
	// створюємо камеру
	var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
	// створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
	var scene = new THREE.Scene();
	var trackballcontrols = new THREE.TrackballControls( camera,
renderer.domElement );
	var light = new THREE.DirectionalLight( 0xffffff,1 );
	light.position.set( 0, 0, 200 ).normalize();
	scene.add(light);
	var axes = new THREE.AxisHelper( 20 );
	scene.add(axes);
	var planeGeometry = new THREE.PlaneGeometry(60,20,1,1);
	var planeMaterial = new THREE.MeshBasicMaterial({color: 0xcccccc});
	var plane = new THREE.Mesh(planeGeometry,planeMaterial);
	plane.rotation.x=-0.5*Math.PI;
	plane.position.x = 15;
	plane.position.y = 0;
	plane.position.z = 0;
```

```
        scene.add(plane);
        var alphamap = new THREE.ImageUtils.loadTexture('./images/1.jpg');
        alphamap.wrapS = THREE.RepeatWrapping;
        alphamap.wrapT = THREE.RepeatWrapping;
        alphamap.repeat.set( 1, 1 );
        var cube_material = new THREE.MeshPhongMaterial({transparent:true,
color:0xFF0000,alphaMap:alphamap,alphaTest:0.5, side: THREE.DoubleSide});
        cube_geometry=new THREE.CubeGeometry(4, 4, 4);
        cube=new THREE.Mesh(cube_geometry, cube_material);
        scene.add(cube);
        cube.position.x = -4;
        cube.position.y = 3;
        cube.position.z = 0;
        scene.add(cube);
        camera.position.x = -30;
        camera.position.y = 40;
        camera.position.z = 30;
        camera.lookAt(scene.position);
        var controls = new function() {
        this.rotationSpeed = 0.02; }
        var gui = new dat.GUI();
        gui.add(controls, 'rotationSpeed',0,0.5);
        renderScene();
        function renderScene() {
        cube.rotation.x += controls.rotationSpeed;
        cube.rotation.y += controls.rotationSpeed;
        cube.rotation.z += controls.rotationSpeed;
        requestAnimationFrame( renderScene );
        trackballcontrols.update();
        renderer.render( scene, camera ); }
};
</script>
</body>
</html>
```

ДОДАТОК Н

Приклад 05-01 – Проста сцена, яка включає наступні елементи: камеру, пряме світло, площину і дві сфери

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title> Проста сцена, яка включає наступні елементи: камеру, пряме світло, площину і дві сфери  </title>
<script src="./js/three.min.js"></script>
 <script src="./js/OrbitControls.js"></script>
</head>
<body>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        renderer.shadowMapSoft = true;
        renderer.shadowMap.enabled = true;
        renderer.shadowMap.type = THREE.PCFSoftShadowMap;
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо сцену, яка буде містити всі наші елементи, такі як об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth / window.innerHeight, 1,10000);
        camera.position.set(-1200, 600, 1600);
        camera.lookAt(new THREE.Vector3(0,0,0));
        var controls = new THREE.OrbitControls(camera, renderer.domElement);
        controls.target = new THREE.Vector3(0, 0, 0);
        // завантажити текстуру
        var texture = THREE.ImageUtils.loadTexture(‘./images/texture.png’);
        texture.repeat.set(10, 10);
        texture.wrapS = texture.wrapT = THREE.RepeatWrapping;
        texture.anisotropy = 16;
        texture.needsUpdate = true;
```

```
        var textureBump = THREE.ImageUtils.loadTexture('./images/bump.png');
        textureBump.repeat.set(10, 10);
        textureBump.wrapS = textureBump.wrapT = THREE.RepeatWrapping;
        textureBump.anisotropy = 16;
        textureBump.needsUpdate = true;
        // підготувати годинник
        var clock = new THREE.Clock();
        // додати directional light
        dLight = new THREE.DirectionalLight(0xffffff,1);
        dLight.position.set(0, 400, 0);
        dLight.castShadow = true;
        dLight.shadowMapWidth = dLight.shadowMapHeight = 1000;
        var d = 500;
        dLight.shadow.camera.left = – d;
        dLight.shadow.camera.right = d;
        dLight.shadow.camera.top = d;
        dLight.shadow.camera.bottom = – d;
        dLight.shadow.camera.far = 1000;
        dLight.shadow.bias = – 0.0001;
        scene.add(dLight);
        // додати площину
        var plane_Geometry = new THREE.PlaneGeometry(1200, 1200, 1, 1);
        plane = new THREE.Mesh(plane_Geometry, new
THREE.MeshLambertMaterial({color: 0x9669FE}));
        plane.position.y = -20;
        plane.rotation.x = – Math.PI / 2;
        plane.receiveShadow = true;
        scene.add(plane);
        // створити new group (Object3D)
        var group = new THREE.Object3D();
        // додати дві сфери
        var sphere  = drawSphere(-100, 150, new THREE.MeshPhongMaterial({
map: texture, bumpMap: textureBump, color: 0x00ff00, specular: 0xff2200,
emissive: 0x004000 }));
        var sphere2 = drawSphere( 100, 150, new THREE.MeshPhongMaterial({
map: texture, bumpMap: textureBump, color: 0x00ff00, specular: 0xff2200,
shininess: 3 }));
        //.. і додати їх до групи
        group.add(sphere);
        group.add(sphere2);
        scene.add(group);
        function drawSphere(x, z, material) {
        var sphere = new THREE.Mesh(new THREE.SphereGeometry(70, 70, 20),
material);
        sphere.rotation.x = sphere.rotation.z = Math.PI * Math.random();
```

```
        sphere.position.x = x;
        sphere.position.y = 100;
        sphere.position.z = z;
        sphere.castShadow = sphere.receiveShadow = true;
        return sphere;}
        renderScene();
        function renderScene() {
        requestAnimationFrame( renderScene );
        controls.update(clock.getDelta());
        renderer.render( scene, camera );}
};
</script>
</body>
</html>
```

ДОДАТОК П

Приклад 05-02 – Помічник ArrowHelper

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 05.02  – Помічник ArrowHelper </title>
<script src="./js/three.min.js"></script>
 <script src="./js/OrbitControls.js"></script>
</head>
<body>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        renderer.shadowMapSoft = true;
        renderer.shadowMap.enabled = true;
        renderer.shadowMap.type = THREE.PCFSoftShadowMap;
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 1,10000);
        camera.position.set(-1200, 600, 1600);
        camera.lookAt(new THREE.Vector3(0,0,0));
        var controls = new THREE.OrbitControls(camera, renderer.domElement);
        controls.target = new THREE.Vector3(0, 0, 0);
        // завантажити текстуру
        var texture = THREE.ImageUtils.loadTexture(‘./images/texture.png’);
        texture.repeat.set(10, 10);
        texture.wrapS = texture.wrapT = THREE.RepeatWrapping;
        texture.anisotropy = 16;
        texture.needsUpdate = true;
        var textureBump = THREE.ImageUtils.loadTexture(‘./images/bump.png’);
        textureBump.repeat.set(10, 10);
```

```
        textureBump.wrapS = textureBump.wrapT = THREE.RepeatWrapping;
        textureBump.anisotropy = 16;
        textureBump.needsUpdate = true;
        // підготувати годинник
        var clock = new THREE.Clock();
        // додати directional light
        dLight = new THREE.DirectionalLight(0xffffff,1);
        dLight.position.set(0, 400, 0);
        dLight.castShadow = true;
        dLight.shadowMapWidth = dLight.shadowMapHeight = 1000;
        var d = 500;
        dLight.shadow.camera.left = – d;
        dLight.shadow.camera.right = d;
        dLight.shadow.camera.top = d;
        dLight.shadow.camera.bottom = – d;
        dLight.shadow.camera.far = 1000;
        dLight.shadow.bias = – 0.0001;
        scene.add(dLight);
        // додати площину
        var plane_Geometry = new THREE.PlaneGeometry(1200, 1200, 1, 1);
        plane = new THREE.Mesh(plane_Geometry, new
THREE.MeshLambertMaterial({color: 0x9669FE}));
        plane.position.y = -20;
        plane.rotation.x = – Math.PI / 2;
        plane.receiveShadow = true;
        scene.add(plane);
        // створити new group (Object3D)
        var group = new THREE.Object3D();
        // додати дві сфери
        var sphere  = drawSphere(-100, 150, new THREE.MeshPhongMaterial({
map: texture, bumpMap: textureBump, color: 0x00ff00, specular: 0xff2200,
emissive: 0x004000 }));
        var sphere2 = drawSphere( 100, 150, new THREE.MeshPhongMaterial({
map: texture, bumpMap: textureBump, color: 0x00ff00, specular: 0xff2200,
shininess: 3 }));
        //.. і додати їх до групи
        group.add(sphere);
        group.add(sphere2);
        scene.add(group);
        // add helpers:
        // 1. ArrowHelper
        var directionV3 = new THREE.Vector3(1, 0, 1);
        var originV3 = new THREE.Vector3(0, 200, 0);
        var arrowHelper = new THREE.ArrowHelper(directionV3, originV3, 100,
0xff0000, 20, 10); // 100 is length, 20 and 10 are head length and width
```

```
        scene.add(arrowHelper);
        function drawSphere(x, z, material) {
        var sphere = new THREE.Mesh(new THREE.CylinderGeometry(70, 70,
20), material);
        sphere.rotation.x = sphere.rotation.z = Math.PI * Math.random();
        sphere.position.x = x;
        sphere.position.y = 100;
        sphere.position.z = z;
        sphere.castShadow = sphere.receiveShadow = true;
        return sphere;}
        renderScene();
        function renderScene() {
        requestAnimationFrame( renderScene );
        controls.update(clock.getDelta());
        renderer.render( scene, camera );}
};
</script>
</body>
</html>
```

ДОДАТОК Р

Приклад 06-00 – Тіні

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 06.00 – Тіні </title>
<script src="./js/three.js"></script>
<script src="./js/TrackballControls.js"></script>
</head>
<body>
<script>
init();
function init() {
// створюємо рендер і задаємо розмір
var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
renderer.setClearColor(new THREE.Color(0xEEEEEE));
renderer.setSize( window.innerWidth, window.innerHeight );
renderer.shadowMap.enabled = true;
renderer.shadowMap.type = THREE.PCFSoftShadowMap; // фільтрує карти
тіней з використанням алгоритму
// процентної фільтрації (PCF) з кращими м’якими тінями,
// особливо при використанні карт тіней з низькою якістю.
// додаємо вивід візуалізатора до html-елементу
document.body.appendChild( renderer.domElement );
// створюємо камеру
var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
// створюємо сцену, яка буде містити всі наші елементи, такі як об’єкти,
камери і джерела світла
var scene = new THREE.Scene();
var trackballcontrols = new THREE.TrackballControls( camera,
renderer.domElement );
// додаємо освітлення ambient lighting
// AmbientLight – це світло, яке освітлює всю сцену.
// Воно не має напрямку і освітлює кожен об’єкт сцени в рівній мірі,
// незалежно від розташування об’єкта.
// Відповідно, у цього світла немає позиції на осі координат.
// Цей світ не може використовуватися для створення тіней.
var ambiColor = "#e3e1e1";
var ambientLight = new THREE.AmbientLight(ambiColor);
```

```
scene.add(ambientLight);
// Джерело DirectionalLight можна розглядати як джерело світла, що
знаходиться дуже далеко.
// Весь світ промені, які він випромінює, паралельні один одному.
var directionalLight = new THREE.DirectionalLight( 0xffffff );
directionalLight.position.set( -10, 100, 0 );

directionalLight.castShadow = true; // Якщо встановлено значення true, це
// світло відкидає тіні.
directionalLight.shadowCameraNear = 2; // Тіньова камера перспективи
directionalLight.shadowCameraFar = 200;
directionalLight.shadowCameraLeft = -50;
directionalLight.shadowCameraRight = 50;
directionalLight.shadowCameraTop = 50;
directionalLight.shadowCameraBottom = -50;

directionalLight.distance = 0;
directionalLight.intensity = 0.5;
directionalLight.shadowMapHeight = 1024; // Висота текстури карти тіней в
пікселях.
// За замовчуванням – 512.
directionalLight.shadowMapWidth = 1024; // Ширина текстури карти тіней в
пікселях.
// За замовчуванням – 512.
scene.add(directionalLight);
var planeGeometry = new THREE.PlaneGeometry(40,40,1,1);
var planeMaterial = new THREE.MeshPhongMaterial({color: 0x9999ff, side:
THREE.DoubleSide});
var plane = new THREE.Mesh(planeGeometry,planeMaterial);
plane.rotation.x= Math.PI/2;
//       plane.position.x = 15;
plane.position.y = 7;
plane.position.z = 0;
plane.receiveShadow = true;
plane.castShadow = true;
scene.add(plane);
var planeGeometry_2 = new THREE.PlaneGeometry(60,60,1,1);
var planeMaterial_2 = new THREE.MeshPhongMaterial( { color: 0xff00ff, side:
THREE.DoubleSide });
var plane2 = new THREE.Mesh(planeGeometry_2,planeMaterial_2);

plane2.rotation.x= Math.PI/2;;
//       plane2.position.x = 20;
plane2.position.y = 0;
plane2.position.z = 0;
```

```
plane2.receiveShadow = true;
scene.add(plane2);
var cubeGeometry = new THREE.CubeGeometry(4,4,4);
var cubeMaterial = new THREE.MeshBasicMaterial({color: 0xff0000});
var cube = new THREE.Mesh(cubeGeometry, cubeMaterial);
cube.position.x = -4;
cube.position.y = 10;
cube.position.z = 0;
cube.castShadow = true;
scene.add(cube);
camera.position.x = 60;
camera.position.y = 20;
camera.position.z = 0;
camera.lookAt(scene.position);
renderScene();
function renderScene()    {
cube.rotation.x += 0.02;
cube.rotation.y += 0.02;
cube.rotation.z += 0.02;
trackballcontrols.update();
requestAnimationFrame( renderScene );
renderer.render( scene, camera );
}
};
</script>
</body>
</html>
```

ДОДАТОК С

Приклад 07-01 – Ambient Light

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 07.01 – Ambient Light </title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
<script src="./js/dat.gui.js"></script>
</head>
<body>
<script>
init();
	function init() {
	// створюємо рендер і задаємо розмір
	var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
	renderer.setClearColor(new THREE.Color(0xEEEEEE));
	renderer.setSize( window.innerWidth, window.innerHeight );
	// додаємо вивід візуалізатора до html-елементу
	document.body.appendChild( renderer.domElement );
	// створюємо камеру
	var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,100);
	// створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
	var scene = new THREE.Scene();
	var trackballcontrols = new THREE.TrackballControls( camera,
renderer.domElement );
	var planeGeometry = new THREE.PlaneGeometry(60,20,1,1);
	var plane_texture = new THREE.TextureLoader().load( ‘./images/textura-
dereva-s-vysokim-razresheniem-11-600x600.jpg’ );
	var plane_material = new THREE.MeshLambertMaterial({map:
plane_texture});
	var plane = new THREE.Mesh(planeGeometry,plane_material);
	plane.rotation.x=-0.5*Math.PI;
	plane.position.x = 15;
	plane.position.y = 0;
	plane.position.z = 0;
	scene.add(plane);
	var cube_geometry = new THREE.CubeGeometry(4, 4, 4);
```

```
        var cube_texture = new THREE.TextureLoader().load( ‘./images/1.jpg’ );
        var cube_material = new THREE.MeshLambertMaterial({map:
cube_texture});
        var cube = new THREE.Mesh(cube_geometry, cube_material);
        cube.position.x = -4;
        cube.position.y = 3;
        cube.position.z = 0;
        scene.add(cube);
        camera.position.x = -25;
        camera.position.y = 30;
        camera.position.z = 25;
        camera.lookAt(new THREE.Vector3(10, 0, 0));
        // додаємо освітлення ambient lighting
        // AmbientLight – це світло, яке освітлює всю сцену.
        // Воно не має напрямку і освітлює кожен об’єкт сцени в рівній мірі,
        // незалежно від розташування об’єкта.
        // Відповідно, у цього світла немає позиції на осі координат.
        // Цей світ не може використовуватися для створення тіней.
        var ambiColor = "#e3e1e1";
        var ambiIntensity=1;
        var ambientLight = new THREE.AmbientLight(ambiColor,ambiIntensity);
        scene.add(ambientLight);
        var controls = new function () {
        this.rotationSpeed = 0.02;
        this.AmbientLightColor = ambiColor;
        this.intensity = 1; };
        var gui = new dat.GUI();
        gui.addColor(controls, ‘AmbientLightColor’).onChange(function (e) {
        ambientLight.color = new THREE.Color(e);
        });
        gui.add(controls, ‘intensity’, 0, 3).onChange(function (e) {
        ambientLight.intensity = e;
        });
        renderScene();
        function renderScene() {
        cube.rotation.x += controls.rotationSpeed;
        cube.rotation.y += controls.rotationSpeed;
        cube.rotation.z += controls.rotationSpeed;
        requestAnimationFrame( renderScene );
        trackballcontrols.update();
        renderer.render( scene, camera ); }
};
</script>
</body>
</html>
```

ДОДАТОК Т

Приклад 07-02 – PointLight

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 07.02 – PointLight </title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
<script src="./js/dat.gui.js"></script>
</head>
<body>
<script>
init();
	function init() {
	// створюємо рендер і задаємо розмір
	var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
	renderer.setClearColor(new THREE.Color(0xEEEEEE));
	renderer.setSize( window.innerWidth, window.innerHeight );
	renderer.shadowMapEnabled = true;
	// додаємо вивід візуалізатора до html-елементу
	document.body.appendChild( renderer.domElement );
	// створюємо камеру
	var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,100);
	// створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
	var scene = new THREE.Scene();
	var trackballcontrols = new THREE.TrackballControls( camera,
renderer.domElement );
var planeGeometry = new THREE.PlaneGeometry(60,20,1,1);
	var plane_texture = new THREE.TextureLoader().load( ‘./images/textura-
dereva-s-vysokim-razresheniem-11-600x600.jpg’ );
	var plane_material = new THREE.MeshLambertMaterial({map:
plane_texture});
	var plane = new THREE.Mesh(planeGeometry,plane_material);
	plane.receiveShadow = true;
	plane.rotation.x=-0.5*Math.PI;
	plane.position.x = 15;
	plane.position.y = 0;
	plane.position.z = 0;
```

```
        scene.add(plane);
        var cube_geometry = new THREE.CubeGeometry(4, 4, 4);
        var cube_texture = new THREE.TextureLoader().load( ‘./images/1.jpg’ );
        var cube_material = new THREE.MeshLambertMaterial({map:
cube_texture});
        var cube = new THREE.Mesh(cube_geometry, cube_material);
        cube.castShadow = true;
        cube.position.x = -4;
        cube.position.y = 3;
        cube.position.z = 0;
        scene.add(cube);
        camera.position.x = -25;
        camera.position.y = 30;
        camera.position.z = 25;
        camera.lookAt(new THREE.Vector3(10, 0, 0));
        // додаємо освітлення ambient lighting
        // додаємо освітлення ambient lighting
        // AmbientLight – це світло, яке освітлює всю сцену.
        // Воно не має напрямку і освітлює кожен об’єкт сцени в рівній мірі,
        // незалежно від розташування об’єкта.
        // Відповідно, у цього світла немає позиції на осі координат.
        // Цей світ не може використовуватися для створення тіней.
        var ambiColor = "#e3e1e1";
        var ambientLight = new THREE.AmbientLight(ambiColor);
        scene.add(ambientLight);

        // PointLight – освітлення, що виходить з однієї точки в усіх напрямках
        // (це джерело освітлення можна порівняти зі звичайною лампочкою).
        var pointColor = "#ccffcc";
        var pointLight = new THREE.PointLight(pointColor);
        pointLight.distance = 100;
        scene.add(pointLight);
        // додаємо невелику сферу, яка імітує PointLight
        var sphereLight = new THREE.SphereGeometry(0.5);
        var sphereLightMaterial = new THREE.MeshBasicMaterial({color:
0xac6c25});
        var sphereLightMesh = new THREE.Mesh(sphereLight,
sphereLightMaterial);
        sphereLightMesh.castShadow = true;
        sphereLightMesh.position = new THREE.Vector3(3, 0, 3);
        scene.add(sphereLightMesh);
        var invert = 1;
        var phase = 0;
        var controls = new function () {
        this.rotationSpeed = 0.02;
```

```
        this.AmbientLightColor = ambiColor;
        this.pointColor = pointColor;
        this.intensity = 1;
        this.distance = 100;
        };
        var gui = new dat.GUI();
        gui.addColor(controls, ‘AmbientLightColor’).onChange(function (e) {
        ambientLight.color = new THREE.Color(e);
        });
        gui.addColor(controls, ‘pointColor’).onChange(function (e) {
        pointLight.color = new THREE.Color(e); });
        gui.add(controls, ‘intensity’, 0, 3).onChange(function (e) {
        pointLight.intensity = e;
        });
        gui.add(controls, ‘distance’, 0, 100).onChange(function (e) {
        pointLight.distance = e;
        });
        renderScene();
        function renderScene()
        {
        cube.rotation.x += controls.rotationSpeed;
        cube.rotation.y += controls.rotationSpeed;
        cube.rotation.z += controls.rotationSpeed;
         if (phase > 2 * Math.PI) {
        invert = invert * -1;
        phase -= 2 * Math.PI;
        } else {
        phase += controls.rotationSpeed;
        }
        sphereLightMesh.position.z = +(7 * (Math.sin(phase)));
        sphereLightMesh.position.x = +(14 * (Math.cos(phase)));
        sphereLightMesh.position.y = 5;
        if (invert < 0) {
        var pivot = 14;
        sphereLightMesh.position.x = (invert * (sphereLightMesh.position.x –
pivot)) + pivot; }
        pointLight.position.copy(sphereLightMesh.position);
        requestAnimationFrame( renderScene );
        trackballcontrols.update();
        renderer.render( scene, camera );
        }
};
</script>
</body>
</html>
```

ДОДАТОК У

Приклад 07-03 – DirectionalLight

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 07.03 – DirectionalLight </title>
<script src="./js/three.js"></script>
<script src="./js/TrackballControls.js"></script>
<script src="./js/dat.gui.js"></script>
</head>
<body>
<script>
	init();
	function init() {
	var stopMovingLight = false;
	// створюємо рендер і задаємо розмір
	var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
	renderer.setClearColor(new THREE.Color(0xEEEEEE));
	renderer.setSize( window.innerWidth, window.innerHeight );
	renderer.shadowMapEnabled = true;
	renderer.shadowMapSoft = true;
	renderer.shadowMapType = THREE.PCFShadowMap;
	// додаємо вивід візуалізатора до html-елементу
	document.body.appendChild( renderer.domElement );
	// створюємо камеру
	var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
	// створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
	var scene = new THREE.Scene();
	var trackballcontrols = new THREE.TrackballControls( camera,
renderer.domElement );
	var planeGeometry = new THREE.PlaneGeometry(60,20,1,1);
	var plane_texture = new THREE.TextureLoader().load( ‘./images/textura-
dereva-s-vysokim-razresheniem-11-600x600.jpg’ );
	var plane_material = new THREE.MeshLambertMaterial({map:
plane_texture});
	var plane = new THREE.Mesh(planeGeometry,plane_material);
	plane.receiveShadow = true;
	plane.rotation.x=-0.5*Math.PI;
```

```
        plane.position.x = 15;
        plane.position.y = -5;
        plane.position.z = 0;
        scene.add(plane);
        var cube_geometry = new THREE.CubeGeometry(4, 4, 4);
        var cube_texture = new THREE.TextureLoader().load( ‘./images/1.jpg’ );
        var cube_material = new THREE.MeshLambertMaterial({map:
cube_texture});
        var cube = new THREE.Mesh(cube_geometry, cube_material);
        cube.castShadow = true;
        cube.position.x = -4;
        cube.position.y = 3;
        cube.position.z = 0;
        scene.add(cube);
        camera.position.x = -25;
        camera.position.y = 30;
        camera.position.z = 25;
        camera.lookAt(new THREE.Vector3(10, 0, 0));
        // додаємо освітлення ambient lighting
        // AmbientLight – це світло, яке освітлює всю сцену.
        // Воно не має напрямку і освітлює кожен об’єкт сцени в рівній мірі,
        // незалежно від розташування об’єкта.
        // Відповідно, у цього світла немає позиції на осі координат.
        // Цей світ не може використовуватися для створення тіней.
        var ambiColor = "#e3e1e1";
        var ambientLight = new THREE.AmbientLight(ambiColor);
        scene.add(ambientLight);
        var target = new THREE.Object3D();
        target.position = new THREE.Vector3(5, 0, 0);
        // Джерело DirectionalLight можна розглядати як джерело світла, що
знаходиться дуже далеко.
        // Весь світ промені, які він випромінює, паралельні один одному.
        var pointColor = "#ff5808";
        var directionalLight = new THREE.DirectionalLight(pointColor);
        directionalLight.position.set(-40, 60, -10);
        directionalLight.castShadow = true;
        directionalLight.shadowCameraNear = 2;
        directionalLight.shadowCameraFar = 200;
        directionalLight.shadowCameraLeft = -50;
        directionalLight.shadowCameraRight = 50;
        directionalLight.shadowCameraTop = 50;
        directionalLight.shadowCameraBottom = -50;
        directionalLight.distance = 0;
        directionalLight.intensity = 0.5;
        directionalLight.shadowMapHeight = 1024;
```

```
directionalLight.shadowMapWidth = 1024;
scene.add(directionalLight);
// додаємо невелику сферу, яка імітує PointLight
var sphereLight = new THREE.SphereGeometry(0.5);
var sphereLightMaterial = new THREE.MeshBasicMaterial({color:
0xac6c25});
var sphereLightMesh = new THREE.Mesh(sphereLight,
sphereLightMaterial);
sphereLightMesh.castShadow = true;
sphereLightMesh.position = new THREE.Vector3(3, 20, 3);
scene.add(sphereLightMesh);
var step = 0;
var controls = new function () {
this.rotationSpeed = 0.03;
this.ambientColor = ambiColor;
this.pointColor = pointColor;
this.intensity = 0.5;
this.distance = 0;
this.exponent = 30;
this.angle = 0.1;
this.castShadow = true;
this.target = "Plane";
};
var gui = new dat.GUI();
gui.addColor(controls, ‘ambientColor’).onChange(function (e) {
ambientLight.color = new THREE.Color(e); });
gui.addColor(controls, ‘pointColor’).onChange(function (e) {
directionalLight.color = new THREE.Color(e); });
gui.add(controls, ‘intensity’, 0, 5).onChange(function (e) {
directionalLight.intensity = e; });
gui.add(controls, ‘distance’, 0, 200).onChange(function (e) {
directionalLight.distance = e; });
gui.add(controls, ‘castShadow’).onChange(function (e) {
directionalLight.castShadow = e; });
gui.add(controls, ‘target’, [‘Plane’,  ‘Cube’]).onChange(function (e) {
console.log(e);
switch (e) {
case "Plane":
directionalLight.target = plane;
break;
case "Cube":
directionalLight.target = cube;
break; }});
renderScene();
function renderScene() {
```

```
        cube.rotation.x += controls.rotationSpeed;
        cube.rotation.y += controls.rotationSpeed;
        cube.rotation.z += controls.rotationSpeed;
        step += controls.rotationSpeed;
        sphereLightMesh.position.z = -8;
        sphereLightMesh.position.y = +(27 * (Math.sin(step / 3)));
        sphereLightMesh.position.x = 10 + (26 * (Math.cos(step / 3)));
        directionalLight.position.copy(sphereLightMesh.position);
        requestAnimationFrame( renderScene );
        trackballcontrols.update();
        renderer.render( scene, camera ); }};
</script>
</body>
</html>
```

ДОДАТОК Ф

Приклад 08-01 – Відео в якості текстури

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 08.01 – Відео в якості текстури </title>
<script src="./js/three.js"></script>
<script src="./js/dat.gui.js"></script>
<script src="./js/TrackballControls.js"></script>
</head>
<body>
<video id="video"  loop
style="display:none; position: absolute; left: 15px; top: 75px;"
src="./movies/NewVideo.mp4">
</video>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        var video = document.getElementById('video');
        var videoTexture = new THREE.Texture( video );
        videoTexture.minFilter = THREE.LinearFilter;
        videoTexture.magFilter = THREE.LinearFilter;
        videoTexture.format = THREE.RGBFormat;
        videoTexture.generateMipmaps = false;
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об'єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var trackballcontrols = new THREE.TrackballControls( camera,
renderer.domElement );
        var cube;
        cube = createMesh(new THREE.BoxGeometry(22, 16, 0.2));
```

```
cube.position.y = 2;
scene.add(cube);
cube.position.y = 2;
cube.name = ‘cube’;
scene.add(cube);
// position and point the camera to the center of the scene
camera.position.x = 00;
camera.position.y = 1;
camera.position.z = 28;
camera.lookAt(new THREE.Vector3(0, 0, 0));
var ambiLight = new THREE.AmbientLight(0x141414);
scene.add(ambiLight);
var light = new THREE.DirectionalLight();
light.position.set(0, 30, 20);
scene.add(light);
 // setup the control gui
var controls = new function () {
this.showVideo = false;
this.rotate = false;
this.scale = 1;
this.showCanvas = function () {
if (controls.showVideo) {
video.play();
} else {
video.pause(); }}};
var gui = new dat.GUI();
gui.add(controls, "rotate");
gui.add(controls, "showVideo").onChange(controls.showCanvas);
gui.add(controls, ‘scale’, 0.01, 5);
renderScene();
function createMesh(geom) {
var materialArray = [];
materialArray.push(new THREE.MeshBasicMaterial({color: 0x0051ba}));
materialArray.push(new THREE.MeshBasicMaterial({color: 0x0051ba}));
materialArray.push(new THREE.MeshBasicMaterial({color: 0x0051ba}));
materialArray.push(new THREE.MeshBasicMaterial({color: 0x0051ba}));
materialArray.push(new THREE.MeshBasicMaterial({map:
videoTexture}));
materialArray.push(new THREE.MeshBasicMaterial({color: 0xff51ba}));
var mesh = new THREE.Mesh(geom, materialArray );
return mesh; }
function renderScene() {
 if (controls.rotate) {
cube.rotation.x += -0.01;
cube.rotation.y += -0.01;
```

```
        cube.rotation.z += -0.01; }
        scene.getObjectByName('cube').scale.set(controls.scale, controls.scale,
controls.scale);
        videoTexture.needsUpdate = true;
        requestAnimationFrame( renderScene );
        trackballcontrols.update();
        renderer.render( scene, camera ); }};
</script>
</body>
</html>
```

ДОДАТОК Х

Приклад 9-01 – Додавання 3D тексту

```
!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 9.01 – Додавання 3D тексту</title>
<script src="./js/three.js"></script>
<script src="./js/TrackballControls.js"></script>
</head>
<body>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var controls = new THREE.TrackballControls( camera );
        var ambiLight = new THREE.AmbientLight(0x141414);
        scene.add(ambiLight);
        var light = new THREE.DirectionalLight();
        light.position.set(0, 80, 200);
        scene.add(light);
        var fontLoader = new THREE.FontLoader();
        fontLoader.load("./fonts/Academia Libera_Regular.json",function(tex){
        var  text_Geometry = new THREE.TextGeometry("Комп’ютерна графіка",
{ size: 20, height: 3,
        curveSegments: 6,
        bevelEnabled: true,
        bevelThickness: 2,
        bevelSize: 1,
        font: tex});
```

```
        var  text_Material = new THREE.MeshBasicMaterial({ color: 0x62254a  });
        var  text = new THREE.Mesh(text_Geometry , text_Material);
        text_Geometry.center();
        scene.add(text);})
        camera.position.x ==-50;
        camera.position.y = 70;
        camera.position.z = 200;
        camera.lookAt(scene.position);
        renderScene();
        function renderScene() {
        requestAnimationFrame( renderScene );
        controls.update();
        renderer.render( scene, camera );}};
</script>
</body>
</html>
```

ДОДАТОК Ц

Приклад 9-02 – 3D текст з текстурою

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 9.02 – 3D текст з текстурою </title>
<script src="./js/three.js"></script>
<script src="./js/TrackballControls.js"></script>
</head>
<body>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var controls = new THREE.TrackballControls( camera );
        var ambiLight = new THREE.AmbientLight(0x141414);
        scene.add(ambiLight);
        var light = new THREE.DirectionalLight();
        light.position.set(0, 80, 200);
        scene.add(light);
        var fontLoader = new THREE.FontLoader();
        fontLoader.load("./fonts/Academia Libera_Regular.json",function(tex){
        var  text_Geometry = new THREE.TextGeometry("Комп’ютерна графіка",
{
        size: 20,
        height: 3,
        curveSegments: 6,
        bevelEnabled: true,
        bevelThickness: 2,
```

```
        bevelSize: 1,
        material: 0,
        extrudeMaterial: 1,
        font: tex});
        var Texture = THREE.ImageUtils.loadTexture( ‘images/fire.jpg’ );
        Texture.wrapS = Texture.wrapT = THREE.RepeatWrapping;
        Texture.repeat.set( 0.05, 0.05 );
        var text_Material =  new THREE.MeshBasicMaterial( { map: Texture } );
        var  text = new THREE.Mesh(text_Geometry , text_Material);
        text_Geometry.center();
        scene.add(text);
})
        camera.position.x ==-50;
        camera.position.y = 70;
        camera.position.z = 200;
        camera.lookAt(scene.position);
        renderScene();
        function renderScene() {
        requestAnimationFrame( renderScene );
        controls.update();
        renderer.render( scene, camera ); }
};
</script>
</body>
</html>
```

ДОДАТОК Ш

Приклад 9-03 – 3D текст з використанням двох текстур

```
<!DOCTYPE html>
<html>
   <head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 9.03 –  3D текст з використанням двох текстур </title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
</head>
<body>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var controls = new THREE.TrackballControls( camera );
        var ambiLight = new THREE.AmbientLight(0x141414);
        scene.add(ambiLight);
        var light = new THREE.DirectionalLight();
        light.position.set(0, 80, 200);
        scene.add(light);
        var fontLoader = new THREE.FontLoader();
        fontLoader.load("./fonts/Academia Libera_Regular.json",function(tex){
        var  text_Geometry = new THREE.TextGeometry("Комп’ютерна графіка",
{
        size: 20,
        height: 3,
        curveSegments: 6,
        bevelEnabled: true,
        bevelThickness: 2,
```

```
        bevelSize: 1,
        material: 0,
        extrudeMaterial: 1,
        font: tex});
        var Texture = THREE.ImageUtils.loadTexture( ‘images/fire.jpg’ );
        Texture.wrapS = Texture.wrapT = THREE.RepeatWrapping;
        Texture.repeat.set( 0.05, 0.05 );
        var Material = new THREE.MeshBasicMaterial( { map: Texture } );
        var ExtrTexture =  THREE.ImageUtils.loadTexture( ‘images/sky.jpg’ );
        ExtrTexture.wrapS = ExtrTexture.wrapT  = THREE.RepeatWrapping;
        ExtrTexture.offset.set( 0, 0.7 );
        ExtrTexture.repeat.set( 0, 0.13 );
        var extrMaterial =  new THREE.MeshBasicMaterial( { map: ExtrTexture }
);
        var materials = [ Material, extrMaterial ];
        var text = new THREE.Mesh(text_Geometry, materials );
        text_Geometry.center();
        scene.add(text);
        })
        camera.position.x ==-50;
        camera.position.y = 70;
        camera.position.z = 200;
        camera.lookAt(scene.position);
        renderScene();
        function renderScene() {
        requestAnimationFrame( renderScene );
        controls.update();
        renderer.render( scene, camera );
        }
};
</script>
</body>
</html>
```

ДОДАТОК Щ

Приклад 10-01 – Таймер Clock

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 10.01 – Таймер Clock </title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
</head>
<body>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var controls = new THREE.TrackballControls( camera );
        var clock = new THREE.Clock( );
        var period = 1;
        var ambiLight = new THREE.AmbientLight(0x141414);
        scene.add(ambiLight);
        var light = new THREE.DirectionalLight();
        light.position.set(0, 30, 20);
        scene.add(light);
        var axes = new THREE.AxisHelper( 20 );
        scene.add(axes);
        var planeGeometry = new THREE.PlaneGeometry(60,20,1,1);
        var planeMaterial = new THREE.MeshBasicMaterial({color: 0xcccccc});
        var plane = new THREE.Mesh(planeGeometry,planeMaterial);
        plane.rotation.x=-0.5*Math.PI;
        plane.position.x = 15;
```

```
        plane.position.y = 0;
        plane.position.z = 0;
        scene.add(plane);
        var cubeGeometry = new THREE.CubeGeometry(4,4,4);
        var cubeMaterial =  new THREE.MeshPhongMaterial( { color: 0x00ff00 } );
        var cube = new THREE.Mesh(cubeGeometry, cubeMaterial);
        cube.position.x = -4;
        cube.position.y = 3;
        cube.position.z = 0;
        scene.add(cube);
        camera.position.x = -30;
        camera.position.y = 40;
        camera.position.z = 30;
        camera.lookAt(scene.position);
        clock.start();

        renderScene();
        function renderScene() {
        cube.rotation.x += 0.02;
        cube.rotation.y += 0.02;
        cube.rotation.z += 0.02;
        if ( clock.running )
        {
        var time = clock.getElapsedTime();
        if ( time > period )
        {
        cubeMaterial.color.setHex( Math.random() * 0xffffff );
        clock.elapsedTime = 0;// обнуляем счетчик таймера
        }
        }
        requestAnimationFrame( renderScene );
        controls.update();
        renderer.render( scene, camera );
        }
};
</script>
</body>
</html>
```

ДОДАТОК Ю

Приклад 11-01 – Створення структурних об’єктів

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 11.01 – Створення структурних об’єктів </title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
</head>
<body>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xffffff));
        renderer.setSize( window.innerWidth, window.innerHeight );
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var controls = new THREE.TrackballControls( camera );
        camera.position.z = 500;
        camera.position.y = 150;
        scene.add(camera);
        var ambientLight = new THREE.AmbientLight(0xffffff);
        scene.add(ambientLight);
        var kapitoshka = new THREE.Object3D();
        var material = new THREE.MeshBasicMaterial({color: 0xADD8E6});
        var geometry = new THREE.SphereGeometry(70, 36, 36);
        var sphere1 = new THREE.Mesh(geometry, material);
        sphere1.position.set(0, 60, 0);
        kapitoshka.add(sphere1);
        material = new THREE.MeshBasicMaterial({color: 0x00008B});
        geometry = new THREE.SphereGeometry(45, 36, 36);
        var sphere2 = new THREE.Mesh(geometry, material);
```

```
sphere2.position.set(0, 115, 0);
kapitoshka.add(sphere2);
material = new THREE.MeshBasicMaterial({color: 0x8B008B});
geometry = new THREE.SphereGeometry(12, 16, 16);
var sphere3 = new THREE.Mesh(geometry, material);
sphere3.position.set(0, 166, 0);
kapitoshka.add(sphere3);

material = new THREE.MeshBasicMaterial({color: 0x00008B});
geometry = new THREE.SphereGeometry(16, 16, 16);
var sphere4 = new THREE.Mesh(geometry, material);
sphere4.position.set(-85, 60, 0);
kapitoshka.add(sphere4);

material = new THREE.MeshBasicMaterial({color: 0x8B008B});
geometry = new THREE.SphereGeometry(16, 16, 16);
var sphere5 = new THREE.Mesh(geometry, material);
sphere5.position.set(84, 75, 0);
kapitoshka.add(sphere5);

material = new THREE.MeshBasicMaterial({color: 0x8B008B});
geometry = new THREE.SphereGeometry(13, 16, 16);
var sphere6 = new THREE.Mesh(geometry, material);
sphere6.position.set(-40, -10, 0);
kapitoshka.add(sphere6);

material = new THREE.MeshBasicMaterial({color: 0x00008B});
geometry = new THREE.SphereGeometry(14, 16, 16);
var sphere7 = new THREE.Mesh(geometry, material);
sphere7.position.set(58, 0, 0);
kapitoshka.add(sphere7);

material = new THREE.MeshBasicMaterial({color: 0xFFFF00});
geometry = new THREE.SphereGeometry(11, 16, 16);
var finger1 = new THREE.Mesh(geometry, material);
finger1.position.set(-109, 70, 0);
kapitoshka.add(finger1);

material = new THREE.MeshBasicMaterial({color: 0xFFFF00});
geometry = new THREE.SphereGeometry(10, 16, 16);
var finger2 = new THREE.Mesh(geometry, material);
finger2.position.set(102, 56, 0);
kapitoshka.add(finger2);

material = new THREE.MeshBasicMaterial({color: 0xFFFF00});
```

```
        geometry = new THREE.SphereGeometry(10, 16, 16);
        var finger3 = new THREE.Mesh(geometry, material);
        finger3.position.set(-58, -22, 0);
        kapitoshka.add(finger3);

        material = new THREE.MeshBasicMaterial({color: 0xFFFF00});
        geometry = new THREE.SphereGeometry(11, 16, 16);
        var finger4 = new THREE.Mesh(geometry, material);
        finger4.position.set(78, -16, 0);
        kapitoshka.add(finger4);

        material = new THREE.MeshLambertMaterial({color: 0x000000});
        geometry = new THREE.SphereGeometry(5, 50, 11);
        var eye1 = new THREE.Mesh(geometry, material);
        eye1.position.set(-15, 80, 70);
        kapitoshka.add(eye1);

        material = new THREE.MeshLambertMaterial({color: 0x000000});
        geometry = new THREE.SphereGeometry(5, 50, 11);
        var eye2 = new THREE.Mesh(geometry, material);
        eye2.position.set(15, 80, 70);
        kapitoshka.add(eye2);

        material = new THREE.MeshBasicMaterial({color: 0xFA8072});
        geometry = new THREE.CircleGeometry(15, 10, 0, Math.PI);
        var mouse = new THREE.Mesh(geometry, material);
        mouse.position.set(0, 50, 68);
        mouse.rotation.z = Math.PI;
        kapitoshka.add(mouse);
        kapitoshka.position.set(0, 0, 0);
        scene.add(kapitoshka);
        renderScene();
        function renderScene() {
        kapitoshka.rotation.y += 0.02;
        kapitoshka.rotation.z += 0.02;
        requestAnimationFrame( renderScene );
        controls.update();
        renderer.render( scene, camera );
        }
};
</script>
</body>
</html>
```

ДОДАТОК Я

Приклад 12-01 – Побудова плоского кола

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 12.01 –  Побудова плоского кола </title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
</head>
<body>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        renderer.shadowMapEnabled = true;
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var controls = new THREE.TrackballControls( camera );
        var ambientLight = new THREE.AmbientLight(0xffffff);
        scene.add(ambientLight);
        var spotLight = new THREE.SpotLight( 0xffffff );
        spotLight.position.set( -80,120, -20 );
        spotLight.castShadow = true;
        scene.add( spotLight );
        var circle_geometry = new THREE.CircleGeometry( 6, 100, 0, 2*Math.PI );
        var circle_texture = new THREE.TextureLoader().load( ‘images/logo.jpg’
);//load texture
        var circle_material = new THREE.MeshLambertMaterial({map:
circle_texture});
        // circle_material.side = THREE.DoubleSide;
        var circle = new THREE.Mesh(circle_geometry, circle_material);
```

```
        circle.castShadow = true;
        circle.position.x=-60; circle.position.y=45;
        scene.add(circle);
        camera.position.x = -50;  camera.position.y = 70;
        camera.position.z = 100;
        camera.lookAt(scene.position);
        renderScene();
        function renderScene() {
        circle.rotation.x += 0.02;
        circle.rotation.z += 0.02;
        requestAnimationFrame( renderScene );
        controls.update();
        renderer.render( scene, camera );}};
</script>
</body>
</html>
```

ДОДАТОК 1

Приклад 13-01 – Створення анімації за допомогою TWEEN.JS

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 13.01 – Створення анімації за допомогою TWEEN.JS </title>
<script src="./js/three.js"></script>
<script src="./js/TrackballControls.js"></script>
<script src="./js/tween.js"></script>
</head>
<body>
<script>
init();
	function init() {
	// створюємо рендер і задаємо розмір
	var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
	renderer.setClearColor(new THREE.Color(0xEEEEEE));
	renderer.setSize( window.innerWidth, window.innerHeight );
	renderer.shadowMapEnabled = true;
	// додаємо вивід візуалізатора до html-елементу
	document.body.appendChild( renderer.domElement );
	// створюємо камеру
	var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
	// створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
	var scene = new THREE.Scene();
	var controls = new THREE.TrackballControls( camera );
	var ambientLight = new THREE.AmbientLight(0xffffff);
	scene.add(ambientLight);
	var spotLight = new THREE.SpotLight( 0xffffff );
	spotLight.position.set( -80,120, -20 );
	spotLight.castShadow = true;
	scene.add( spotLight );
	var circle_position = { x : -300, y: 0 };
	var circle_target = {};
	circle_target.x = 0;
	circle_target.y = 45;
	var circle_geometry = new THREE.CircleGeometry( 6, 100, 0, 2*Math.PI );
```

```
        var circle_texture = new THREE.TextureLoader().load( ‘images/logo.jpg’ );//load texture
        var circle_material = new THREE.MeshLambertMaterial({map: circle_texture});
        circle_material.side = THREE.DoubleSide;
        var circle = new THREE.Mesh(circle_geometry, circle_material);
        circle.castShadow = true;
        circle.position.x=-300;
        circle.position.y=0;
        scene.add(circle);
        var circle_tween = new TWEEN.Tween(circle_position).to(circle_target, 4000);
        circle_tween.start();
        circle_tween.onUpdate(function(){
        circle.position.x = circle_position.x;
        circle.position.y =circle_position.y;
        circle.rotation.x += 0.02;
        circle.rotation.y += 0.02;
        circle.rotation.z += 0.02;
        });
        camera.position.x = -50;
        camera.position.y = 70;
        camera.position.z = 100;
        camera.lookAt(scene.position);
        renderScene();
        function renderScene() {
        TWEEN.update();
        requestAnimationFrame( renderScene );
        controls.update();
        renderer.render( scene, camera );
        }
};
</script>
</body>
</html>
```

ДОДАТОК 2

Приклад 13-02 – Зміна функції плавності TWEEN.JS

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 13.02 – Зміна функції плавності TWEEN.JS </title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
 <script src="./js/tween.js"></script>
</head>
<body>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        renderer.shadowMapEnabled = true;
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var controls = new THREE.TrackballControls( camera );
        var ambientLight = new THREE.AmbientLight(0xffffff);
        scene.add(ambientLight);
        var spotLight = new THREE.SpotLight( 0xffffff );
        spotLight.position.set( -80,120, -20 );
        spotLight.castShadow = true;
        scene.add( spotLight );
        var circle_position = { x : -90, y: 0 };
        var circle_target = {};
        circle_target.x = 0;
        circle_target.y = 45;
        var circle_geometry = new THREE.CircleGeometry( 6, 100, 0, 2*Math.PI );
```

```
        var circle_texture = new THREE.TextureLoader().load( ‘images/logo.jpg’ );//load texture
        var circle_material = new THREE.MeshLambertMaterial({map: circle_texture});
        circle_material.side = THREE.DoubleSide;
        var circle = new THREE.Mesh(circle_geometry, circle_material);
        circle.castShadow = true;
        circle.position.x=-90;
        circle.position.y=0;
        scene.add(circle);
        var circle_tween = new TWEEN.Tween(circle_position).to(circle_target, 4000);
        circle_tween.easing(TWEEN.Easing.Quadratic.In);
        //circle_tween.easing(TWEEN.Easing.Quadratic.Out);
        circle_tween.start();
        circle_tween.onUpdate(function(){
        circle.position.x = circle_position.x;
        circle.position.y =circle_position.y;
        circle.rotation.x += 0.02;
        circle.rotation.y += 0.02;
        circle.rotation.z += 0.02;
        });
        camera.position.x = -50;
        camera.position.y = 70;
        camera.position.z = 100;
        camera.lookAt(scene.position);
        renderScene();
        function renderScene() {
        TWEEN.update();
        requestAnimationFrame( renderScene );
        controls.update();
        renderer.render( scene, camera );
        }};
</script>
</body>
</html>
```

ДОДАТОК 3

Приклад 13-03 – Tween.js

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 13.03 –  Tween.js </title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
<script src="./js/tween.js"></script>
</head>
<body>
<video id="video"  loop autoplay="autoplay"
style="display:none; position: absolute; left: 15px; top: 75px;"
src="./movies/Fire.mp4">
</video>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        renderer.shadowMapEnabled = true;
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        var video = document.getElementById('video');
        var videoTexture = new THREE.Texture( video );
        videoTexture.minFilter = THREE.LinearFilter;
        videoTexture.magFilter = THREE.LinearFilter;
        videoTexture.format = THREE.RGBFormat;
        videoTexture.generateMipmaps = false;
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об'єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var controls = new THREE.TrackballControls( camera );
        var ambientLight = new THREE.AmbientLight(0xffffff);
        scene.add(ambientLight);
```

```
var spotLight = new THREE.SpotLight( 0xffffff );
spotLight.position.set( -80,120, -20 );
spotLight.castShadow = true;
scene.add( spotLight );
var fontLoader = new THREE.FontLoader();
fontLoader.load("./fonts/Academia Libera_Regular.json",function(tex){
var  text_Geometry = new THREE.TextGeometry("Комп’ютерна графіка",
{
size: 10,
height: 3,
curveSegments: 6,
bevelEnabled: true,
bevelThickness: 2,
bevelSize: 1,
material: 0,
extrudeMaterial: 1,
font: tex});
var text_position = { x : -300, y: 100 };
var circle_position = { x : -300, y: 0 };
var plane_position = {x:0,y:300};
var sphere_position = {x:300,y:0,z:0};
var plane_target = {};
plane_target.x = 25;
plane_target.y = 0;
var plane_geometry = new THREE.PlaneGeometry(120,40,1,1);
var  plane_material = new THREE.MeshBasicMaterial({map:
videoTexture});
var plane = new THREE.Mesh(plane_geometry,plane_material);
plane.rotation.x=-0.5*Math.PI;
plane.position.x = 0;
plane.position.y = 300;
scene.add(plane);
var plane_tween = new TWEEN.Tween(plane_position).to(plane_target,
2000);
plane_tween.start();
plane_tween.onUpdate(function(){
plane.position.x = plane_position.x;
plane.position.y = plane_position.y;
});
plane_tween.onComplete(function() {
var plane_tween_1 = new TWEEN.Tween(plane_position).to(plane_target,
2000);
plane_tween_1.start();
plane_tween_1.onUpdate(function(){
video.play();
```

```
        });
        });
        var circle_target = {};
        circle_target.x = -60;
        circle_target.y = 45;
        var circle_geometry = new THREE.CircleGeometry( 6, 100, 0, 2*Math.PI );
        var circle_texture = new THREE.TextureLoader().load( ‘images/logo.jpg’
);//load texture
        var circle_material = new THREE.MeshLambertMaterial({map:
circle_texture});
        circle_material.side = THREE.DoubleSide;
        var circle = new THREE.Mesh(circle_geometry, circle_material);
        circle.castShadow = true;
        circle.position.x=-300;
        circle.position.y=0;
        scene.add(circle);
        var circle_tween = new TWEEN.Tween(circle_position).to(circle_target,
2000);
        circle_tween.start();
        circle_tween.onUpdate(function(){
        circle.position.x = circle_position.x;
        circle.position.y =circle_position.y;
        circle.rotation.x += 0.02;
        circle.rotation.y += 0.02;
        circle.rotation.z += 0.02;
        });
        circle_tween.onComplete(function() {
        var circle_tween_1 = new TWEEN.Tween(circle_position).to(circle_target,
2000);
        circle_tween_1.repeat(Infinity);
        circle_tween_1.start();
        circle_tween_1.onUpdate(function(){
        circle.rotation.x += 0.02;
        circle.rotation.y += 0.02;
        circle.rotation.z += 0.02;
        });
        });
        var step=0;
        var sphere_target = {};
        sphere_target.x = 30;
        sphere_target.y = 8;
        sphere_target.z = 12;
        var sphereGeometry = new THREE.SphereGeometry(8,20,20);
        var sphere_texture = new THREE.TextureLoader().load(
‘images/sphere.jpg’ );//load texture
```

```
        var sphereMaterial = new THREE.MeshLambertMaterial({map:
sphere_texture});
        var sphere = new THREE.Mesh(sphereGeometry,sphereMaterial);
        sphere.castShadow = true;
        sphere.position.x = 300;
        sphere.position.y = 0;
        sphere.position.z = 0;
        scene.add(sphere);
        var sphere_tween = new TWEEN.Tween(sphere_position).to(sphere_target,
2000);
        sphere_tween.start();
        sphere_tween.onUpdate(function(){
        sphere.position.x = sphere_position.x;
        sphere.position.y = sphere_position.y;
        sphere.position.z = sphere_position.z;
        });
        sphere_tween.onComplete(function() {
        var sphere_tween_1 = new
TWEEN.Tween(sphere_position).to(sphere_target, 2000);
        sphere_tween_1.repeat(Infinity);
        sphere_tween_1.start();
        sphere_tween_1.onUpdate(function(){
        step+=0.05;
        sphere.position.x = 30+( 10*(Math.cos(step)));
        sphere.position.y = 8 +( 10*Math.abs(Math.sin(step)));
        sphere.position.z = 12;
        sphere.rotation.x += 0.02;
        sphere.rotation.y += 0.02;
        sphere.rotation.z += 0.02;
        });
        });
        var Texture = THREE.ImageUtils.loadTexture( 'images/fire.jpg' );
        Texture.wrapS = Texture.wrapT = THREE.RepeatWrapping;
        Texture.repeat.set( 0.05, 0.05 );
        var Material = new THREE.MeshBasicMaterial( { map: Texture } );
        var ExtrTexture =  THREE.ImageUtils.loadTexture( 'images/sky.jpg' );
        ExtrTexture.wrapS = ExtrTexture.wrapT  = THREE.RepeatWrapping;
        ExtrTexture.offset.set( 0, 0.7 );
        ExtrTexture.repeat.set( 0, 0.13 );
        var extrMaterial =  new THREE.MeshBasicMaterial( { map: ExtrTexture }
);
        var materials = [ Material, extrMaterial ];
        var text = new THREE.Mesh(text_Geometry, materials );
        var text_target = {};
        text_target.x = -50;
```

```
        text_target.y = 30;
        text.position.x=-300;
        text.position.y=100;
        scene.add(text);
        var text_tween = new TWEEN.Tween(text_position).to(text_target, 2000);
        text_tween.start();
        text_tween.onUpdate(function(){
        text.position.x = text_position.x;
        text.position.y = text_position.y;
        plane.position.x = plane_position.x;
        plane.position.y = plane_position.y;
        });
        })
        camera.position.x = -50;
        camera.position.y = 70;
        camera.position.z = 100;
        camera.lookAt(scene.position);
        renderScene();
        //      tween.easing(TWEEN.Easing.Bounce.InOut);
        function renderScene() {
        videoTexture.needsUpdate = true;
        requestAnimationFrame( renderScene );
        controls.update();
        TWEEN.update();
        renderer.render( scene, camera );
        }
};
</script>
</body>
</html>
```

ДОДАТОК 4

Приклад 14-01 – Логічні операції над моделлю з використанням THREECSG

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 14.01 – ЛОГІЧНІ ОПЕРАЦІЇ НАД МОДЕЛЛЮ З
ВИКОРИСТАННЯМ THREECSG</title>
<script src="./js/three.js"></script>
<script src="./js/TrackballControls.js"></script>
</head>
<body>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var controls = new THREE.TrackballControls( camera,
renderer.domElement );
        // Створюємо геометрію
        var cylinderGeometry = new
THREE.CylinderGeometry(100,100,20,100,20);
        var boxGeometry = new THREE.BoxGeometry(100,100,100,50,50);
        // Створюємо матеріал
        var materials = new THREE.MeshNormalMaterial({wireframe:true});
        // Створюємо  Mesh
        var cylinder = new THREE.Mesh(cylinderGeometry,materials);
        var box = new THREE.Mesh(boxGeometry, materials);
        scene.add(box);
        scene.add(cylinder);
```

```
        camera.position.x = 0;
        camera.position.y = 50;
        camera.position.z = 300;
        camera.lookAt(scene.position);
        renderScene();
        function renderScene() {
        requestAnimationFrame( renderScene );
        controls.update();
        renderer.render( scene, camera );
        }
};
</script>
</body>
</html>
```

ДОДАТОК 5

Приклад 14-02 – Приклад знаходження точки перетину циліндра та куба

```
<!DOCTYPE html>
<html>
<head>
meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 14.02 – Приклад знаходження точки перетину циліндра та куба </title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
 <script src="./js/ThreeCSG.js"></script>
</head>
<body>
<script>
        init();
        function init() {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var controls = new THREE.TrackballControls( camera,
renderer.domElement );
        // Світло
        var light = new THREE.PointLight(0xffffff);
        light.position.set(0,250,0);
        scene.add(light);
        // Створюємо геометрію
        var cylinderGeometry = new
THREE.CylinderGeometry(100,100,20,100,20);
        var boxGeometry = new THREE.BoxGeometry(100,100,100,50,50);
        // Створюємо матеріал
        var materials = new THREE.MeshNormalMaterial({wireframe:true});
        var materials1 = new THREE.MeshNormalMaterial({wireframe:false});
```

```
        // Створюємо Mesh
        var cylinder = new THREE.Mesh(cylinderGeometry,materials);
        var box = new THREE.Mesh(boxGeometry, materials);
        //scene.add( box);
        //scene.add( cylinder );
        // Об’єкт перетворення
        var cylinderBSP = new ThreeBSP(cylinder);
        var boxBSP = new ThreeBSP(box);
        // Перетин двох об’єктів
        var newBSP = cylinderBSP.intersect(boxBSP);
        // Перетворити об’єкт BSP на об’єкт Mesh
        var newMesh = newBSP.toMesh(materials1);
        scene.add( newMesh );
        camera.position.x = 0;
        camera.position.y = 200;
        camera.position.z = 300;
        camera.lookAt(scene.position);
        renderScene();
        function renderScene() {
        requestAnimationFrame( renderScene );
        controls.update();
        renderer.render( scene, camera );
        }
};
</script>
</body>
</html>
```

ДОДАТОК 6

Приклад 14-03 – Приклад знаходження об’єднання циліндра та куба (union)

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 14.03 – Приклад знаходження об’єднання циліндра та куба (union)</title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
 <script src="./js/ThreeCSG.js"></script>
</head>
<body>
<script>
        init();
        function init()
        {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth / window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var controls = new THREE.TrackballControls( camera, renderer.domElement );
        // Світло
        var light = new THREE.PointLight(0xffffff);
        light.position.set(0,250,0);
        scene.add(light);
        // Створюємо геометрію
        var cylinderGeometry = new THREE.CylinderGeometry(100,100,20,100,20);
        var boxGeometry = new THREE.BoxGeometry(100,100,100,50,50);
        // Створюємо матеріал
```

```
        var materials = new THREE.MeshNormalMaterial({wireframe:true});
        var materials1 = new THREE.MeshNormalMaterial({wireframe:false});
        // Створюємо Mesh
        var cylinder = new THREE.Mesh(cylinderGeometry,materials);
        var box = new THREE.Mesh(boxGeometry, materials);
        // Об’єкт перетворення
        var cylinderBSP = new ThreeBSP(cylinder);
        var boxBSP = new ThreeBSP(box);
        // Об’єднання двох об’єктів BSP
        var newBSP = cylinderBSP.union(boxBSP);
        // Перетворити об’єкт BSP на об’єкт Mesh
        var newMesh = newBSP.toMesh(materials1);
        scene.add( newMesh );
        camera.position.x = 0;
        camera.position.y = 200;
        camera.position.z = 300;
        camera.lookAt(scene.position);
        renderScene();
        function renderScene() {
        requestAnimationFrame( renderScene );
        controls.update();
        renderer.render( scene, camera );
        }
};
</script>
</body>
</html>
```

ДОДАТОК 7

Приклад 14-04 – Приклад різниці між циліндром та кубом (subtract)

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 14.04 – Приклад різниці між циліндром та кубом
(subtract)</title>
<script src="./js/three.js"></script>
 <script src="./js/TrackballControls.js"></script>
 <script src="./js/ThreeCSG.js"></script>
</head>
<body>
<script>
        init();
        function init()
        {
        // створюємо рендер і задаємо розмір
        var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE));
        renderer.setSize( window.innerWidth, window.innerHeight );
        // додаємо вивід візуалізатора до html-елементу
        document.body.appendChild( renderer.domElement );
        // створюємо камеру
        var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
        // створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
        var scene = new THREE.Scene();
        var controls = new THREE.TrackballControls( camera,
renderer.domElement );
        // Світло
        var light = new THREE.PointLight(0xffffff);
        light.position.set(0,250,0);
        scene.add(light);
        // Створюємо геометрію
        var cylinderGeometry = new
THREE.CylinderGeometry(100,100,20,100,20);
        var boxGeometry = new THREE.BoxGeometry(100,100,100,50,50);
        // Створюємо матеріал
        var materials = new THREE.MeshNormalMaterial({wireframe:true});
```

```
        var materials1 = new THREE.MeshNormalMaterial({wireframe:false});
        // Створюємо Mesh
        var cylinder = new THREE.Mesh(cylinderGeometry,materials);
        var box = new THREE.Mesh(boxGeometry, materials);
        // Об’єкт перетворення
        var cylinderBSP = new ThreeBSP(cylinder);
        var boxBSP = new ThreeBSP(box);
        // Різниця між двома об’єктами BSP
        var newBSP = cylinderBSP.subtract(boxBSP);
        // Перетворити об’єкт BSP на об’єкт Mesh
        var newMesh = newBSP.toMesh(materials1);
        scene.add( newMesh );
        camera.position.x = 0;
        camera.position.y = 200;
        camera.position.z = 300;
        camera.lookAt(scene.position);
        renderScene();
        function renderScene() {
        requestAnimationFrame( renderScene );
        controls.update();
        renderer.render( scene, camera );
        }
};
</script>
</body>
</html>
```

ДОДАТОК 8

Приклад 15-01 – Плоский шестикутник

```
<html>
<head>
<title> Приклад 15-01 – Плоский шестикутник </title>
<style>
body{
margin: 0;
overflow: hidden;
}
</style>
</head>
<body>
<script src="js/three.js"></script>
<script src="js/OrbitControls.js"></script>
<script>
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 20, window.innerWidth /
window.innerHeight, 0.1, 10000 );
camera.position.set( 0, 0, 140 );
var renderer = new THREE.WebGLRenderer({antialias: true});
renderer.setSize( window.innerWidth, window.innerHeight );
renderer.setClearColor(0x7b08f5);
document.body.appendChild( renderer.domElement );
var controls = new THREE.OrbitControls( camera, renderer.domElement );
var dirLight = new THREE.DirectionalLight( 0xffffff );
dirLight.position.set( -10, 10, 10 );
scene.add( dirLight );
var light = new THREE.DirectionalLight( 0xffffff );
light.position.set( 10, 10, -10 );
scene.add( light );
var geometry = new THREE.Geometry();
var material = new THREE.LineBasicMaterial( { color: 0xffffff } );
for (var i = 0; i <= 6; i++){
        var a = new THREE.Vector3(20*Math.cos((Math.PI/3)*i),
20*Math.sin((Math.PI/3)*i), 0);
        geometry.vertices.push(a);
}
var line = new THREE.Line(geometry, material);
scene.add(line);
render();
function render() {
```

```
controls.update();
requestAnimationFrame(render);
renderer.render( scene, camera );
}
</script>
</body>
</html>
```

ДОДАТОК 9

Приклад 15-02 – Пряма та площина

```
<html>
<head>
<title> Приклад 15-02 – Пряма та площина </title>
<style>
body{
margin: 0;
overflow: hidden;
}
</style>
</head>
<body>
<script src="js/three.js"></script>
<script src="js/OrbitControls.js"></script>
<script>
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 70, window.innerWidth /
window.innerHeight, 0.1, 10000 );
camera.position.set( 100, 30, 140 );
var renderer = new THREE.WebGLRenderer({antialias: true});
renderer.setSize( window.innerWidth, window.innerHeight );
renderer.setClearColor(0x7b08f5);
document.body.appendChild( renderer.domElement );
var controls = new THREE.OrbitControls( camera, renderer.domElement );
var dirLight = new THREE.DirectionalLight( 0xffffff );
dirLight.position.set( -10, 10, 10 );
scene.add( dirLight );
var light = new THREE.DirectionalLight( 0xffffff );
light.position.set( 10, 10, -10 );
scene.add( light );
var material = new THREE.MeshBasicMaterial({ wireframe: true, color:
0x21f344, side: THREE.DoubleSide });
var plane = new THREE.PlaneGeometry(80, 60, 1, 1);
var pl = new THREE.Mesh(plane, material);
pl.rotation.x = Math.PI/2;
pl.position.set(50, 0, 50);
scene.add(pl);
var lineGeom = new THREE.Geometry();
var a = new THREE.Vector3(50, 60, 50);
var b = new THREE.Vector3(50, 0, 50);
lineGeom.vertices.push(a, b);
```

```
var material = new THREE.LineBasicMaterial({ color: 0x0000ff });
var line = new THREE.Line(lineGeom, material);
scene.add(line);
var lineGeom = new THREE.Geometry();
var a = new THREE.Vector3(50, 0, 50);
var b = new THREE.Vector3(50, -60, 50);
lineGeom.vertices.push(a, b);
lineGeom.computeLineDistances();
var material = new THREE.LineDashedMaterial({ color: 0x0000ff, dashSize: 2,
gapSize: 1 });
var line = new THREE.Line(lineGeom, material);
line.computeLineDistances();
scene.add(line);
render();
function render() {
controls.update();
requestAnimationFrame(render);
renderer.render( scene, camera );
}
</script>
</body>
</html>
```

ДОДАТОК 10

Приклад 17-01 – Побудова параметричних кривих

```
<html>
<head>
<title> Приклад 17-01 – Побудова параметричних кривих </title>
<style>
body{
margin: 0;
overflow: hidden;
}
</style>
</head>
<body>
<script src="js/three.js"></script>
<script src="js/OrbitControls.js"></script>
<script>
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 10, window.innerWidth /
window.innerHeight, 0.1, 10000 );
camera.position.set( 400, 800, 0 );
var renderer = new THREE.WebGLRenderer({antialias: true});
renderer.setSize( window.innerWidth, window.innerHeight );
renderer.setClearColor(0x7b08f5);
document.body.appendChild( renderer.domElement );
var controls = new THREE.OrbitControls( camera, renderer.domElement );
var dirLight = new THREE.DirectionalLight( 0xffffff );
dirLight.position.set( -10, 10, 10 );
scene.add( dirLight );
var light = new THREE.DirectionalLight( 0xffffff );
light.position.set( 10, 10, -10 );
scene.add( light );
var a = 20, b = 4;
var geometry = new THREE.Geometry();
var material = new THREE.LineBasicMaterial({ color: 0x21f344 });
for (var i =0; i <= 30; i++ ) {
        var vec = new THREE.Vector3(a*Math.cos(i), a*Math.sin(i), b*i);
        geometry.vertices.push(vec);
}
var line = new THREE.Line(geometry, material);
scene.add(line);
render();
function render() {
```

```
controls.update();
  requestAnimationFrame(render);
  renderer.render( scene, camera );
}
</script>
</body>
</html>
```

ДОДАТОК 11

Приклад 18-01 –Рисування шахової фігури – королева

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/three.js/89/three.min.js"></script>
<script src="js/TrackballControls.js"></script>
<html>
<head>
<title> Приклад 18-01 –Рисування шахової фігури – королева </title>
<style>
body { margin: 0; overflow:hidden;}
canvas { width: 100%; height: 100% }
</style>
</head>
<body>
<script>
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 40, window.innerWidth /
window.innerHeight, 0.1, 1000 );
camera.position.z = 400;
camera.position.y = 150;
var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
renderer.setClearColor (new THREE.Color(0xffffff));
document.body.appendChild( renderer.domElement );
var light = new THREE.HemisphereLight( 0xffffff, 0xc0c0c0, 1 );
scene.add( light );

var queen = new THREE.Object3D();
var material = new THREE.MeshPhongMaterial ({color: 0xdaa520,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var points = [];
for (var i=-2.4; i<0.7; i=i+0.1)
{
points.push(new THREE.Vector2(10+18*Math.exp(-i*i),34*i));
}
var geometry = new THREE.LatheGeometry(points,32);
object1=new THREE.Mesh(geometry,material);
object1.position.set(0,26,0);
object1.rotation.x=Math.PI;
queen.add(object1);
```

```
var material = new THREE.MeshPhongMaterial ({color: 0xdaa520,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var points = [];
for (var i=-2.4; i<0.7; i=i+0.1)
{
points.push(new THREE.Vector2(10+11*Math.exp(-i*i),14*i)); }
var geometry = new THREE.LatheGeometry(points,32);
object2=new THREE.Mesh(geometry,material);
object2.position.set(0,130,0);
<!-- object1.rotation.x=Math.PI; -->
queen.add(object2);
var geometry = new THREE.CylinderGeometry(28,28,6,32);
var material  = new THREE.MeshPhongMaterial ({color: 0x4c3c18,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var disc1 =  new THREE.Mesh(geometry,material);
disc1.position.set(0,10,0);
queen.add(disc1);
var geometry = new THREE.CylinderGeometry(28,28,6,32);
var material = new THREE.MeshPhongMaterial ({color: 0xdaa520,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var disc2 =  new THREE.Mesh(geometry,material);
disc2.position.set(0,4,0);
queen.add(disc2);
var geometry = new THREE.CylinderGeometry(14,14,6,32);
var material  = new THREE.MeshPhongMaterial ({color: 0x4c3c18,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var disc3 =  new THREE.Mesh(geometry,material);
disc3.position.set(0,114,0);
queen.add(disc3);
var geometry = new THREE.CylinderGeometry(14,18,16,32);
var material = new THREE.MeshPhongMaterial ({color: 0xdaa520,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var disc4 =  new THREE.Mesh(geometry,material);
disc4.position.set(0,103,0);
queen.add(disc4);
var geometry = new THREE.CylinderGeometry(22,20,3,32);
var material  = new THREE.MeshPhongMaterial ({color: 0x4c3c18,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var disc5 =  new THREE.Mesh(geometry,material);
```

```
disc5.position.set(0,138,0);
queen.add(disc5);
var geometry = new THREE.SphereGeometry(7,9,10);
var material = new THREE.MeshPhongMaterial ({color: 0xdaa520,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var sphere =  new THREE.Mesh(geometry,material);
sphere.position.set(0,143,0);
queen.add(sphere);
scene.add(queen);
queen.position.set(0,-40,0);
controls = new THREE.TrackballControls( camera );
controls.rotateSpeed = 1.0;
controls.zoomSpeed = 1.2;
controls.panSpeed = 0.8;
controls.noZoom = false;
controls.noPan = false;
controls.staticMoving = true;
controls.dynamicDampingFactor = 0.3;
function render() {
requestAnimationFrame(render);
controls.update();
<!-- castle.rotation.y+=0.05 -->
renderer.render(scene, camera); }
render();
</script>
</body>
```

ДОДАТОК 12

Приклад 18-02 –Рисування шахової фігури – король

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/three.js/89/three.min.js"></script>
<script src="js/TrackballControls.js"></script>
<html>
<head>
<title> Приклад 18-02 –Рисування шахової фігури – король </title>
<style>
body { margin: 0; overflow:hidden;}
canvas { width: 100%; height: 100% }
</style>
</head>
<body>
<script>
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 40, window.innerWidth /
window.innerHeight, 0.1, 1000 );
camera.position.z = 400;
camera.position.y = 150;

var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
renderer.setClearColor (new THREE.Color(0xffffff));
document.body.appendChild( renderer.domElement );

var light = new THREE.HemisphereLight( 0xffffff, 0xc0c0c0, 1 );
scene.add( light );

var king = new THREE.Object3D();
var material = new THREE.MeshPhongMaterial ({color: 0xdaa520,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var points = [];
for (var i=-2.4; i<0.7; i=i+0.1)
{
points.push(new THREE.Vector2(10+18*Math.exp(-i*i),34*i));
}
var geometry = new THREE.LatheGeometry(points,32);
object1=new THREE.Mesh(geometry,material);
object1.position.set(0,26,0);
object1.rotation.x=Math.PI;
```

```
king.add(object1);

var material = new THREE.MeshPhongMaterial ({color: 0xdaa520,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var points = [];
for (var i=-2.4; i<0.7; i=i+0.1)
{
points.push(new THREE.Vector2(10+11*Math.exp(-i*i),14*i));
}
var geometry = new THREE.LatheGeometry(points,32);
object2=new THREE.Mesh(geometry,material);
object2.position.set(0,130,0);
<!-- object1.rotation.x=Math.PI; -->
king.add(object2);

var geometry = new THREE.CylinderGeometry(28,28,6,32);
var material  = new THREE.MeshPhongMaterial ({color: 0x4c3c18,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var disc1 =  new THREE.Mesh(geometry,material);
disc1.position.set(0,10,0);
king.add(disc1);

var geometry = new THREE.CylinderGeometry(28,28,6,32);
var material = new THREE.MeshPhongMaterial ({color: 0xdaa520,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var disc2 =  new THREE.Mesh(geometry,material);
disc2.position.set(0,4,0);
king.add(disc2);

var geometry = new THREE.CylinderGeometry(14,14,6,32);
var material  = new THREE.MeshPhongMaterial ({color: 0x4c3c18,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var disc3 =  new THREE.Mesh(geometry,material);
disc3.position.set(0,114,0);
king.add(disc3);
var geometry = new THREE.CylinderGeometry(14,18,16,32);
var material = new THREE.MeshPhongMaterial ({color: 0xdaa520,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var disc4 =  new THREE.Mesh(geometry,material);
disc4.position.set(0,103,0);
```

```
king.add(disc4);
var geometry = new THREE.CylinderGeometry(22,20,3,32);
var material  = new THREE.MeshPhongMaterial ({color: 0x4c3c18,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var disc5 =  new THREE.Mesh(geometry,material);
disc5.position.set(0,138,0);
king.add(disc5);
var geometry = new THREE.CylinderGeometry(7,11,6,32);
var material  = new THREE.MeshPhongMaterial ({color: 0xdaa520,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var disc6 =  new THREE.Mesh(geometry,material);
disc6.position.set(0,141,0);
king.add(disc6);
var geometry = new THREE.CylinderGeometry(6,2,13,4);
var material  = new THREE.MeshPhongMaterial ({color: 0xdaa520,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var korona1 =  new THREE.Mesh(geometry,material);
korona1.position.set(0,150,0);
korona1.rotation.x=Math.PI;
king.add(korona1);
var geometry = new THREE.CylinderGeometry(6,2,13,4);
var material  = new THREE.MeshPhongMaterial ({color: 0xdaa520,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var korona2 =  new THREE.Mesh(geometry,material);
korona2.position.set(0,160,0);
king.add(korona2);
var geometry = new THREE.CylinderGeometry(6,2,13,4);
var material  = new THREE.MeshPhongMaterial ({color: 0xdaa520,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var korona3 =  new THREE.Mesh(geometry,material);
korona3.position.set(-8,155,0);
korona3.rotation.x=Math.PI/2;
korona3.rotation.z=Math.PI/2;
king.add(korona3);
var geometry = new THREE.CylinderGeometry(6,2,13,4);
var material  = new THREE.MeshPhongMaterial ({color: 0xdaa520,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var korona4 =  new THREE.Mesh(geometry,material);
korona4.position.set(8,155,0);
```

```
korona4.rotation.x=Math.PI/2;
korona4.rotation.z=-Math.PI/2;
king.add(korona4);
var geometry = new THREE.SphereGeometry(5,5,10);
var material = new THREE.MeshPhongMaterial ({color: 0x4c3c18,
specular:0x00b2fc, shininess:50, blending:THREE.NormalBlending,
depthTest:true});
var sphere =  new THREE.Mesh(geometry,material);
sphere.position.set(0,155,0);
king.add(sphere);
scene.add(king);
king.position.set(0,-40,0);
scene.add(king);
king.position.set(0,-40,0);
controls = new THREE.TrackballControls( camera );
controls.rotateSpeed = 1.0;
controls.zoomSpeed = 1.2;
controls.panSpeed = 0.8;
controls.noZoom = false;
controls.noPan = false;
controls.staticMoving = true;
controls.dynamicDampingFactor = 0.3;
function render() {
requestAnimationFrame(render);
controls.update();
renderer.render(scene, camera);
}
render();
</script>
</body>
```

ДОДАТОК 13

Приклад 19-01 – ExtrudeGeometry

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Приклад 19.01 – ExtrudeGeometry</title>
<script src="./js/three.js"></script>
<script src="./js/TrackballControls.js"></script>
</head>
<body>
<script>
	init();
	function init() {
	// створюємо рендер і задаємо розмір
	var renderer = window.WebGLRenderingContext ? new
THREE.WebGLRenderer({antialias:true}) : new THREE.CanvasRenderer();
	renderer.setClearColor(new THREE.Color(0xEEEEEE));
	renderer.setSize( window.innerWidth, window.innerHeight );
	// додаємо вивід візуалізатора до html-елементу
	document.body.appendChild( renderer.domElement );
	// створюємо камеру
	var camera = new THREE.PerspectiveCamera(45,window.innerWidth /
window.innerHeight, 0.1,1000);
	// створюємо сцену, яка буде містити всі наші елементи, такі як
об’єкти, камери і джерела світла
	var scene = new THREE.Scene();
	var trackballcontrols = new THREE.TrackballControls( camera,
renderer.domElement );
	var ambientLight = new THREE.AmbientLight(0xffffff);
	scene.add(ambientLight);
	var spotLight = new THREE.SpotLight( 0xffffff );
	spotLight.position.set( -80,120, -20 );
	spotLight.castShadow = true;
	scene.add( spotLight );
	var arrowShape = new THREE.Shape();
	(function roundedRect( ctx ){
	ctx.moveTo( -80, 10 );
	ctx.lineTo( 10, 10 );
	ctx.lineTo( -20, 40 );
	ctx.lineTo( 10, 40 );
	ctx.lineTo( 50, 0 );
```

```
        ctx.lineTo( 10, -40 );
        ctx.lineTo( -20, -40 );
        ctx.lineTo( 10, -10 );
        ctx.lineTo( -80, -10 );
        ctx.lineTo( -80, 10 );
        }) ( arrowShape);
        var size = 8;
        var extrudeSettings ={ amount: size, bevelSegments: 8, curveSegments: 32
};
        var geometry = new THREE.ExtrudeGeometry( arrowShape,
extrudeSettings );
        var material = new THREE.MeshPhongMaterial({ color:0xcccccc,specular:
0x708090, ambient: 0x777777, shininess: 50 });
        var arrow_right = new THREE.Mesh( geometry, material );
        arrow_right.position.set( 100, 0, – size/2 );
        scene.add( arrow_right );
        camera.position.x = 250;
        camera.position.y = 40;
        camera.position.z = 250;
        camera.lookAt(scene.position);
        renderScene();
        function renderScene() {
        requestAnimationFrame( renderScene );
        trackballcontrols.update();
        renderer.render( scene, camera );
        }
};
</script>
</body>
</html>
```

Навчальне видання
(*українською мовою*)

Спиця Оксана Геннадіївна
Лісняк Андрій Олександрович
Мухін Віталій Вікторович
Мильцев Олександр Михайлович

ПРОГРАМУВАННЯ КОМП'ЮТЕРНОЇ ГРАФІКИ У ВЕБ

Навчальний посібник
для здобувачів ступеня вищої освіти бакалавра
спеціальності «Інженерія програмного забезпечення»
освітньопрофесійної програми «Програмна інженерія»

Рецензент *С. М. Гребенюк*
Відповідальний за випуск *А. О. Лісняк*
Коректор *О. Г. Спиця*