Создаем реалистичную 3D-графику в браузере с Three.js и Cannon.js

FRONTEND

Ирина Романова
фронтенд-разработчик

ВЕЩАЕТ

Ирина Романова

Работаю JS-программистом в FunBox.

СВЯЗАТЬСЯ СО МНОЙ

@i.romanova

ira.romanova

Трёхмерная графика

Трёхмерная графика — раздел компьютерной графики, посвящённый методам создания изображений или видео путём моделирования объёмных объектов в трёхмерном пространстве.

WebGL (Web Graphics Library)

Кроссплатформенный API для 3D-графики в браузере,
разрабатываемый некоммерческой организацией Khronos Group.

Приложение WebGL — всё сложно 🤯

1. Создается элемент canvas.
2. Определяется WebGL-контекст.
3. Определяется геометрия и сохраняется в буферных объектах.
4. Создаются и компилируются шейдерные программы.
5. Шейдерные программы связываются с буферными объектами.
6. Рисуется объект.

Three.js облегчает работу с WebGL

1. Набор готовых классов для создания интерактивной 3D-графики.
2. Представлена в 2010 году.
3. Открытый исходный код.

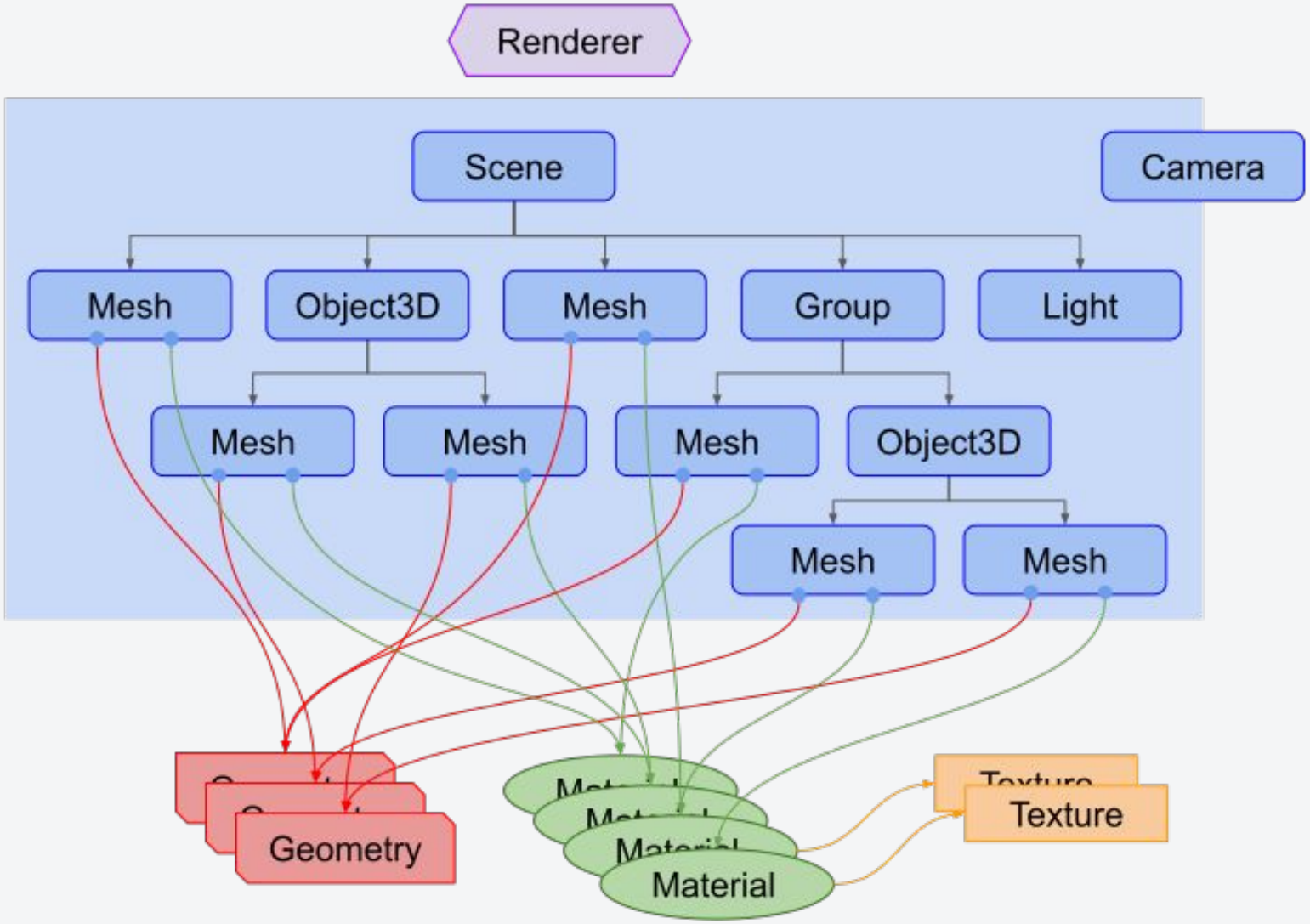
Физический движок

Производит компьютерное моделирование физических законов реального мира в виртуальном мире.

Cannon.js сделает графику реалистичной

1. Написан на JS и использует все его возможности.
2. Компактен и производителен.
3. Несложен в освоении.

Основные объекты Three.js



Структура приложения

Scene

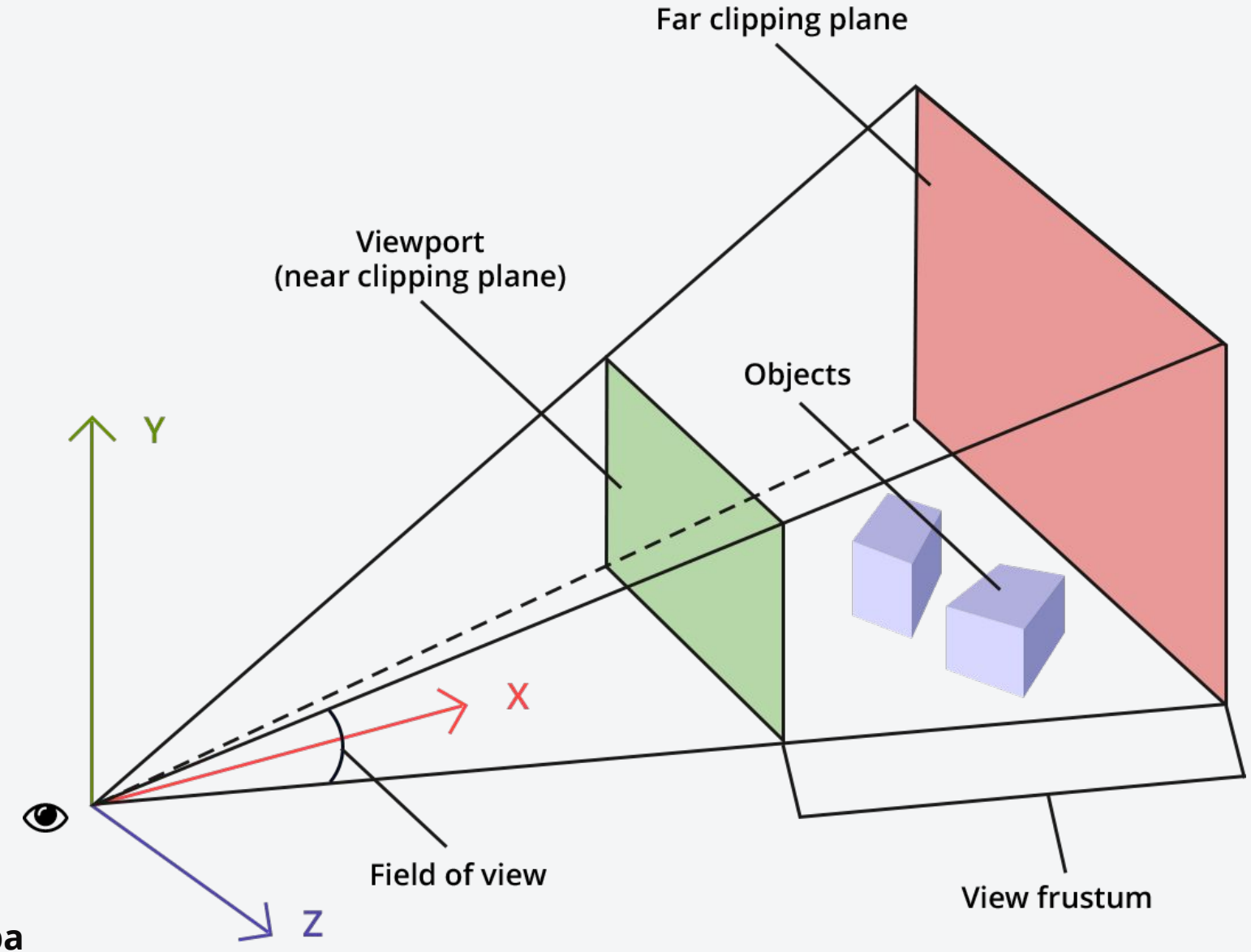
Объект Scene определяет корень графа сцены и содержит такие свойства, как цвет фона и туман.

Создадим сцену

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

Camera

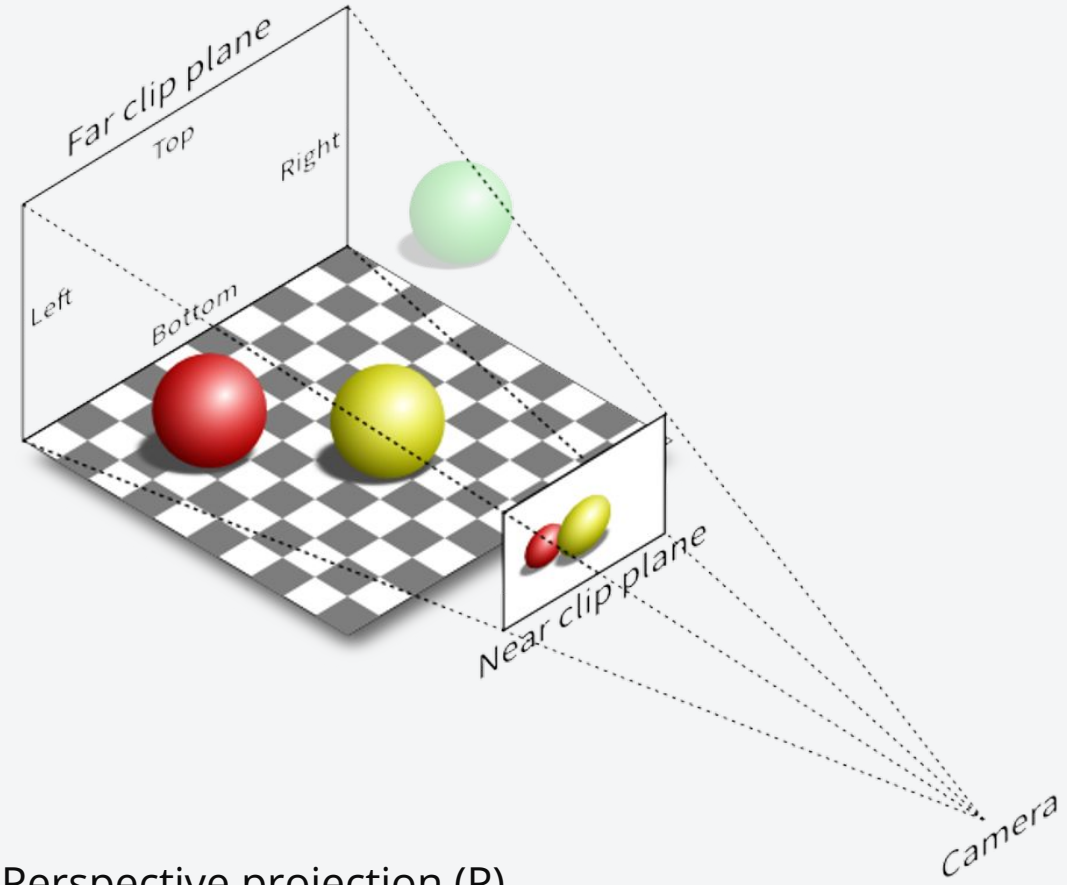
Объект, который определяет, где относительно сцены располагается и как ориентирован пользователь, а также другие свойства камеры реального мира.



Объём обзора

PerspectiveCamera

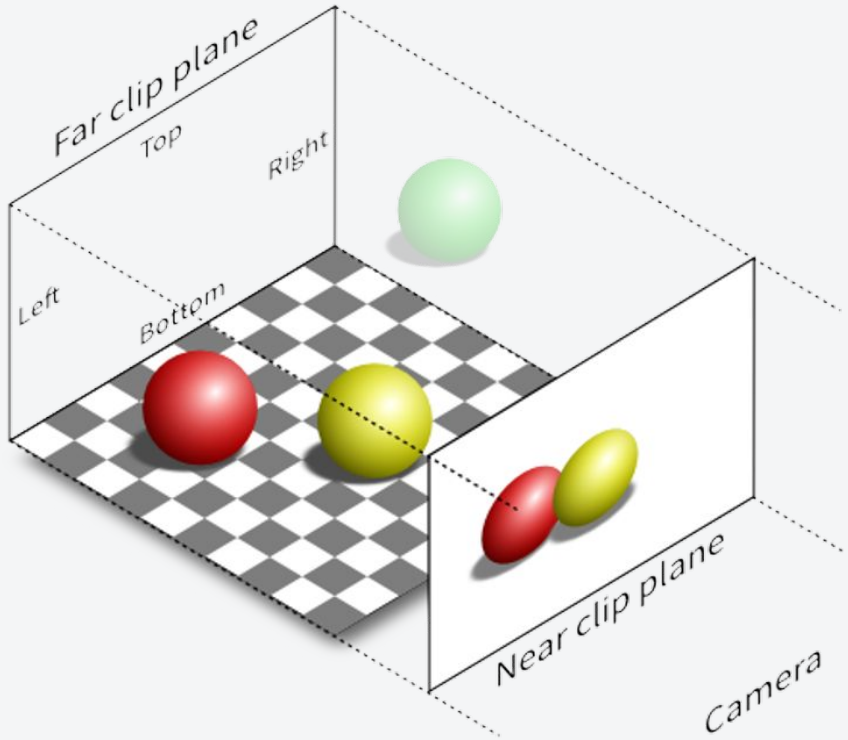
Использует перспективную проекцию



Perspective projection (P)

OrthographicCamera

Использует ортографическую проекцию



Orthographic projection (O)

Добавим камеру

```
const camera = new THREE.PerspectiveCamera(75, window.innerWidth
/ window.innerHeight, 0.1, 1000);

camera.position.set(0, 0, 10);
```

Renderer

1. Отвечает за фактический сбор всех предоставленных нами данных и их рендеринг на canvas.
2. На данный момент Three.js включает класс WebGLRenderer, который использует WebGL для визуализации 3D.

Добавим рендерер

```
const renderer = new THREE.WebGLRenderer();

renderer.setSize(window.innerWidth, window.innerHeight);

document.body.appendChild(renderer.domElement);


renderer.render(scene, camera);
```

Mesh

Сочетает в себе:

- Geometry — форма объекта.
- Material — свойства поверхности.
- Положение, ориентация и масштаб этого объекта в сцене относительно его родителя.

Geometry

Представляет данные о вершинах, которые определяют форму объекта.

- Встроенные примитивы — Box, Sphere, Plane и многие другие.
- Возможность создавать пользовательскую геометрию.
- Возможность загружать файлы с геометрией.

Material

Описывает, как нарисовать объект с учётом свойств его поверхности: блестящий или матовый, какого цвета, какую текстуру применить.

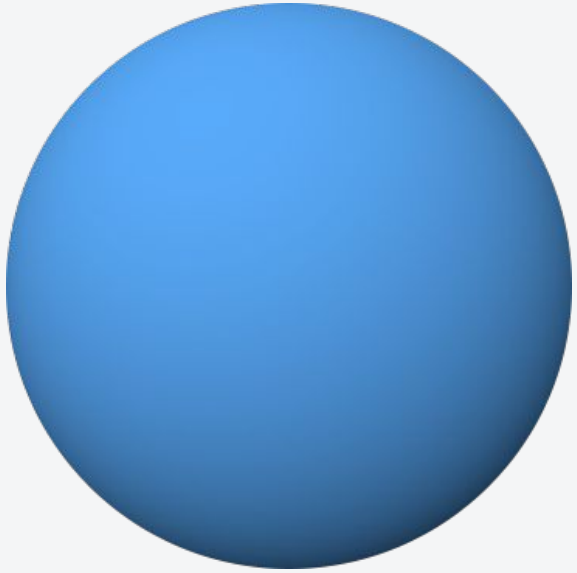
MeshBasicMaterial

Не учитывается освещённость

Basic

MeshLambertMaterial

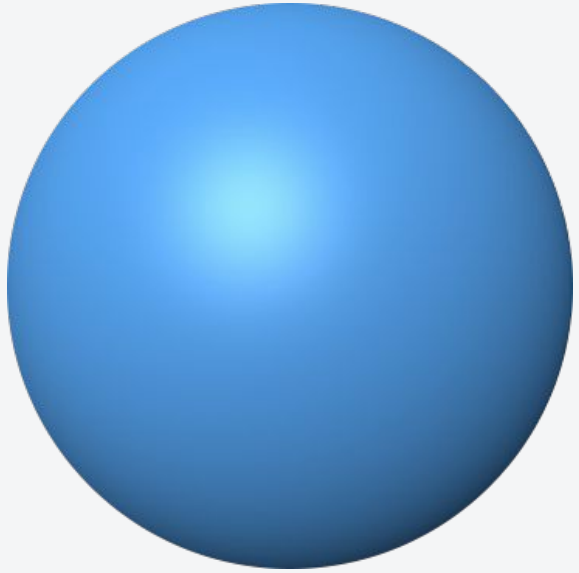
Вычисляется освещённость только для вершин



Lambert

MeshPhongMaterial

Вычисляется освещённость для каждого пикселя



Phong

Добавим объект на сцену

```
const geometry = new THREE.SphereGeometry(2, 32, 32);

const material = new THREE.MeshBasicMaterial({ color: '#cc182a' });

const sphere = new THREE.Mesh(geometry, material);

scene.add(sphere);
```

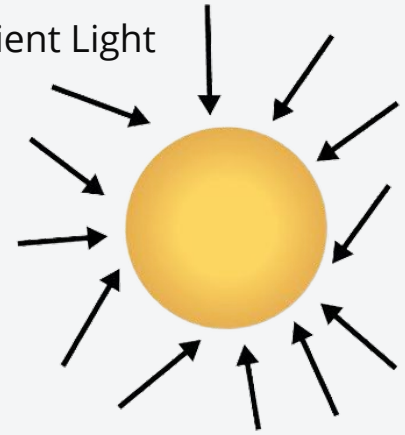
Поменяем материал на более реалистичный

```
const material = new THREE.MeshPhongMaterial({ color: '#cc182a' });
```

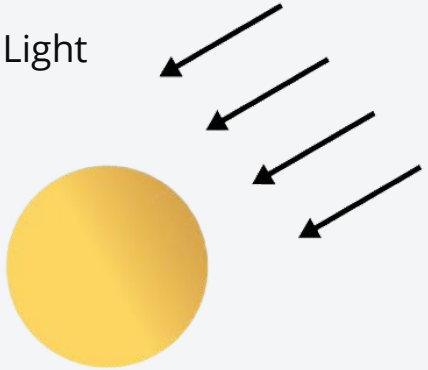
Light

Представляет освещение.

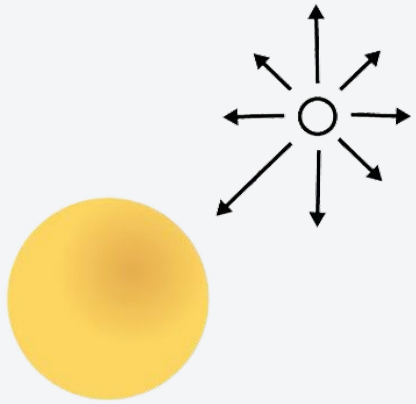
Ambient Light



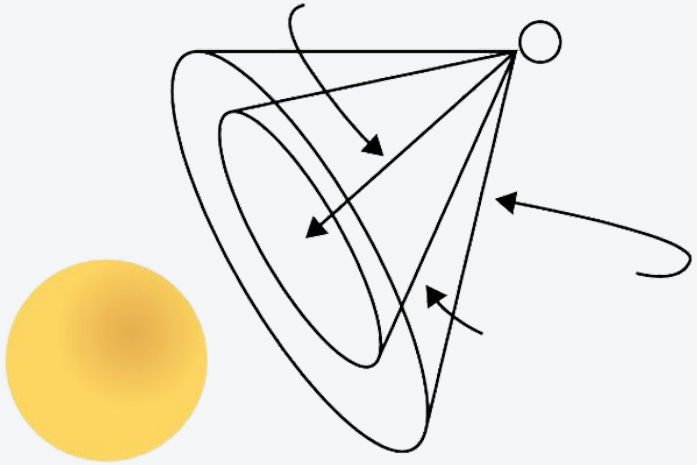
Directional Light



Point Light



Spot Light



Виды освещения

Добавим направленный свет

```
const light = new THREE.DirectionalLight('white', 1);

light.position.set(12, 12, 12);

scene.add(light);
```

И ещё подсветим общим рассеянным

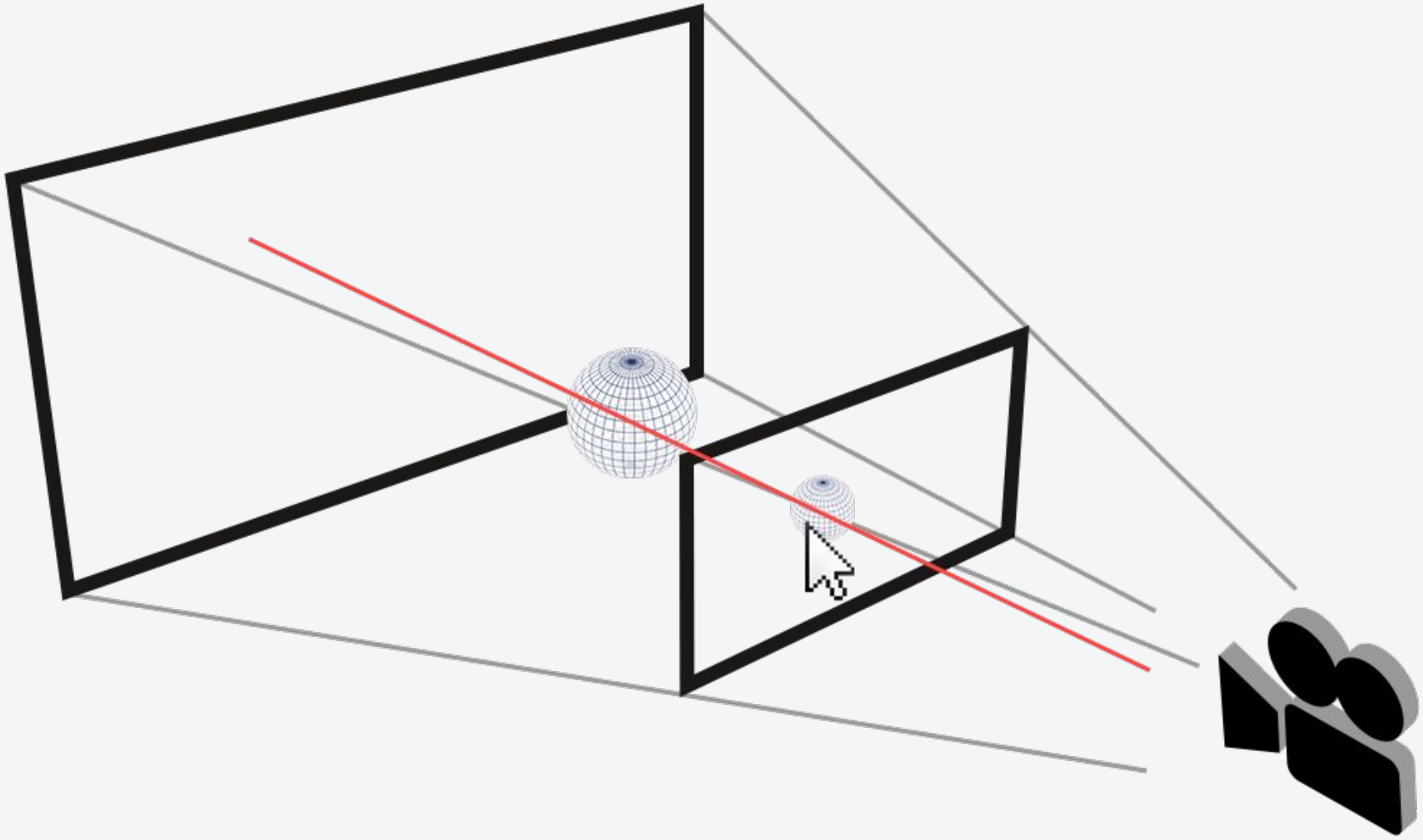
```
const additionalLight = new THREE.AmbientLight('white', 0.2);

scene.add(additionalLight);
```

Взаимодействие со сценой и выбор объекта

Метод Raycasting

Raycasting — метод, основанный на замерах пересечения лучей с визуализируемой поверхностью, широко используется в 3D.

Raycaster — класс Three.js, созданный для использования метода raycasting. Используется для выбора объектов мышью.



Метод Raycasting

Добавляем Raycaster

```
const raycaster = new THREE.Raycaster();
```

И обработчик клика по документу

```
function onClick(event) {
const x = ( event.clientX / window.innerWidth ) * 2 - 1;
const y = -( event.clientY / window.innerHeight ) * 2 + 1;
}
```

Находим пересечения

```
raycaster.setFromCamera( new THREE.Vector2(x, y), camera );

const intersects = raycaster.intersectObjects( scene.children );
```

Манипулируем объектами

```
if ( intersects.length ) {
    const { object } = intersects[0];
    object.material.color.set('#6919bf');
    renderer.render(scene, camera);
}
```

Перемещение объектов

Перемещаем шарик сверху вниз

```
const animate = function () {
 requestAnimationFrame( animate );
 sphere.position.y -= 0.02;
 renderer.render( scene, camera );
};
animate();
```

А если добавить плоскость?

```
const groundGeometry = new THREE.PlaneGeometry( 100, 50 );

const groundMaterial = new THREE.MeshBasicMaterial( {color: '#ced6d0'} );

const ground = new THREE.Mesh( groundGeometry, groundMaterial );

scene.add( ground);
```

Поворачиваем плоскость вокруг оси X

```
const quaternion = new THREE.Quaternion();

quaternion.setFromAxisAngle( new THREE.Vector3( 1, 0, 0 ), -Math.PI/2 );

const { x,y,z,w } = quaternion;

ground.quaternion.set(x,y,z,w);
```

Добавляем физику с Cannon.js

World

Каждое приложение Cannon.js начинается с создания объекта World.
World управляет объектами и симуляцией.

Создаём мир

```
const world = new CANNON.World();

world.gravity.set(0, -9.8, 0);
```

Body

Объект, представляющий физическое тело.

Нам потребуется тело для шара

```
const sphereBody = new CANNON.Body({
    mass: 200,
    shape: new CANNON.Sphere(2),
    position: new CANNON.Vec3(0, 25, 0)
});
world.addBody(sphereBody);
```

И тело для поверхности земли

```
const groundBody = new CANNON.Body({
    mass: 0,
    shape: new CANNON.Plane(),
    position: new CANNON.Vec3(0, 0, 0)
});
groundBody.quaternion.set(x,y,z,w);
world.addBody(groundBody);
```

Симуляция мира

Cannon.js использует вычислительный алгоритм, называемый интегратором. Интеграторы моделируют уравнения физики в дискретные моменты времени. Обычно физические движки используют шаг в 1/60 секунды.

Задаем шаг времени

```
const animate = function () {
 requestAnimationFrame( animate );
 world.step(1 / 60);
// делаем что-то полезное здесь
 renderer.render( scene, camera );
};
animate();
```

Доработаем рендеринг

```
const {x, y, z} = sphereBody.position;

sphere.position.set(x, y, z);

const {xQ, yQ, zQ, wQ} = sphereBody.quaternion;

sphere.quaternion.set(xQ, yQ, zQ, wQ);
```

Добавим ещё один объект на стену

```
const boxGeometry = new THREE.BoxGeometry(2, 5, 10);

const box = new THREE.Mesh(boxGeometry, new
THREE.MeshPhongMaterial({color: '#77bf19' }));

box.position.set(-10,5,5);

scene.add(box);
```

И добавим его тело

```
const boxBody = new CANNON.Body({
 mass: 50,
 shape: new CANNON.Box(new CANNON.Vec3(1, 2.5, 5)),
 position: new CANNON.Vec3(-10,5,5)
});
world.addBody(boxBody);
```

Свяжем тела и визуализации через имя

```
sphere.name = 'sphere';

sphereBody.name = 'sphere';

box.name = 'box';

boxBody.name = 'box';
```

При клике найдём тело

```
const { object } = intersects[0];

const body = world.bodies.find(body => body.name === object.name);
```

Сообщим телу импульс на клик

```
const impulse = new CANNON.Vec3(-1500,30,30);

body.applyLocalImpulse(impulse, new CANNON.Vec3());
```

Дополнительные материалы

Для вдохновения

Портфолио Bruno Simon

SDU Future Education

Scars For Honor

Для изучения

Документация и примеры Three.js

Документация и примеры Cannon.js

Hello Cannon.js!

Building a Physics-based 3D Menu with Cannon.js Three.js