Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э. Баумана

(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» КАФЕДРА «Информационная безопасность»

**Контрольная работа 1:**

**«Теория игр**

**и исследование операций»**

**Решение бесконечной игры поиска на эллипсоиде**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: |  |  | Гончарова М.К.  (И.О. Фамилия) |
| Преподаватель: |  |  | Коннова Н.С.  (И.О. Фамилия) |

2020 г.

Постановка задачи:

Бесконечные антагонистические игры отличаются от матричных игр тем, что в них оба игрока имеют бесконечное множество стратегий. В бесконечных играх исследуются системы вида

X и Y – произвольные бесконечные множества, элементы которых являются стратегиями игроков 1 и 2 соответственно.

H – функция выигрыша игрока 1:

Постановка задачи:

Игрок 1 выбирает произвольным образом s точек на эллипсоиде. Его вектор стратегий состоит их координат данных точек  Игрок 2 выбирает точку на том же эллипсоиде. Таким образом, функция выигрыша для игрока 1 имеет следующий вид:

r – радиус поражения каждой точки, выбранной игроком 1. Таким образом, игрок 1 выигрывает в том случае, если игрок 2 выбрал такую точку, евклидово расстояние от которой хотя бы до одной из точек игрока 1 меньше r. Таким образом можно провести m экспериментов и узнать цену игры:

Ход работы:

Очевидно, что для решения данной задачи аналитическим методом необходимо вычислить следующую формулу:

Нахождение площади пересечения сферы и эллипсоида является на данный момент surface-to-surface проблемой.

SSI проблема является основной задачей в автоматизированном геометрическом проектировании. Звучит она следующим образом: имеются две пересекающиеся поверхности в , вычислить все части кривой пересечения. На данный момент удалось решить данную задачу только для некоторых специальных классов поверхностей.

Можно попробовать приближённо оценить H, приняв поверхность пересечения сферы и эллипсоида за круг. Формулу площади поверхности эллипсоида нельзя выразить при помощи простейших функций, поэтому воспользуемся приближённой формулой, предложенной Кнудом Томсеном. Тогда, для следующих параметров:

Описание алгоритма программы:

На вход данной программы поступают следующие параметры:

* Количество точек покрытия эллипсоида (параметр s);
* Параметры a, b и c с помощью которого задается эллипсоид;
* Количество раундов в игре;
* Радиус поражения.

В самом начале мы формируем покрытие для игрока 1. Для равномерного покрытия эллипсоида точками решим вспомогательную задачу: равномерно распределим точке на прямоугольнике со сторонами и 2. Каждая полученная точка в таком квадрате своими координатами однозначно задаёт точку на эллипсоиде (так как для задания точки на эллипсоиде достаточно двух углов со следующими ограничениями: ). Для описанного распределения воспользуемся последовательностью . – d-мерная квазислучайная последовательность с низким расхождением, не требующая выбора базисных параметров, основанная на золотом сечении. Существует множество способов обобщения последовательности Фибоначчи и/или золотого сечения. Мы определяем обобщённый вид золотого сечения как уникальный положительный корень . Тогда, для d=2, 1.32471795724474602596. Это значение часто называется пластической константой. Предполагается, что это значение с наибольшей вероятностью является оптимальным для соответствующей двухмерной задачи. Сама последовательность выглядит следующим образом:

Таким образом, координаты x, y (то есть углы ) в нашем прямоугольнике будут получены по следующим формулам:

Эллипсоид имеет следующее параметрическое уравнение:

Программа получает необходимое количество точек, далее для каждой точки строится сфера на эллипсоиде.

Далее начинается сама игра. Игрок 2 каждый раунд выбирает произвольную точку на эллипсоиде. Для данной точки проверяется, попала ли она в покрытие игрока 1, если попала, то игрок 1 одерживает победу.

Для расчета цены игры используется формула, приведенная в постановке задачи.

Также для сравнения реализован алгоритм случайного распределения точек: в заданных интервалах генерируются значения двух углов, далее вычисляются координаты центра сферы на эллипсоиде и, если новая сфера не пересекается ни с одной из уже добавленных, точка добавляется в массив для дальнейшей отрисовки.

Пример:

Введём следующие параметры для решения бесконечной игры поиска (рисунок 1):

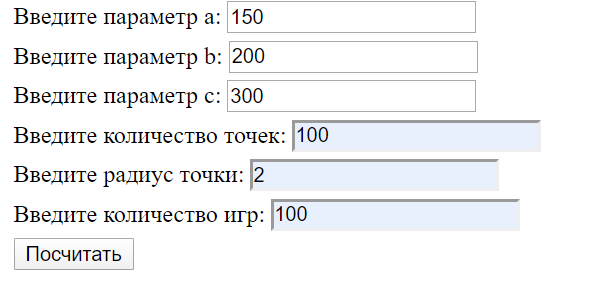


Рисунок 1 – Параметры для решения бесконечной игры поиска

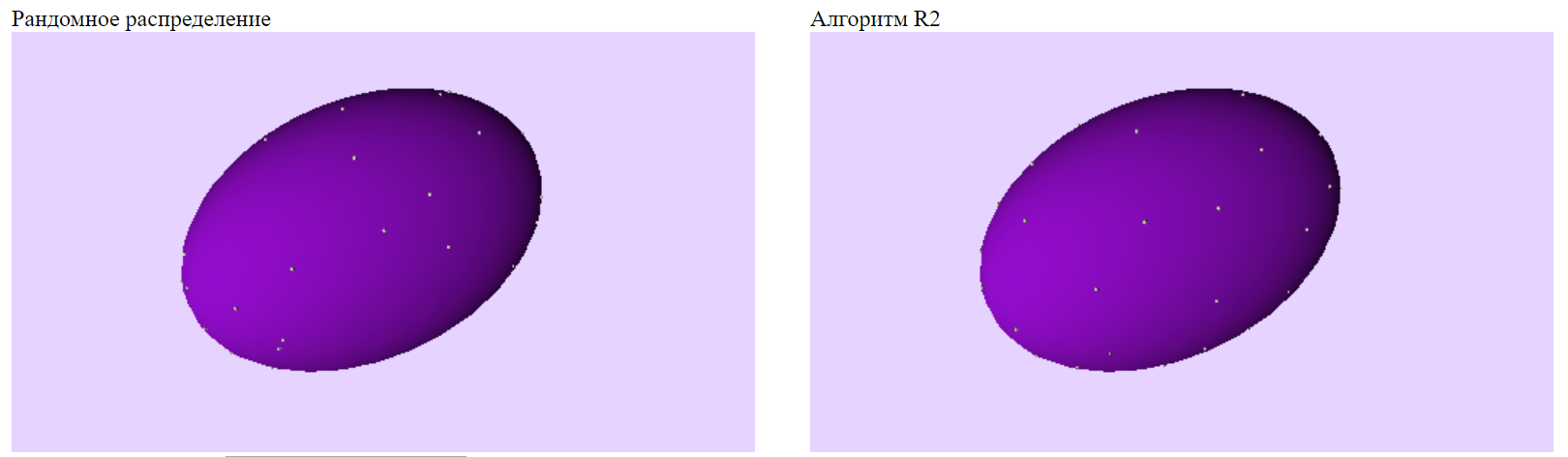
Выбранные точки игрока 1 в данном примере представлены на рисунке 2: 

Рисунок 2 – Выбранные точки игрока 1

Для заданных параметров были получены следующие результаты (рисунок 3):

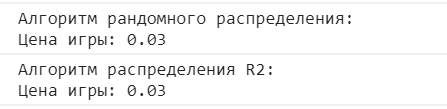


Рисунок 3 – Цена игры при 100 раундах

Увеличим количество раундов до 1000, не меняя остальные параметры. Получим следующие результаты (рисунок 4).

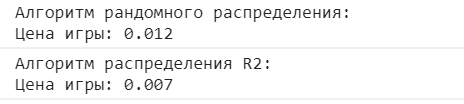


Рисунок 4 –Цена игры при 1000 раундах

Попробуем увеличить количество раундов до 10000 для тех же параметров и получим следующий результат (рисунок 5).

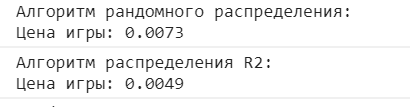


Рисунок 5 –Цена игры при 10000 раундах

Заметим, что цена игры для рандомного распределения получается выше, чем для распределения R2. Попробуем увеличить количество точек до 500, не меняя остальные параметры (рисунок 6):

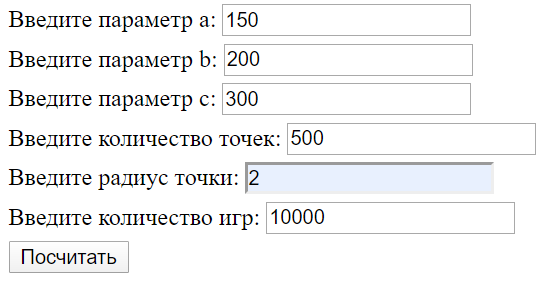


Рисунок 6 –Параметры игры для 500 точек

Получившееся распределение представлено на рисунке 7:

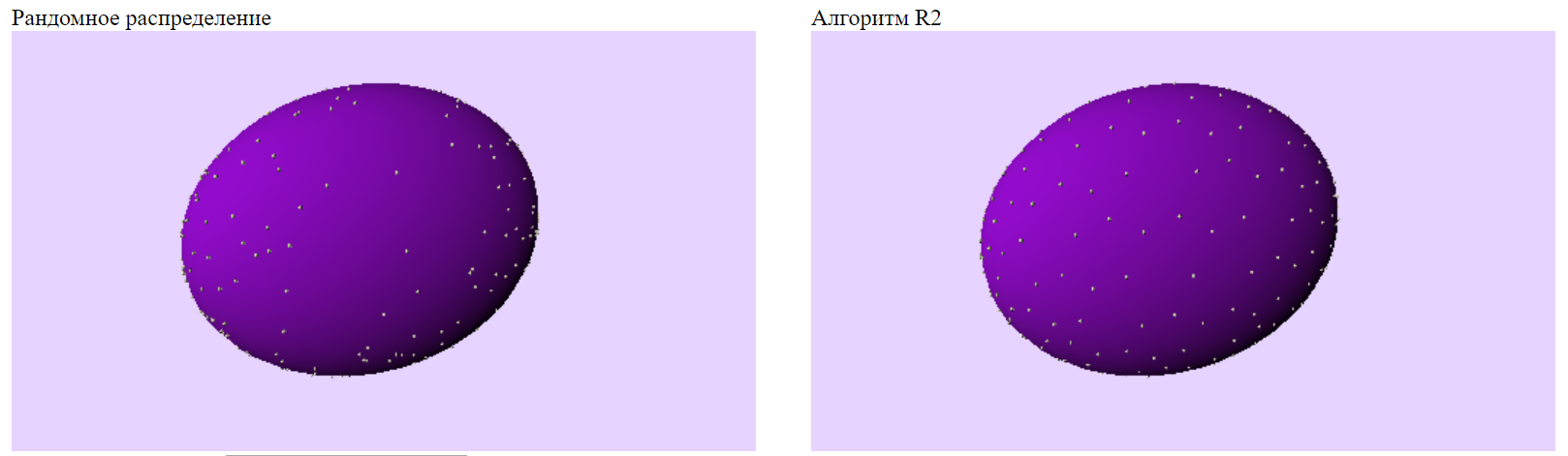


Рисунок 7 – Выбранные точки игрока 1 при количестве точек = 500

При заданном количестве уже можно заметить, что точки, построенные по алгоритму R2 распределены более равномерно. Обратим внимание на получившиеся цены игр (рисунок 8):

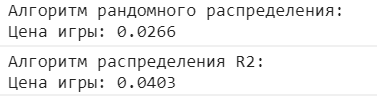


Рисунок 8 –Цена игры при количестве точек = 500

Попробуем увеличить количество точек до 700, не изменяя остальные параметры. Получаем следующее распределение (рисунок 9) и результаты (рисунок 10):

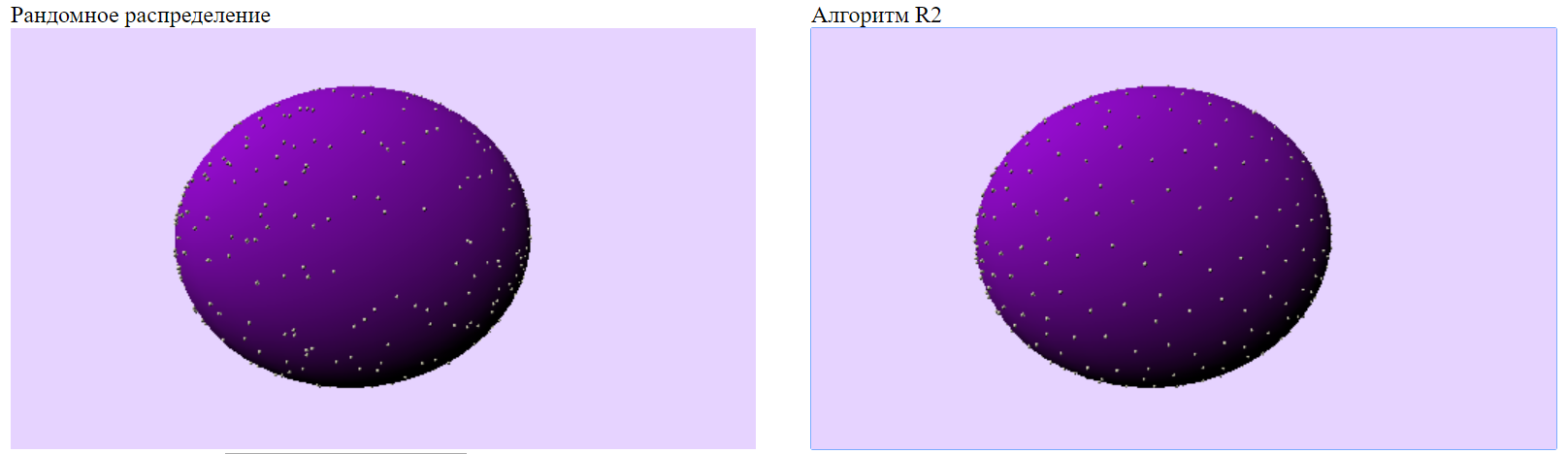


Рисунок 9 – Выбранные точки игрока 1 при количестве точек = 700

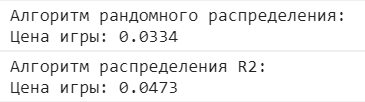


Рисунок 10 –Цена игры при количестве точек = 700

Таким образом, при увеличении количества точек, алгоритм распределения точек R2 демонстрирует лучшие результаты, чем случайное распределение.

Увеличивая радиус точек покрытия, можно ожидать увеличения цены игры.

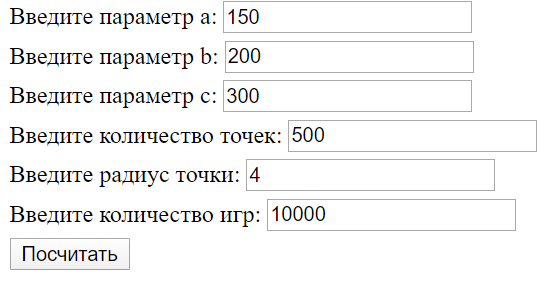


Рисунок 11 – Параметры игры для радиуса точки = 4

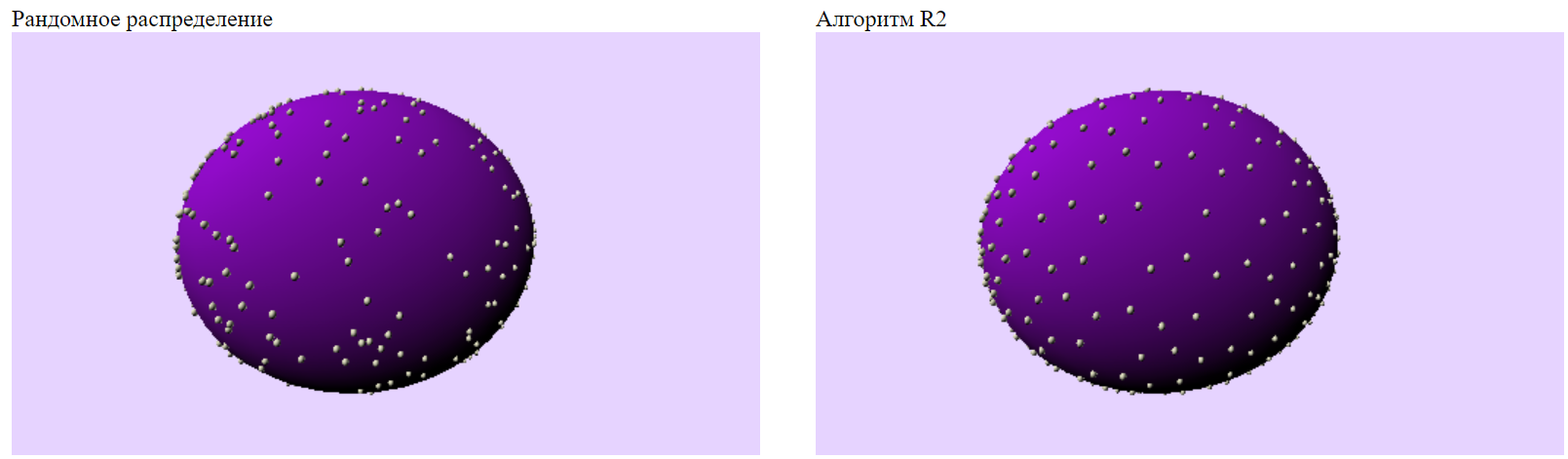


Рисунок 12 – Выбранные точки игрока 1 при радиусе точек = 4

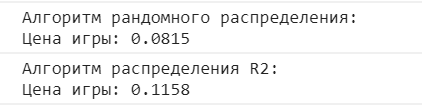


Рисунок 13 –Цена игры при радиусе точек = 4

Вывод:

В результате выполнения работы была написана программа, которая экспериментальным путем решает бесконечную игру поиска на эллипсоиде. Был приведен пример решения для определенных параметров игры и параметров эллипсоида, было показано изменение цены игры в зависимости от изменения какого-либо параметра, а также было проведено сравнение двух способов распределения точек на эллипсоиде.

ПРИЛОЖЕНИЕ А

Файл geometry.js

const THREE = require('three');  
const OrbitControls = require('three-orbitcontrols');  
  
//-------------------------------------------------------------------------  
let scene = new THREE.Scene();  
scene.background = new THREE.Color(0xE6D3FF);//ffe4c4  
let scene2 = new THREE.Scene();  
scene2.background = new THREE.Color(0xE6D3FF);  
  
let renderer = new THREE.WebGLRenderer();  
let renderer2 = new THREE.WebGLRenderer();  
let camera = new THREE.PerspectiveCamera(75, ***window***.innerWidth / ***window***.innerHeight, 0.1, 1000);  
let controls = new OrbitControls(camera, renderer.domElement);  
  
renderer.setSize(***window***.innerWidth / 2.5, ***window***.innerHeight / 2);  
renderer2.setSize(***window***.innerWidth / 2.5, ***window***.innerHeight / 2);  
***document***.getElementById('scene').appendChild(renderer.domElement);  
***document***.getElementById('scene2').appendChild(renderer2.domElement);  
  
camera.position.z = 400;  
  
  
let val = ***document***.getElementById('count');  
  
val.addEventListener('click', algorithm);  
  
function getRandomArbitrary(min, max) {  
 return ***Math***.random() \* (max - min) + min;  
}  
  
function findDistance(point1, point2) {  
  
 return ***Math***.sqrt((point1.x - point2.x) \*\* 2 + (point1.y - point2.y) \*\* 2 + (point1.z - point2.z) \*\* 2)  
}  
  
function getRandomPoint(a, b, c) {  
 let teta = getRandomArbitrary(0, ***Math***.PI);  
 let fi = getRandomArbitrary(0, 2 \* ***Math***.PI);  
 let p = {  
 x: a \* ***Math***.sin(teta) \* ***Math***.cos(fi),  
 y: b \* ***Math***.sin(teta) \* ***Math***.sin(fi),  
 z: c \* ***Math***.cos(teta)  
 };  
 //console.log(p);  
 return p;  
}  
  
function game(a, b, c, r, points, gameCount) {  
  
 let counter = 0;  
 for (let i = 0; i < gameCount; i++) {  
 let point = getRandomPoint(a, b, c);  
 let goodPoint = points.find(function (item) {  
 return (findDistance(item, point) <= r)  
  
 });  
 if (goodPoint !== undefined) {  
 counter++;  
  
  
 }  
  
 }  
 let price = (counter / gameCount);  
 return (price);  
   
  
}  
  
function randomDist(a, b, c, pointsCount, pointRadius) {  
 let points = [];  
 let pCounter = 0;  
 while (pCounter < pointsCount) {  
 let point = getRandomPoint(a, b, c);  
 let checkFist = points.find(function (item) {  
 return findDistance(point, item) <= 2 \* pointRadius;  
  
 });  
 if (checkFist === undefined) {  
 points.push(point);  
 pCounter++;  
 }  
  
 }  
 //console.log(points);  
 return points;  
}  
  
function r2Dist(a, b, c, pointsCount) {  
  
 let g = 1.32471795724474602596;  
 let a1 = 1.0 / g;  
 let a2 = 1.0 / (g \* g);  
  
  
 let points = [];  
  
 for (let i = 0; i < pointsCount; i++) {  
 let teta = ((a1 \* (i + 1)) % 1) \* ***Math***.PI;  
 let fi = ((a2 \* (i + 1)) % 1) \* 2 \* ***Math***.PI;  
  
  
 let point = {  
 x: a \* ***Math***.sin(teta) \* ***Math***.cos(fi),  
 y: b \* ***Math***.sin(teta) \* ***Math***.sin(fi),  
 z: c \* ***Math***.cos(teta)  
 };  
 points.push(point);  
 }  
 return points;  
}  
  
  
function algorithm(e) {  
 e.preventDefault();  
 scene.dispose();  
 while (scene.children.length > 0) {  
 scene.remove(scene.children[0]);  
 }  
 scene2.dispose();  
 while (scene2.children.length > 0) {  
 scene2.remove(scene2.children[0]);  
 }  
  
 const frontSpot = new THREE.SpotLight(0xeeeece);  
 frontSpot.position.set(1000, 1000, 1000);  
 const frontSpot2 = new THREE.SpotLight(0xddddce);  
 frontSpot2.position.set(-500, -500, -500);  
  
 const frontSpot3 = new THREE.SpotLight(0xeeeece);  
 frontSpot3.position.set(1000, 1000, 1000);  
 const frontSpot4 = new THREE.SpotLight(0xddddce);  
 frontSpot4.position.set(-500, -500, -500);  
  
 scene.add(frontSpot, frontSpot2);  
 scene2.add(frontSpot3, frontSpot4);  
  
 let pointsCount = ***document***.getElementById('number').value;  
 let pointRadius = ***document***.getElementById('radius').value;  
 let gamesCount = ***document***.getElementById('gameCount').value;  
 let a = ***document***.getElementById('a').value;  
 let b = ***document***.getElementById('b').value;  
 let c = ***document***.getElementById('c').value;  
  
 let pointsRand = randomDist(a, b, c, pointsCount, pointRadius);  
 let pointsR2 = r2Dist(a, b, c, pointsCount);  
  
 let priceRandom = game(a, b, c, pointRadius, pointsRand, gamesCount);  
 let priceR2 = game(a, b, c, pointRadius, pointsR2, gamesCount);  
 ***console***.log('Алгоритм рандомного распределения:' + '\n' + 'Цена игры: ' + priceRandom);  
 ***console***.log('Алгоритм распределения R2:' + '\n' + 'Цена игры: ' + priceR2);  
  
 let matrix = new THREE.Matrix4().makeScale(a, b, c);  
  
 let geometryEllipse = new THREE.SphereGeometry(1, 32, 32).applyMatrix4(matrix);  
 let geometryEllipse2 = new THREE.SphereGeometry(1, 32, 32).applyMatrix4(matrix);  
  
 const material = new THREE.MeshLambertMaterial({  
 color: 0x9F0DFF,  
 //specular: 0x9F0DFF,  
 //emissive: 0x6809A8  
 // wireframe: true  
 });  
 let ellipse = new THREE.Mesh(geometryEllipse, material);  
 let ellipse2 = new THREE.Mesh(geometryEllipse2, material);  
  
 scene.add(ellipse);  
 scene2.add(ellipse2);  
  
 let makeSphere = (item) => {  
 const materialS = new THREE.MeshLambertMaterial({  
 color: 0xFFFFFF,  
 // wireframe: true  
 });  
 let Sphere = new THREE.Mesh(new THREE.SphereGeometry(pointRadius, 32, 32), materialS);  
 Sphere.position.x = item.x;  
 Sphere.position.y = item.y;  
 Sphere.position.z = item.z;  
 return Sphere;  
 };  
  
 let spherasRand = pointsRand.map(makeSphere);  
  
 let spherasR2 = pointsR2.map(makeSphere);  
  
  
 scene.add(...spherasRand);  
 scene2.add(...spherasR2);  
 ***console***.log("ready")  
  
}  
  
  
function animate() {  
 requestAnimationFrame(animate);  
 // ellipse.rotation.x += 0.01;  
 // ellipse.rotation.y += 0.01;  
  
 // Sphere.rotation.y += 0.01;  
 // ellipse.rotation.z +=0.01;  
 controls.update();  
 //controls2.update();  
 renderer.render(scene, camera);  
 renderer2.render(scene2, camera);  
  
}  
  
  
animate();

ПРИЛОЖЕНИЕ B

Файл 3d.html

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>3d</title>  
 <link rel="stylesheet" href="./style.css">  
 <script src="geometry.js"></script>  
</head>  
<body>  
<div class = 'container'>  
 <div id="scene"> Рандомное распределение </div>  
 <div id="scene2">Алгоритм R2 </div>  
 <div class="wrapper">  
 <form>  
 <div class='child'>  
 <label for="a">  
 Введите параметр а:  
 </label>  
 <input id="a" value="150">  
 </div>  
 <div class='child'>  
 <label for='b'>  
 Введите параметр b:  
 </label>  
 <input id ='b' value="200">  
 </div>  
 <div class='child'>  
 <label for="c">  
 Введите параметр c:  
 </label>  
 <input id="c" value="300">  
 </div>  
 <div class='child'>  
 <label for="number">  
 Введите количество точек:  
 </label>  
 <input id="number" value="100">  
 </div>  
 <div class="child">  
 <label for="radius">  
 Введите радиус точки:  
 </label>  
 <input id="radius" value="2">  
 </div>  
 <div class="child">  
 <label for="gameCount">  
 Введите количество игр:  
 </label>  
 <input id="gameCount" value="100">  
 </div>  
  
 <button id="count"> Посчитать</button>  
 </form>  
 </div>  
</div>  
</body>  
</html>