**HTML5 круговая панорама на three.js**

* [JavaScript](https://habr.com/ru/hub/javascript/)
* [Из песочницы](https://habr.com/ru/sandbox/)

    В последнее время все чаще можно наткнуться на круговую панораму в интернете. Взять к примеру тот же [сырный домик](http://dom.ikea.ru/shkola-remonta/issue/14). Это эффектная штучка возможно вскоре станет обыкновенным делом для сайтов ресторанов, гостиниц, отелей и т. п. Однако, что делать, если существуют проблемы с flash на компьютере? Это реальная ситуация, которая возникла у меня на работе, по причине жесткой политики безопасности IT-отдела.  
    В этой небольшой статье я расскажу вам о трудностях, которые встретил на пути создания интерактивной панорамы на WebGL + THREE.js.  
  
  
Определим требования к панораме:

* используем WebGL, так как Canvas панорамы просто ужас, и еще новый IE [поддерживает](http://habrahabr.ru/post/174865/);
* Просмотр панорамы путем перетаскиванием зажатой кнопки мыши;
* Активные элементы призванные обратить внимание и сообщить информацию;
* Возможность посмотреть несколько панорам на одной странице.

    Чтобы понять, как это реализовать, предлагаю представить следующее: «Мы находимся в центре сферы, на внутреннюю поверхность которой нанесена текстура панорамы. Мы — это камера, которая может поворачиваться и смотреть вверх или вниз. На внутренней поверхности сферы ( „у“ поверхности, как оказалось ) находятся элементы, и когда мы наводим на них мышку, появляется окошко с информацией».

**Cфера**

    Существует как минимум два варианта реализации панорамы: кубическая и сферическая. Я выбрал сферическую, так как размещение визуально одинаковых активных элементов предполагает равноудаленное расстояние от центра. За прототип возьмем существующий [образец панорамы](http://threejs.org/examples/webgl_panorama_equirectangular.html) с сайта разработчика библиотеки. Он нам полностью подходит за одним исключением. Это сфера с текстурой вывернутая на изнанку. Трудно представить как это может выглядеть.

mesh.scale.x = -1;

    К сожалению, одна эта деталь ставит полный крест на возможности добавления элементов. Суть в том, что получить хоть какую то информацию об объекте в пространстве при помощи указателя, надо «выстрелить» лучом (Raycaster) из точки указателя в направление вектора, только в данном случае луч не ударится о стенку сферы, потому как камера (логичная точка начала) не находится внутри. Сделано это потому, что текстура натягивается только с внешней стороны, но теперь есть возможность это исправить просто добавив свойство **side** в объект материи сферы:

mesh = new THREE.Mesh( new THREE.SphereGeometry( 500, 60, 40 ), new THREE.MeshBasicMaterial( {

map: THREE.ImageUtils.loadTexture( 'textures/2294472375\_24a3b8ef46\_o.jpg' )

side: THREE.BackSide

} ) );

Кстати, это типичное создание объекта в мире THREE.js. Как правило создается геометрия объекта, материал ( текстура ), а потом добавляется на сцену.

**Элементы**

Теперь надо подумать о том, что из себя будут представлять те самые активные элементы. В идеале ими должны быть плоские спрайты, всегда отображающиеся лицом к камере. Да, да именно так :)  
  
  
  
    Но здесь нас ждет сюрприз. В THREE.js существует два «класса» для рендеринга: *WebGLRenderer* и *CanvasRenderer*. И два типа простых объектов: *Particle* (частица) и *Sprite*. WebGL работает с Sprite, а Canvas с Particle. При этом нам необходим именно WebGL, а объект Sprite не имеет объема в пространстве. То есть по нему **невозможно** попасть мышкой.  
    Что ж возьмем и с имитируем эти самые спрайты при помощи 3D объекта с геометрией плоскости ( PlaneGeometry):

var itemGeometry = new THREE.PlaneGeometry(35, 35),

pointMapHovered = THREE.ImageUtils.loadTexture( "img/information-hover.PNG" ),

pointMap = THREE.ImageUtils.loadTexture( "img/information-unhover.PNG" );

point = new THREE.Mesh( itemGeometry, new THREE.MeshBasicMaterial({

map: pointMap,

transparent:true //планка квадратная, не думаю, что кому-нибудь понравятся черные углы

}) );

    Хочу обратить ваше внимание, почему именно так: закеширована геометрия точки и текстура, а материал будет каждый раз создаваться заново. Дело в том, что при наведении мышки на точку, мы будем подменять текстуру на более яркую, к примеру, но если MeshBasicMaterial закеширована и является для всех общей, загораться будут все. Насчет наведения мышкой, все просто — берем [отсюда](http://threejs.org/examples/webgl_interactive_cubes.html).  
    Последняя важная часть, которая вызвала затруднения — это перерасчет координат области панорамы на странице на координаты WebGL пространства для точного выделения точек. Визуально элементы видны, но их координаты на страничке не совпадают с координатами 3D пространства. На StackOverFlow нашелся [ответ](http://stackoverflow.com/questions/11586527/converting-world-coordinates-to-screen-coordinates-in-three-js-using-projection).

**Данные**

     Так как предполагается возможность просматривать много панорам на одной странице, лучше все данные поместить в JSON. Примерно так:

{

"points": [

{

"coords": {

"x": 302.0282521739266,

"y": -32.30522607035554,

"z": 389.86440214968525

},

"header": "Заголовок",

"body": "Описание."

},

],

"texture": "panorama.jpg",

"name": "название панорамы"

}

    Чтобы просматривать разные панорамы, мы будем подменять текстуру сферы, удалять точки и по координатам выстраивать новые.