INSTITUTO TECNOLOGICO DE CHIHUAHUA ll



GRAFICACIÓN 9-10 AM

PROJECT: DOCUMENTACIÓN DE EXAMEN DE FRACTALES,

ESCENA DE BURBUJAS EN 3D.

Author: Grupo Graficación 9-10

Teacher: Alonso Salcido

Fecha: 17/10/17

Introducción

Este proyecto, es un exámen que se basa en la creación de una escena en HTML5, dicha escena se compone de burbujas flotando y girando sobre su propio eje y conjuntas sobre un fondo, las cuales pueden reventarse al darles click.

A continuación se mostrará el paso a paso de la creación de nuestro exámen.

**Base del proyecto: Movimiento, animación de burbujas, sonido de las burbujas**

Primero que nada se carga el script para el uso de los controles de esta manera:

<script src="js/controls/ TrackballControls.js"></script>

Variables principales para inicializar la cámara, scene, renderer que es el encargado que las imágenes se vean, control encargado de la vista:

var camera, scene, renderer;

var controls;

scene = new THREE.Scene();

camera = new THREE.PerspectiveCamera(60, window.innerWidth / window.innerHeight, 1, 100000);

camera.position.z = 3200;

controls = new THREE.TrackballControls(camera);

for (var i = 0; i < 50; i++) {

var material = new THREE.MeshBasicMaterial({color: Math.random()});

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

sphere.position.x = Math.random() \* 200 - 100;

sphere.position.y = Math.random() \* 200 - 100;

sphere.position.z = Math.random() \* 200 - 100;

sphere.scale.x = sphere.scale.y = sphere.scale.z = Math.random() \* 3 + 1;

scene.add(sphere);

}

renderer = new THREE.WebGLRenderer();

renderer.setPixelRatio(window.devicePixelRatio);

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

renderer.autoClear = false;

document.body.appendChild(renderer.domElement);



function animate() {

controls.update();

renderer.render(scene, camera);

render();

}

function render() {

var timer = 0.0001 \* Date.now();

for (var i = 0, il = spheres.length; i < il; i++) {

var sphere = spheres[i];

sphere.position.x = 5000 \* Math.cos(timer + i);

sphere.position.y = 5000 \* Math.sin(timer + i \* 1.1);

}

renderer.render(scene, camera);

}

<audio id="audio" src="sound/burbuja3.wav" autostart="true"></audio>

function playSound() {

var sound = document.getElementById("audio");

sound.currentTime = 1;

sound.play();

}

animate();



**Fondo con mapeado cúbico**

En esta sección definimos las localizaciones de las imágenes de textura, concatenamos dentro de un arreglo de tipo String con el siguiente formato: “Directorio(textures/cube/)”+”Nombre”+”Formato(.jpg)”

var path = "textures/cube/";

var format = '.jpg';

var urls = [

path + 'px' + format, path + 'nx' + format,

path + 'py' + format, path + 'ny' + format,

path + 'pz' + format, path + 'nz' + format

];

En esta sección del código generamos el mapa cúbico de textura llamando la librería imageUtils.loadTextureCube con los argumentos del arreglo de strings que contienen las direcciones y un mapeo de refracciones:

var textureCube = THREE.ImageUtils.loadTextureCube( urls, THREE.CubeRefractionMapping );

En esta sección del código se define un sombreado de tipo fresnel en la variable Shader:

//Propiedades de la esfera

var shader = THREE.FresnelShader;

aquí se unen las texturas cúbicas con el sombreado fresnel:

var uniforms = THREE.UniformsUtils.clone( shader.uniforms );

shader.uniforms[ "tCube" ].value = textureCube;

En esta sección del código se utiliza lo anteriormente unido para crear un material y sus propiedades “físicas”:

var material = new THREE.ShaderMaterial( {

uniforms: uniforms,

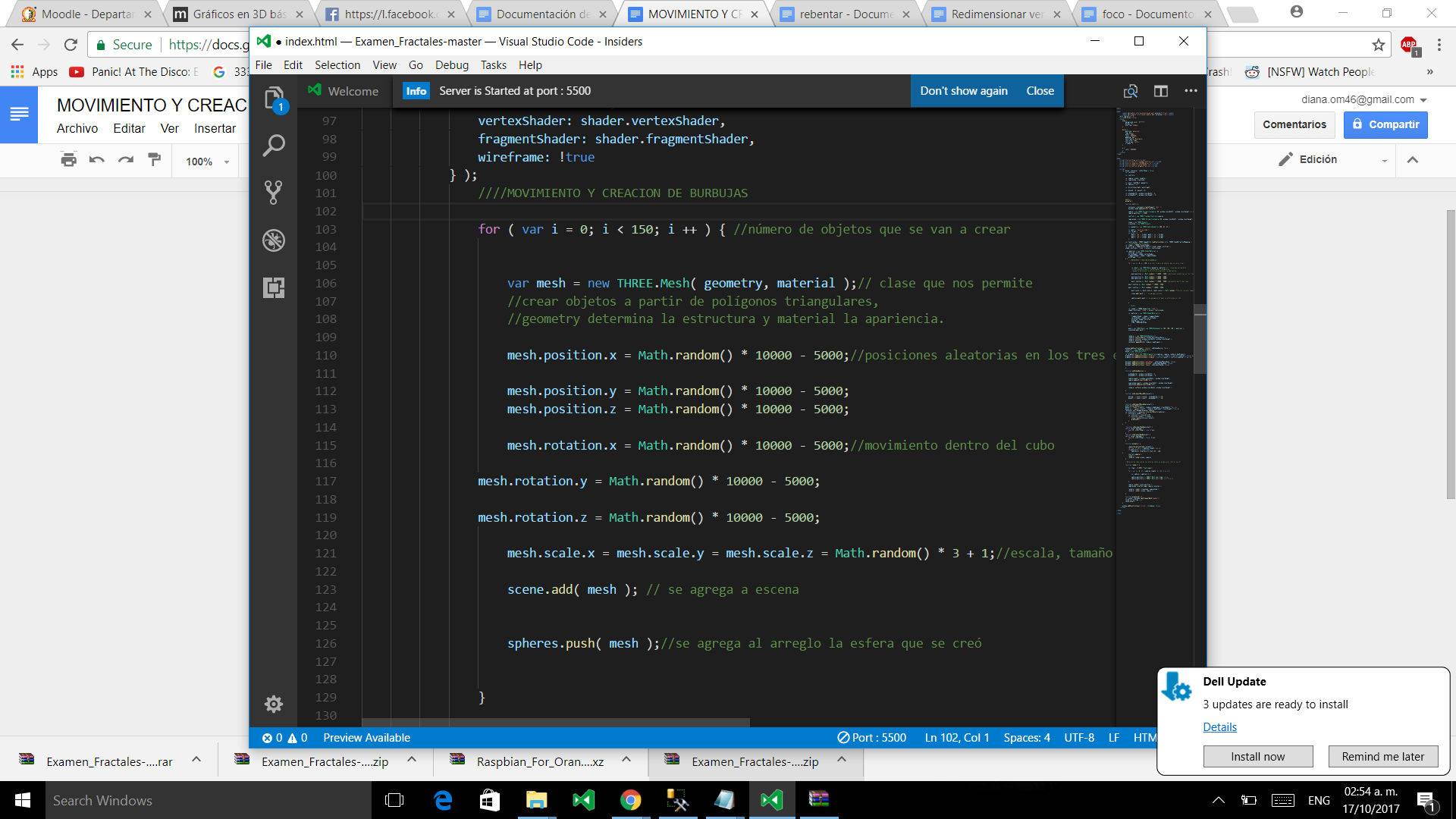
vertexShader: shader.vertexShader,

fragmentShader: shader.fragmentShader,

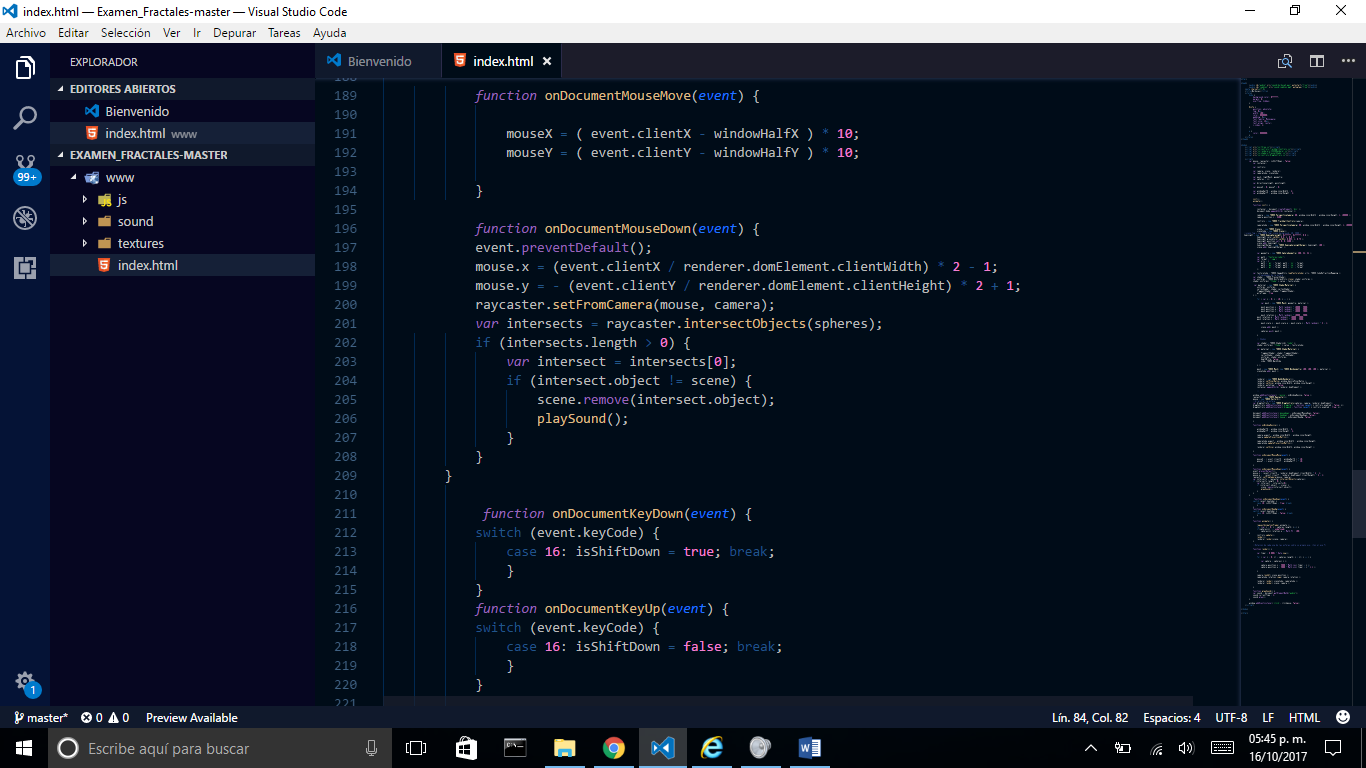
wireframe: !true

} );

**Movimiento y creación de las burbujas:**



**Función para reventar burbujas**



A continuación se crea un evento en una función onDocumentMouseDown, que detecta cuando el mouse se encuentra sobre algún componente, en este caso las burbujas generadas anteriormente.

La función nos dará las coordenadas del mouse en *mouse.x* y *mouse.y*.

Raycasting se utiliza para la selección del ratón (la elaboración de qué objetos en el espacio 3d el ratón es más), entre otras cosas.

Utilizamos el raycaster.setFromCamera(mouse, camera); para determinar qué objetos ha terminado el mouse. Si el mouse está sobre un cubo, el cubo hecho clic sufrirá una rotación progresiva y cambio de posición. El conteo también se incrementa, y nuestro elemento de puntuación también se modificará, de acuerdo con el nuevo valor de conteo.

Creamos la variable intersects y guardamos raycaster.intersectObjects(objects); que se utiliza para calcular los objetos que se cruzan con el rayo de recolección.

Comparamos que haya elementos en nuestro arreglo para hacer una intercepción e interactuar con ellos; mientras la intersección del objeto (burbuja) sea diferente de todo el grupo de objetos usaremos *remove()* para eliminar ese objeto (burbuja) interceptado del arreglo.

Con *playSound();* mandamos llamar a un sonido introducido por nosotros.

Las funciones onDocumentKeyDown y onDocumentKeyUp

Si se necesita capturar una tecla en cualquier lugar (por ejemplo, para implementar teclas de atajo globales en una página), es útil adjuntar este comportamiento al objeto document. Debido al burbujeo del evento, todas las teclas presionadas subirán al DOM al objeto document a menos que se detengan explícitamente.

El evento keyup se envía a un elemento cuando el usuario suelta una tecla en el teclado. Puede asociarse a cualquier elemento, pero el evento solo se envía al elemento que tiene el foco.

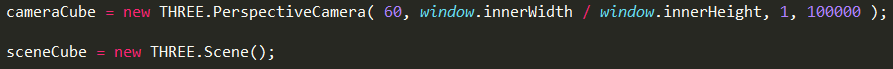
El evento keydown se envía a un elemento cuando el usuario presiona una tecla en el teclado. Si se mantiene presionada la tecla, el evento se envía cada vez que el sistema operativo repite la tecla. Puede asociarse a cualquier elemento, pero el evento solo se envía al elemento que tiene el foco.

**Redimensionar ventana y sceneCube**

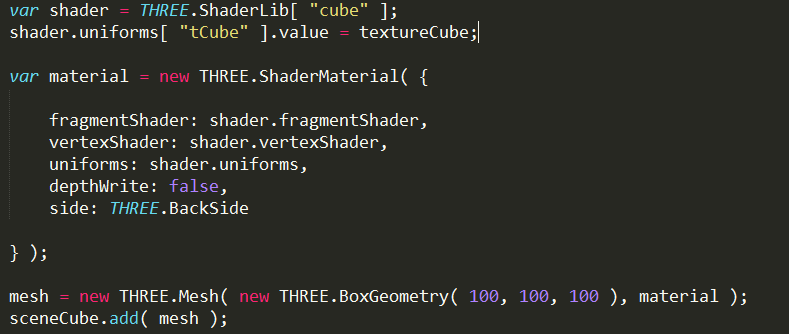
Variables sceneCube y cameraCube



Creación de la cámara y escena para la vista o perspectiva en cubo.

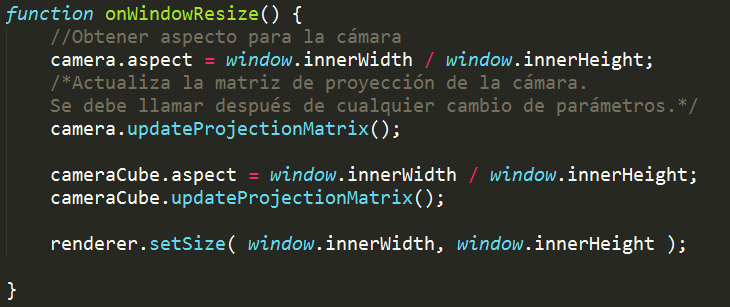


Textura del cubo. Creación de mesh, al que se le asignan las siguientes propiedades. Por último se añade a sceneCube.

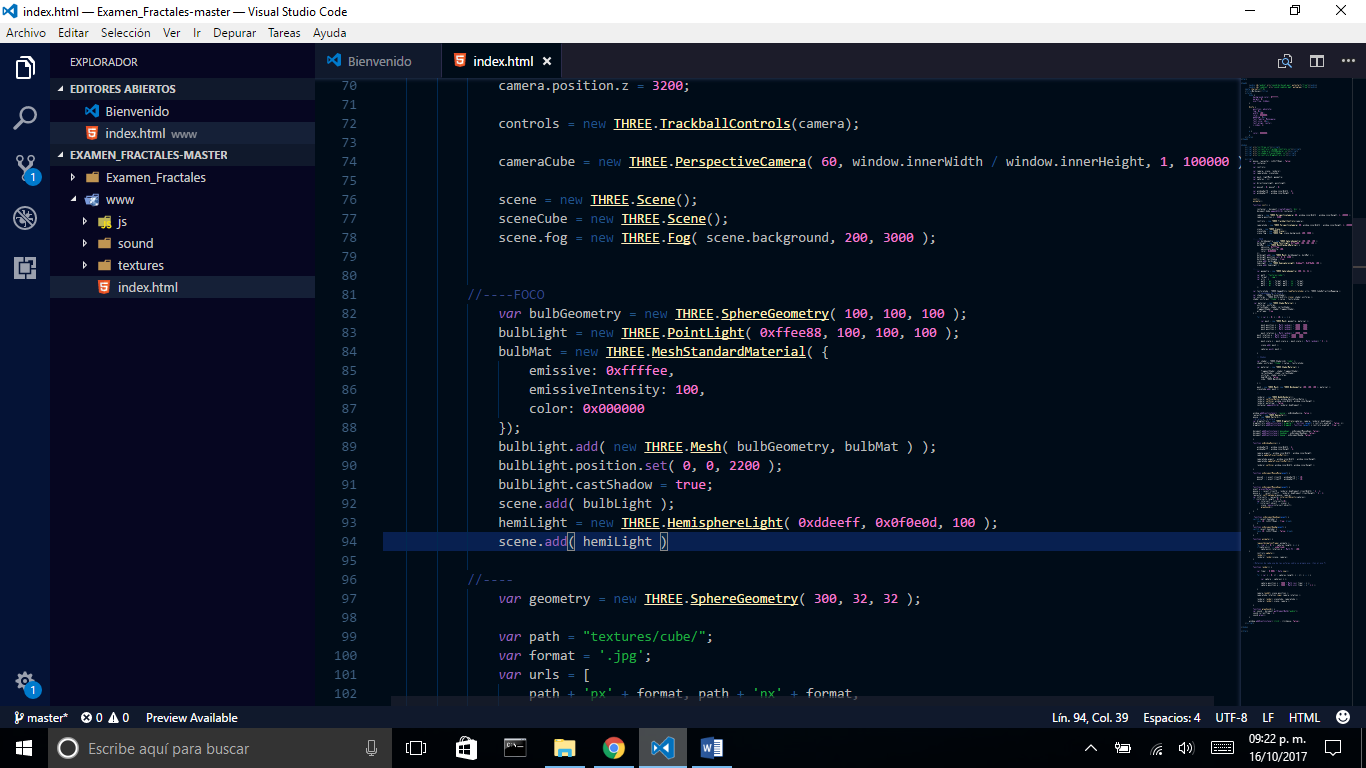


Evento en la ventana al redimensionarla, mediante la función onWindowsResize.





**Foco**



SphereGeometry(), establecerá el tamaño y polígonos totales de nuestra esfera que será una luz.

PointLight(), servirá para emitir la luz en todas las direcciones.

MeshStandardMaterial(), define el tipo de material que se quiera usar, aquí solo se usa un color, para simular el color amarillo de un foco así como la intensidad de luz que emitirá.

Le damos sombra a nuestra luz, para que haya mayor contraste y luzca más real

Ya solo se define la posición en la que se encontrara nuestro foco y lo añadimos a nuestro scene.

**Rotación de las burbujas**

//Forma del objeto

var geometry = new THREE.SphereGeometry(10, 16, 16);

for (var i = 0; i < 100; i++) {

var material = new THREE.MeshBasicMaterial( //Propiedades de la esfera

{

color: Math.random() \* 0xffff00,

opacity: .3,

transparent: true,

wireframe : true

});

Se crea el objeto (burbuja o esfera), el cual será uno solo, para luego reproducirlas con un for, para que en la pantalla se muestren menos de 100 burbujas, de esta manera se van incrementando.

var material define las propiedades del material del que están hechas las burbujas.

El wireframe nos ayuda a poligonizar nuestras figuras, de esta manera nos es más posible visualizar la animación de la rotación de cada esfera sobre su propio eje, por esta razón se asigna un true, si quisiéramos que las burbujas se vean transparentes, solo se asignaría un valor false a nuestro wireframe.

function animate() {

requestAnimationFrame(animate);

//Rotacion

for (var i = 0; i < objects.length; i++) {

if(objects[i] !== undefined)

objects[i].rotation.y += Math.PI / 100;

}

controls.update();

render();

};

En esta parte del código, donde se encuentra nuestra función de animación, implementaremos la rotación de nuestro objeto con el uso de un for, en el cual asignamos un valor inicial a la variable i, la cual será el arreglo de los clones de nuestro objeto (burbuja o esfera).

vamos a implementar la rotación de cada uno de nuestros objetos en el arreglo “i”, le estamos diciendo que gire sobre su propio eje de las y, y concatenamos el valor de math.PI, con el cual le decimos que va a ser un giro completo, de 360 grados, lo dividimos entre 100 para de esta manera definir la velocidad de la rotación de nuestros objetos.

**A continuación se muestra nuestro código, de una manera más concreta y resumida**

Entre las etiquetas head <head></head> tenemos información general (metadatos) acerca del documento, incluyendo su título y enlaces a scripts y hojas de estilos.

<head>

<audio id="audio" src="sound/burbuja3.wav" autostart="true"></audio>

<audio id="cumbion" src="sound/cumbion.mp3" autoplay="true"></audio>

<meta charset=utf-8>

<title>Burbujas</title>

<style>

body {background-color: #ffffff; margin: 0; overflow: hidden;}

#info {position: absolute; top: 0px; width: 100%;

color: #000000; padding: 5px; font-family: Monospace;

font-size: 13px; text-align: center; z-index: 1;}

a {color: #000000;}

</style>

</head>

Para controlar el reproductor de audio de nuestro documento HTML se usan las etiquetas <audio></audio>

Asignando un identificador único id para cada archivo de sonido que se utilizará, así como definir de donde se carga el elemento multimedia src="", y el evento que se realizará con el elemento multimedia autostart.

<audio id="audio" src="sound/burbuja3.wav" autostart="true"></audio>

<audio id="cumbion" src="sound/cumbion.mp3" autoplay="true"></audio>

Entre las etiquetas <title></title> se establece el título del documento “Burbujas”, el cual se muestra en la barra de título del navegador o en las pestañas de página.

<title>Burbujas</title>

Usamos las etiquetas <style></style> para indicar la información de estilo del documento.

body. Establece valores del grosor del área del margen, colores.

body { background-color: #ffffff; margin: 0; overflow: hidden;}

background-color: #ffffff; establece el color blanco como color de fondo.

margin: 0; establece un margen de cero (no hay margen)

overflow: hidden; se establece en oculto para no mostrar cualquier elemento que no quepa en pantalla y evitar el uso de barras de desplazamiento.

Especifica hasta qué punto el margen del borde superior de una caja absolutamente posicional está desplazada por debajo del borde superior del ‘bloque que contiene’ en la caja.

#info {

position: absolute; top: 0px; width: 100%;

color: #000000; padding: 5px; font-family: Monospace;

font-size: 13px; text-align: center; z-index: 1;

}

position: absolute; Si el elemento tiene márgenes, se añaden al desplazamiento.

width: 100%; abarca el 100% del espacio disponible en pantalla.

color: #000000; El fondo se torna de este color, inicialmente, y en caso de que el explorador no soporta la app.

font-family: Monospace; Especifica una lista priorizada de uno o más fonts para el elemento seleccionado. (necesario para mostrar texto, en caso de que el documento no se visualice a causa de que el explorador no lo soporte ó reconozca).

font-size: 13px; Asigna el tamaño de letra.

text-align: center; centra el texto en pantalla.

Se establece el color del texto que se muestre en pantalla. (cuando la imágen es soportada por el explorador)

a {color: #000000;}

En este caso se establece un color oscuro para contrastar con el color de nuestro background, anteriormente definido con un color claro.

Dentro de las etiquetas <body></body> tenemos el contenido de nuestro documento.

Para trabajar con nuestro documento usamos etiquetas <script></script> para insertar código de JavaScript que nos permitirá establecer el comportamiento de nuestra página web.

<body>

<script src="js/three.js"></script>

<script src="js/controls/TrackballControls.js"></script>

<script src="js/renderers/CSS3DRenderer.js"></script>

<script src="js/shaders/FresnelShader.js"></script>

<script src="js/controls/DragControls.js"></script>

<script>

var mouse, raycaster, isShiftDown = false;

var container,controls;

var camera, scene, renderer;

var cameraCube, sceneCube;

var mesh, lightMesh, geometry;

var spheres = [];

var directionalLight, pointLight;

var mouseX = 0, mouseY = 0;

var windowHalfX = window.innerWidth / 2;

var windowHalfY = window.innerHeight / 2;

init();

animate();

function init() {

container = document.createElement( 'div' );

document.body.appendChild( container );

camera = new THREE.PerspectiveCamera( 60, window.innerWidth / window.innerHeight, 1, 100000 );

camera.position.z = 3200;

controls = new THREE.TrackballControls(camera);

cameraCube = new THREE.PerspectiveCamera( 60, window.innerWidth / window.innerHeight, 1, 100000 );

scene = new THREE.Scene();

sceneCube = new THREE.Scene();

var geometry = new THREE.SphereGeometry( 300, 32, 32 );

var path = "textures/cube/";

var format = '.jpg';

var urls = [

path + 'px' + format, path + 'nx' + format,

path + 'py' + format, path + 'ny' + format,

path + 'pz' + format, path + 'nz' + format

];

var textureCube = THREE.ImageUtils.loadTextureCube( urls, THREE.CubeRefractionMapping );

**//Propiedades de la esfera**

var shader = THREE.FresnelShader;

var uniforms = THREE.UniformsUtils.clone( shader.uniforms );

shader.uniforms[ "tCube" ].value = textureCube;

var material = new THREE.ShaderMaterial( {

uniforms: uniforms,

vertexShader: shader.vertexShader,

fragmentShader: shader.fragmentShader,

wireframe: !true **//es para visualizar los poligonos en cada Burbuja**

} );

for ( var i = 0; i < 150; i ++ ) {

var mesh = new THREE.Mesh( geometry, material );

mesh.position.x = Math.random() \* 10000 - 5000;

mesh.position.y = Math.random() \* 10000 - 5000;

mesh.position.z = Math.random() \* 10000 - 5000;

mesh.rotation.x = Math.random() \* 10000 - 5000;

mesh.rotation.y = Math.random() \* 10000 - 5000;

mesh.rotation.z = Math.random() \* 10000 - 5000;

mesh.scale.x = mesh.scale.y = mesh.scale.z = Math.random() \* 3 + 1;

scene.add( mesh );

spheres.push( mesh );

}

// Skybox

var shader = THREE.ShaderLib[ "cube" ];

shader.uniforms[ "tCube" ].value = textureCube;

var material = new THREE.ShaderMaterial( {

fragmentShader: shader.fragmentShader,

vertexShader: shader.vertexShader,

uniforms: shader.uniforms,

depthWrite: false,

side: THREE.BackSide

} );

mesh = new THREE.Mesh( new THREE.BoxGeometry( 100, 100, 100 ), material );

sceneCube.add( mesh );

//

renderer = new THREE.WebGLRenderer();

renderer.setPixelRatio( window.devicePixelRatio );

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

renderer.autoClear = false;

container.appendChild( renderer.domElement );

//

window.addEventListener( 'resize', onWindowResize, false );

raycaster = new THREE.Raycaster();

mouse = new THREE.Vector2();

**//Arrastrar la esfera**

var dragControls = new THREE.DragControls(spheres, camera, renderer.domElement);

dragControls.addEventListener('dragstart', function (event) { controls.enabled = false; });

dragControls.addEventListener('dragend', function (event) { controls.enabled = true; });

document.addEventListener('mousedown', onDocumentMouseDown, false);

document.addEventListener('keydown', onDocumentKeyDown, false);

document.addEventListener('keyup', onDocumentKeyUp, false);

}

function onWindowResize() {

camera.aspect = window.innerWidth / window.innerHeight;

camera.updateProjectionMatrix();

cameraCube.aspect = window.innerWidth / window.innerHeight;

cameraCube.updateProjectionMatrix();

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

}

function onDocumentMouseMove(event) {

mouseX = ( event.clientX - windowHalfX ) \* 10;

mouseY = ( event.clientY - windowHalfY ) \* 10;

}

function onDocumentMouseDown(event) {

event.preventDefault();

mouse.x = (event.clientX / renderer.domElement.clientWidth) \* 2 - 1;

mouse.y = - (event.clientY / renderer.domElement.clientHeight) \* 2 + 1;

raycaster.setFromCamera(mouse, camera);

var intersects = raycaster.intersectObjects(spheres);

if (intersects.length > 0) {

var intersect = intersects[0];

if (intersect.object != scene) {

scene.remove(intersect.object);

playSound();

}

}

}

function onDocumentKeyDown(event) {

switch (event.keyCode) {

case 16: isShiftDown = true; break;

}

}

function onDocumentKeyUp(event) {

switch (event.keyCode) {

case 16: isShiftDown = false; break;

}

}

function animate() {

requestAnimationFrame( animate );

/\*En esta parte de código podemos hacer rotaciones y visualizar quitando el ! de (wireframe: !true) donde creamos una animación para los clones de nuestra esfera dándole el redibujado dentro de su propio eje, dando también velocidad al redibujado para que simule el efecto de rotación dándole un movimiento más natural a las burbujas e indefinido para que no deje de rotar.\*/

for (var i = 0; i < spheres.length; i++) {

if(spheres[i] !== undefined)

spheres[i].rotation.y += Math.PI / 100;

}

controls.update();

render();

renderer.render(scene, camera);

}

function render() {

var timer = 0.0001 \* Date.now();

for ( var i = 0, il = spheres.length; i < il; i ++ ) {

var sphere = spheres[ i ];

sphere.position.x = 5000 \* Math.cos( timer + i );

sphere.position.y = 5000 \* Math.sin( timer + i \* 1.1 );

}

camera.lookAt( scene.position );

cameraCube.rotation.copy( camera.rotation );

renderer.render( sceneCube, cameraCube );

renderer.render( scene, camera );

}

function playSound() {

var sound = document.getElementById("audio");

sound.currentTime = 1;

sound.play();

}

window.addEventListener('click', clickmose, false);

</script>

</body>

Para añadir las librerías necesarias a nuestro documento html, necesitamos hacer referencia a ellas de la siguiente manera:

<script src=""></script>

De esta manera podremos hacer uso de la librería three.js

<script src="js/three.js"></script>

Para renderizar nuestros elementos CSS en un contexto 3D

<script src="js/renderers/CSS3DRenderer.js"></script>

Para cargar efectos de sombreado

<script src="js/shaders/FresnelShader.js"></script>

Para arrastrar las burbujas.

<script src="js/controls/DragControls.js"></script>

Hasta ahora solo hemos cargado librerías a nuestro documento, por lo tanto necesitamos un nuevo script para configurar comportamientos.

En la estructura del Script mantendremos las variables globales que necesitemos sobre la marcha en la parte superior del código justo debajo de donde abre la etiqueta <script>

var mouse, raycaster, isShiftDown = false;

var container;

var controls;

var camera, scene, renderer;

var cameraCube, sceneCube;

var mesh, lightMesh, geometry;

var spheres = [];

var directionalLight, pointLight;

var mouseX = 0, mouseY = 0;

var windowHalfX = window.innerWidth / 2;

var windowHalfY = window.innerHeight / 2;

var Shperes = []; Se define un arreglo el cual contendrá nuestras esferas.

Se mandan llamar inmediatamente las funciones init() y animate(), para mostrar en pantalla la escena de nuestro documento, las cuales provocan el efecto en cadena de los demás comportamientos.

init();

animate();

**función init()**

Se define el valor de nuestra variable container, como un nuevo document el cual llama al método createElement() el cual recibe el nombre 'div'

container = document.createElement( 'div' );

Una vez creado el elemento se usa el método appendChild() para insertarlo en el documento.

El valor de la variable camera se inicializa en un nuevo objeto THREE y llama a la propiedad PerspectiveCamera(), para definir el comportamiento de la cámara.

camera = new THREE.PerspectiveCamera(60,window.innerWidth/window.innerHeight,1,100000);

la variable controls se inicializa en un nuevo objeto THREE para establecer la propiedad TrackballControls() , que modificara el comportamiento de la cámara, cada vez que se desplace.

controls = new THREE.TrackballControls(camera);

Para las texturas de las burbujas y del fondo se establece la variable path, la cual contendrá la imagen definida para el fondo de la escena, y en este caso también se usará para agregar un detalle reflejante a las burbujas.

var path = "textures/cube/";

Dado que es necesario establecer el formato del elemento multimedia que se aplique en cada ocasión, se establece este formato en una nueva variable format para agilizar la escritura del código.

var format = '.jpg';

La burbuja se compone de de 6 paredes, por ello se debe cargar una imagen en cada pared, para que el efecto sea visible desde cualquier ángulo en que se visualice, cada imagen se cargará en un arreglo dentro de la variable urls.

var urls = [

path + 'px' + format, path + 'nx' + format,

path + 'py' + format, path + 'ny' + format,

path + 'pz' + format, path + 'nz' + format

];

Para cargar estas imágenes en escena es necesario definir una nueva variable la cual contendrá las texturas del cubo.

var textureCube = THREE.ImageUtils.loadTextureCube(urls,THREE.CubeRefractionMapping);

**Propiedades de la esfera**

En está sección se define un sombreador de tipo fresnel en la variable Shader

var shader = THREE.FresnelShader;

Luego para unir las texturas cúbicas con el sombreador fresnel

var uniforms = THREE.UniformsUtils.clone( shader.uniforms );

shader.uniforms[ "tCube" ].value = textureCube;

Haciendo uso de lo que creamos con las dos líneas anteriores, creamos un material y sus propiedades físicas

var material = new THREE.ShaderMaterial( {

uniforms: uniforms,

vertexShader: shader.vertexShader,

fragmentShader: shader.fragmentShader,

wireframe: !true

} );

Para mostrar los polígonos de nuestras burbujas se usa esta linea de código. Esto es opcional por lo tanto permanecerá negado.

wireframe: !true;

**Movimiento y creación de burbujas**

Para establecer el número de objetos que se van a crear, se utilizó un ciclo for

for ( var i = 0; i < 150; i ++ ) {

var mesh = new THREE.Mesh( geometry, material );

mesh.position.x = Math.random() \* 10000 - 5000;

mesh.position.y = Math.random() \* 10000 - 5000;

mesh.position.z = Math.random() \* 10000 - 5000;

mesh.rotation.x = Math.random() \* 10000 - 5000;

mesh.rotation.y = Math.random() \* 10000 - 5000;

mesh.rotation.z = Math.random() \* 10000 - 5000;

mesh.scale.x = mesh.scale.y = mesh.scale.z = Math.random() \* 3 + 1;

scene.add( mesh );

spheres.push( mesh );

}

La clase que nos permite crear objetos a partir de polígonos triangulares es geometry determina la estructura y la clase material determina la apariencia.

Para que los objetos tengan posiciones aleatorias en los tres ejes (x, y , z)

mesh.position.x = Math.random() \* 10000 - 5000;

mesh.position.y = Math.random() \* 10000 - 5000;

mesh.position.z = Math.random() \* 10000 - 5000;

Luego para que estos objetos generados se muevan dentro del cubo implementamos:

mesh.rotation.x = Math.random() \* 10000 - 5000;

mesh.rotation.y = Math.random() \* 10000 - 5000;

mesh.rotation.z = Math.random() \* 10000 - 5000;

La escala y tamaño se definen:

mesh.scale.x = mesh.scale.y = mesh.scale.z = Math.random() \* 3 + 1;

Todo esto se agrega a escena con scene.add( mesh );

Cada esfera que se generó se agrega al arreglo con la siguiente línea:

spheres.push( mesh );

**función onWindowResize()**

**función onDocumentMouseMove()**

**función onDocumentMouseDown()**

**función onDocumentKeyDown()**

**función onDoucmentKeyUp()**

**función animate()**

…

Para rotar las esferas hacemos uso de un ciclo for, el cual inicia en 0 y se ejecuta tantas veces como esferas hayamos generado

...

**función render()**

function render() {

var timer = 0.0001 \* Date.now();

for ( var i = 0, il = spheres.length; i < il; i ++ ) {

var sphere = spheres[ i ];

sphere.position.x = 5000 \* Math.cos( timer + i );

sphere.position.y = 5000 \* Math.sin( timer + i \* 1.1 );

}

camera.lookAt( scene.position );

cameraCube.rotation.copy( camera.rotation );

renderer.render(sceneCube,cameraCube );

renderer.render(scene,camera);

}

Dentro de la función render() se incluye el efecto de rotación de cada una de las esferas con respecto al eje y.

función playSound()

function playSound() {

var sound = document.getElementById("audio");

sound.currentTime = 1;

sound.play();

}

Se define la función para establecer el sonido que se reproducirá cada vez que se haga clic sobre el documento html. Para ello se establece una variable local var sound que obtendrá el valor del identificador que usamos para cargar previamente nuestro archivo multimedia id="audio"

También podremos asignar establecer el tiempo que se reproducirá el sonido sound.currentTime = 1;

Se define un evento para activar la funcionalidad del “click” para detectar cuando sea presionado.

window.addEventListener('click', clickmose, false);

Al final, nuestro paisaje se puede apreciar de esta manera:

