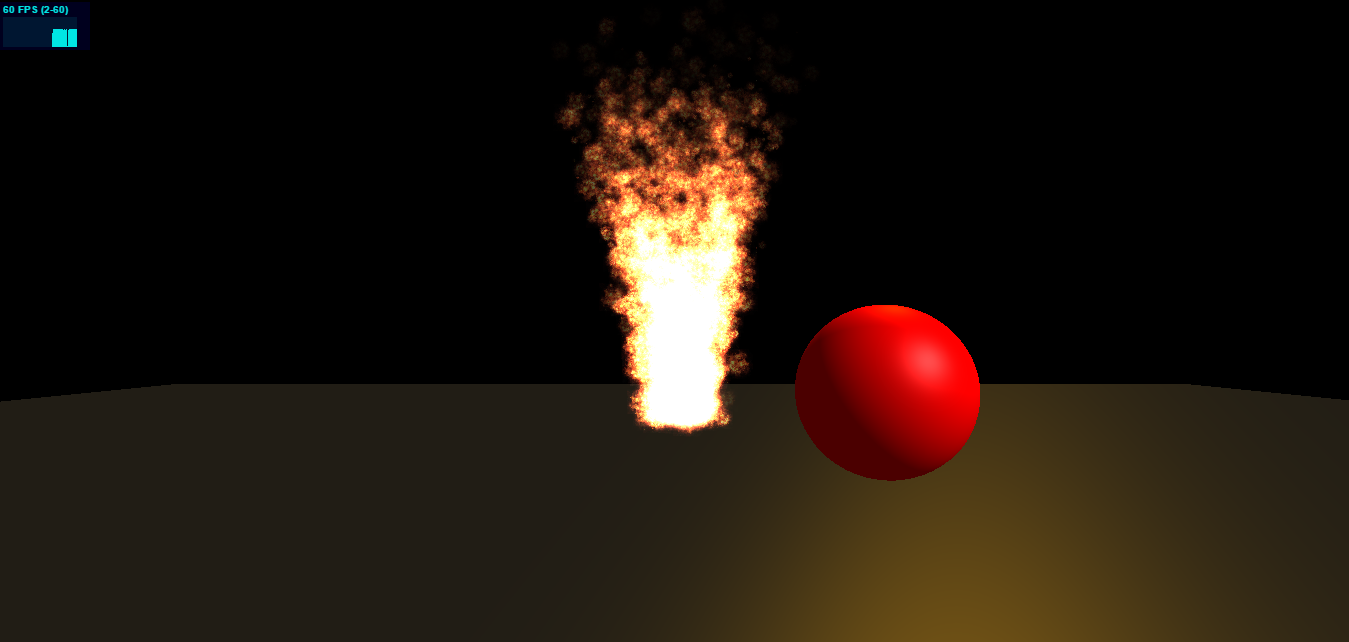
[실전 예제 : Fire Ball]



**[실전 예제 “Fire Ball”의 결과물 : Plane위의 붉은 구체와 불타는 객체]**

***★Tutorial2와 Tutorial2\_1을 꼭 참고해주세요! – 설명을 자세히 하지 않는 부분은 이전에 설명을 했던 부분입니다.***

**▶ 타오르는 불 객체를 만들기 위해서 다음과 같은 순서로 진행을 할 것입니다.**

1. 필요한 js라이브러리 연결시키기
2. 왼쪽 상단의 FPS(Frame Per Second)측정 통계치 만들기
3. Init( ) 함수 선언 및 정의
4. addShadowedLight( ) 함수 선언 및 정의
5. animate( ) 함수 선언 및 정의
6. 필요 함수 실행
7. 전체 코드 공개.
8. 만들어진 최종 결과물 객체 확인하기

*(부분적인 소스코드를 설명하며 마지막에 전체 코드가 첨부됩니다. 각 Step별로 추가된 소스코드에 집중해주세요!)*

**Step1. 필요한 js라이브러리 연결시키기**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 이번 Fire Ball예제를 수행하기 위하여 필요한 라이브러리는 총9가지이다.   1. three.min.js 2. TrackballControls.js 3. SphereObject.js 4. PlaneObject.js 5. Grid.js 6. ObjectField.js 7. ParticleSystem.js 8. Stat.js 9. gui.min.js   Three.js라이브러리를 사용하기위하여 three.min.js와 추가적인 8가지가 있다.  마우스를 사용하여 화면을 control할 TrackballControls.js가 있고, 구체와 평면을 구현할 SphereObject.js와 PlaneObject.js가 있다. 또한 불을 표현하기 위해 가장 중요한 ParticleSystem.js도 있다. 이 밖의 추가적인 기능 수행을 위해 필요한 라이브러리가 있다.  <코드 1>의 6~17번째줄의 부분이 상대경로를 이용한 js라이브러리를 연결시키는 부분이다. | |  |  |  | | --- | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>Fire Ball</title>      <script src="./three.js-master/build/three.min.js"></script>      <script src="./three.js-master/examples/js/controls/TrackballControls.js"></script>        <script src="./SphereObject.js"></script>      <script src="./PlaneObject.js"></script>      <script src="./Grid.js"></script>      <script src="./ObjectField.js"></script>        <script src="./ParticleSystem.js"></script>      <script src="./Lab/WEB3D/VFXLib/lib/Stats.js"></script>        <script src="./dat.gui.min.js"></script>  </head>  <body>[*r*](http://colorscripter.com/info#e) |  | |

**<코드1>**

**Step2. 왼쪽 상단의 FPS(Frame Per Second)측정 통계치 만들기**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <코드 2>의 24~31번째 줄까지는 최종 결과물들을 생성하기 위하여 필요한 변수들을 전역으로 생성하여 주는 코드이다.  Line 24 : clock변수는 Three.js라이브러리의 THREE.Clock( )을 사용하여 생성하였다. 이 clock변수는 이후 getDelta( )라는 함수를 사용하여 재귀호출시 경과한 시간을 가져오는 역할을 할것이다.  Line 26 : FPS측정 통계치를 만들기 위한 stats변수를 선언한다.  Line 27~30 : 생성된 stats의 위치와 크기를 설정한다.   * Position : 절대위치로서 absolute로 설정한다. * Left : 왼쪽에서 10픽셀 거리를 둔다. * Top : 위쪽에서 10픽셀 거리를 둔다. * Width : 너비는 90픽셀로 만든다.   Line 31 : 만들어진 stats을 HTML문서에 붙인다. | |  |  |  | | --- | --- | --- | | 20  21  22  23  24  25  26  27  28  29  30  31  32  33 | <script>      //noise.seed(Math.random());        var clock = new THREE.Clock();        var stats = new Stats();//왼쪽상단의 FPS측정 통계치      stats.domElement.style.position = 'absolute';      stats.domElement.style.left = '10px';      stats.domElement.style.top = '10px';      stats.domElement.style.width = '90px'      document.body.appendChild(stats.domElement); |  | |

**<코드2>**

**Step3. Init( ) 함수 선언 및 정의**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <코드 3>의 34~36번째 줄은 Three.js의 기본적인 객체들을 선언하는 부분이다.  <코드 3>의 48~123번째 줄은 init( )함수의 선언 및 정의 부분이다.  \*설명하지 않은 부분은 Tutorial2와 Tutorial2\_1을 꼭 참고해주세요!  Line 80~81 : 구체 Mesh객체를 생성하기 위하여 필요한 geometry와 material객체를 설정하고 만든다.  Line 83~85 : sphereMesh객체를 위에서 만든 두 객체로 선언하고 만든다. 그리고 x, y위치를 설정한다.  Line 87~92 : 평면 객체인 plane객체를 만든다. sphere객체와 같이 필요한 geometry와 material객체를 만들어 사용한다. 또한 위치를 설정해준다.  Line 93~95 : 생성한 plane과 sphereMesh객체를 scene에 붙인다.  Line 98~101 : Three.js의 main 객체인 Light객체를 선언하고 scene에 붙인다. 또한 addShadowedLight를 통해 빛의 방향과 색을 정한다. Light가 있어야 생성한 plane과 sphereMesh가 우리눈에 보일 수 있다.  \*addShadowedLight함수에 대해서는 Step4에서 설명 할 예정이다. | |  |  |  | | --- | --- | --- | | 34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123 | var scene, camera, renderer;      var controls;      var particleSystem;    /\*    var gui, ParamConfig = {          seedVelDir : new THREE.Vector3(0.7, -2, 0),          seedVelMag : 500.0,          seedLife : 1000,          seedSize : 200,          seedSpread : 0.15,          colorBase : new THREE.Color(0Xfffafa),          tex : THREE.ImageUtils.loadTexture("./textures/snowflake.png")      };\*/        function init() {            renderer = new THREE.WebGLRenderer();          renderer.setSize(window.innerWidth, window.innerHeight);          renderer.setClearColor(0x000000);            document.body.appendChild(renderer.domElement);            camera = new THREE.PerspectiveCamera(60, window.innerWidth / window.innerHeight, 0.1, 100000 );          camera.position.z = 2000;            controls = new THREE.TrackballControls( camera );            controls.rotateSpeed = 10.0;          controls.zoomSpeed = 5;          controls.panSpeed = 2;            controls.noZoom = false;          controls.noPan = false;            controls.staticMoving = true;          controls.dynamicDampingFactor = 0.3;            scene = new THREE.Scene();            window.addEventListener( 'resize', onWindowResize, false );            particleSystem = new ParticleSystem();          particleSystem.initialize(100000);            scene.add(particleSystem.getMesh());            var sphereGeometry = new THREE.SphereGeometry( 300, 32, 32 );          var sphereMaterial = new THREE.MeshPhongMaterial( { color: 0xaa00000, specular: 0x111111 } );            var sphereMesh = new THREE.Mesh(sphereGeometry, sphereMaterial);          sphereMesh.position.x = 700;          sphereMesh.position.y = -200;            var plane = new THREE.Mesh(                  new THREE.PlaneBufferGeometry( 30000, 30000 ),                  new THREE.MeshPhongMaterial( { color: 0x111111, specular: 0x222222 } )          );          plane.rotation.x = -Math.PI/2;          plane.position.y = -1500;          scene.add( plane );            scene.add(sphereMesh);            // Lights          scene.add( new THREE.AmbientLight( 0x777777 ) );            addShadowedLight( 1, 1, 1, 0xffffff, 1.35 );          addShadowedLight( 0.5, 1, -1, 0xffaa00, 1 );    //        gui = new dat.GUI();  //        gui.add(ParamConfig, 'seedVelMag', 200, 1000).onChange(function(value){              var velMag = {seedVelMag : 450};              particleSystem.setParameters(velMag);  //        });    //        gui.add(ParamConfig, 'seedLife', 1, 2000).onChange(function(value){              var life = {seedLife : 3};              particleSystem.setParameters(life);  //        });    //        gui.add(ParamConfig, 'seedSize', 100, 1000).onChange(function(value){              var seed = {seedSize : 200};              particleSystem.setParameters(seed);  //        });    //        gui.add(ParamConfig, 'seedSpread', 0.1, 4.0).onChange(function(value){              var spread = {seedSpread : 0.35};              particleSystem.setParameters(spread);  //        });      } |  | |

**<코드3>**

**Step4. addShadowedLight( ) 함수 선언 및 정의**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <코드 4>의 125~149번째 줄은 <코드 3>의 init( )함수에서 사용한 addShadowedLight( )함수에 대한 정의 부분이다.  Line 125 : addShadowedLight함수는 매개변수로 5가지를 받는다.   * X : 절대위치의 x축 값을 의미한다. * Y : 절대위치의 y축 값을 의미한다. * Z : 절대위치의 z축 값을 의미한다. * Color : 빛의 색상을 의미하며 RGB값으로 표현한다. * Intensity : 빛의 강도를 의미한다.   Line 127~129 : directionLight라는 빛의 방향성을 결정할 객체를 선언한다. 객체의 위치를 x, y, z로 설정을 한 뒤 scene에 붙인다.  Line 131 : 앞서 만든 directionLight라는 객체에 그림자효과를 true로 설정한다.  Line 134 : 1과 -1로 사용할 변수 d선언  Line 135~138 : 각 좌, 우, 위, 아래의 카메라가 그림자를 표현하는 수치를 1과 -1로 조절한다.  Line 140~141 : 카메라가 그림자를 식별하는 거리감을 near와 far로 조절한다.  Line 143~144 : 픽셀적 수치에 있어 그림자 맵의 텍스쳐 너비와 높이를 설정한다.  Line 146~147 : 그림자의 표면을 결정할 때, 표준화된 깊이로부터 얼마의 수치만큼 빼고 더할 것인지를 정하는 Bias와 그림자의 어두운 정도를 표현하는 Darkness를 설정한다. | |  |  |  | | --- | --- | --- | | 124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149 | function addShadowedLight( x, y, z, color, intensity ) {            var directionalLight = new THREE.DirectionalLight( color, intensity );          directionalLight.position.set( x, y, z )          scene.add( directionalLight );            directionalLight.castShadow = true;          //directionalLight.shadowCameraVisible = true;            var d = 1;          directionalLight.shadowCameraLeft = -d;          directionalLight.shadowCameraRight = d;          directionalLight.shadowCameraTop = d;          directionalLight.shadowCameraBottom = -d;            directionalLight.shadowCameraNear = 1;          directionalLight.shadowCameraFar = 4;            directionalLight.shadowMapWidth = 1024;          directionalLight.shadowMapHeight = 1024;            directionalLight.shadowBias = -0.005;          directionalLight.shadowDarkness = 0.15;        } |  | |

**<코드4>**

**Step5. animate( ) 함수 선언 및 정의**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <코드 5>의 151~159번째 줄은 window창을 변경했을 때 다시 렌더링을 해주는 onWindowResize( )함수에 대한 부분이다.  \*설명하지 않은 부분은 Tutorial2와 Tutorial2\_1을 꼭 참고해주세요!  <코드 5>의 161~184번째 줄은 animate( )함수에 대한 부분이다.  Line 163 : FPS측정 통계치로 만든 stats객체를 시작 시킨다.  Line 181 : FPS측정 통계치를 animate( )함수가 종료되기 직전에 종료시킨다. animate( )함수는 계속해서 재귀적으로 호출이 되기 때문에 이화 함께 stats객체도 시작과 종료를 반복하며 측정하게 된다. | |  |  |  | | --- | --- | --- | | 150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184 | function onWindowResize() {            camera.aspect = window.innerWidth / window.innerHeight;          camera.updateProjectionMatrix();            renderer.setSize( window.innerWidth, window.innerHeight );            controls.handleResize();      }        function animate() {            stats.begin();            var cen = new THREE.Vector3(0,-300,0);          var nor = new THREE.Vector3(0,1,0);          var rad = 150;            //particleSystem.addParticlesFromSphere(100, cen, rad);          particleSystem.addParticlesFromDisk(50, cen, nor, rad);              particleSystem.updateParticles(clock.getDelta());            controls.update();            renderer.clear();          renderer.clearDepth();          renderer.render( scene, camera );            stats.end();            requestAnimationFrame( animate );      } |  | |

**<코드5>**

**Step6. 필요 함수 실행**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <코드 6>의 나머지는 실행에 필요한 함수를 실행하는 부분과 태그를 닫는 부분이다.  Line 185~186 : 위에서 만든 init( )함수와 animate( )함수를 실행하는 부분이다. Animate( )함수는 함수내의 requestAnimationFrame함수를 통해 재귀적으로 계속 호출될 것이다. | |  |  |  | | --- | --- | --- | | 185  186  187  188  189  190  191 | init();      animate();    </script>    </body>  </html> |  | |

**<코드6>**

**Step7. 전체 코드 공개**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96 | <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>Fire Ball</title>      <script src="./three.js-master/build/three.min.js"></script>      <script src="./three.js-master/examples/js/controls/TrackballControls.js"></script>        <script src="./SphereObject.js"></script>      <script src="./PlaneObject.js"></script>      <script src="./Grid.js"></script>      <script src="./ObjectField.js"></script>        <script src="./ParticleSystem.js"></script>      <script src="./Lab/WEB3D/VFXLib/lib/Stats.js"></script>        <script src="./dat.gui.min.js"></script>  </head>  <body>    <script>      //noise.seed(Math.random());        var clock = new THREE.Clock();        var stats = new Stats();//왼쪽상단의 FPS측정 통계치      stats.domElement.style.position = 'absolute';      stats.domElement.style.left = '10px';      stats.domElement.style.top = '10px';      stats.domElement.style.width = '90px'      document.body.appendChild(stats.domElement);          var scene, camera, renderer;      var controls;      var particleSystem;    /\*    var gui, ParamConfig = {          seedVelDir : new THREE.Vector3(0.7, -2, 0),          seedVelMag : 500.0,          seedLife : 1000,          seedSize : 200,          seedSpread : 0.15,          colorBase : new THREE.Color(0Xfffafa),          tex : THREE.ImageUtils.loadTexture("./textures/snowflake.png")      };\*/        function init() {            renderer = new THREE.WebGLRenderer();          renderer.setSize(window.innerWidth, window.innerHeight);          renderer.setClearColor(0x000000);            document.body.appendChild(renderer.domElement);            camera = new THREE.PerspectiveCamera(60, window.innerWidth / window.innerHeight, 0.1, 100000 );          camera.position.z = 2000;            controls = new THREE.TrackballControls( camera );            controls.rotateSpeed = 10.0;          controls.zoomSpeed = 5;          controls.panSpeed = 2;            controls.noZoom = false;          controls.noPan = false;            controls.staticMoving = true;          controls.dynamicDampingFactor = 0.3;            scene = new THREE.Scene();            window.addEventListener( 'resize', onWindowResize, false );            particleSystem = new ParticleSystem();          particleSystem.initialize(100000);            scene.add(particleSystem.getMesh());            var sphereGeometry = new THREE.SphereGeometry( 300, 32, 32 );          var sphereMaterial = new THREE.MeshPhongMaterial( { color: 0xaa00000, specular: 0x111111 } );            var sphereMesh = new THREE.Mesh(sphereGeometry, sphereMaterial);          sphereMesh.position.x = 700;          sphereMesh.position.y = -200;            var plane = new THREE.Mesh(                  new THREE.PlaneBufferGeometry( 30000, 30000 ),                  new THREE.MeshPhongMaterial( { color: 0x111111, specular: 0x222222 } )          );          plane.rotation.x = -Math.PI/2;          plane.position.y = -1500;          scene.add( plane );            scene.add(sphereMesh); |  | | |  |  |  | | --- | --- | --- | | 97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191 | // Lights          scene.add( new THREE.AmbientLight( 0x777777 ) );            addShadowedLight( 1, 1, 1, 0xffffff, 1.35 );          addShadowedLight( 0.5, 1, -1, 0xffaa00, 1 );    //        gui = new dat.GUI();  //        gui.add(ParamConfig, 'seedVelMag', 200, 1000).onChange(function(value){              var velMag = {seedVelMag : 450};              particleSystem.setParameters(velMag);  //        });    //        gui.add(ParamConfig, 'seedLife', 1, 2000).onChange(function(value){              var life = {seedLife : 3};              particleSystem.setParameters(life);  //        });    //        gui.add(ParamConfig, 'seedSize', 100, 1000).onChange(function(value){              var seed = {seedSize : 200};              particleSystem.setParameters(seed);  //        });    //        gui.add(ParamConfig, 'seedSpread', 0.1, 4.0).onChange(function(value){              var spread = {seedSpread : 0.35};              particleSystem.setParameters(spread);  //        });      }        function addShadowedLight( x, y, z, color, intensity ) {            var directionalLight = new THREE.DirectionalLight( color, intensity );          directionalLight.position.set( x, y, z )          scene.add( directionalLight );            directionalLight.castShadow = true;          //directionalLight.shadowCameraVisible = true;            var d = 1;          directionalLight.shadowCameraLeft = -d;          directionalLight.shadowCameraRight = d;          directionalLight.shadowCameraTop = d;          directionalLight.shadowCameraBottom = -d;            directionalLight.shadowCameraNear = 1;          directionalLight.shadowCameraFar = 4;            directionalLight.shadowMapWidth = 1024;          directionalLight.shadowMapHeight = 1024;            directionalLight.shadowBias = -0.005;          directionalLight.shadowDarkness = 0.15;        }        function onWindowResize() {            camera.aspect = window.innerWidth / window.innerHeight;          camera.updateProjectionMatrix();            renderer.setSize( window.innerWidth, window.innerHeight );            controls.handleResize();      }        function animate() {            stats.begin();            var cen = new THREE.Vector3(0,-300,0);          var nor = new THREE.Vector3(0,1,0);          var rad = 150;            //particleSystem.addParticlesFromSphere(100, cen, rad);          particleSystem.addParticlesFromDisk(50, cen, nor, rad);              particleSystem.updateParticles(clock.getDelta());            controls.update();            renderer.clear();          renderer.clearDepth();          renderer.render( scene, camera );            stats.end();            requestAnimationFrame( animate );      }      init();      animate();    </script>    </body>  </html> |  | |

**<코드7>**

**Step8. 만들어진 최종 결과물 객체 확인하기**

(자~ 오늘은 실전 예제 “Fire Ball”을 배워 보았습니다! 역시 결과물을 확인해 봐야겠죠?)

▶ 실전 예제 Fire Ball입니다. 자세히 설명되지 않은 부분은 꼭 앞부분 Tutorial들을 공부하고 난 뒤 이해하고 넘어 갑시다!

* “Fire Ball” 결과물 링크 : <https://vimeo.com/154476303>

***다음 번에는 더 재미있는 실전 예제로 실습해봅시다 ~ !***