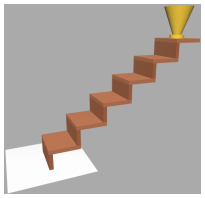
Memo WebGL

Points, vecteurs et maillage



Création de Géométrie dans three.js

```
const geometry = new THREE.BufferGeometry();
const vertices = new Float32Array( [
  -1.0, -1.0, 1.0,
   1.0, -1.0, 1.0,
   1.0, 1.0, 1.0,

   1.0, 1.0, 1.0,
  -1.0, 1.0, 1.0,
  -1.0, -1.0, 1.0
] );

geometry.setAttribute( 'position', new THREE.
     BufferAttribute( vertices, 3 ) );
const material = new THREE.MeshBasicMaterial( { color: 0
     xff0000 } );
const mesh = new THREE.Mesh( geometry, material );

scene.add( mesh ); // ajout de l'objet au graph de scene
```

Conception d'un escalier

```
var geoBlock = new THREE.BoxGeometry( width, height,
     thickness );
var blockMesh = new THREE.Mesh( geoBlock, material );
// set the X,Y,Z position
stepMesh.position.x = xPosition;
stepMesh.position.y = yPosition;
stepMesh.position.z = zPosition;
```

Le Drinking Bird

```
// Les valeurs 32, 16 sont la longitude et la latitude des
     tessellations
var sphere = new THREE.Mesh(
    new THREE.SphereGeometry( radius, 32, 16 ),
         sphereMaterial );

var cylinder = new THREE.Mesh(
    new THREE.CylinderGeometry( radiusTop, radiusBottom,
         height, 32 ), cylMaterial );
```

Les couleurs et matériaux

Réglage de la Couleur

```
var sphereMaterial = new THREE.MeshLambertMaterial( );
// Trois valeurs pour regler RGB
sphereMaterial.color.r = 1.0;
sphereMaterial.color.g = 0.0;
sphereMaterial.color.b = 0.0;
// ou utilisation de la methode setRGB
sphereMaterial.color.setRGB( 0.972, 0.749, 0.141 );
// ou utilisation des valeurs hexadecimales de 0x00 a 0xFF
     (0-255)
sphereMaterial.color.setHex( 0x1280FF );

// ou initilalisation du materiau avec la couleur
var cylMaterial = new THREE.MeshLambertMaterial( { color: 0
     xF4F100 } );
```

Attributs de sommets

```
// Comment regler les trois couleurs des face #0
const colors = new Float32Array( [
  0.2, 0.3, 0.1,
  0.0, 0.8, 0.3,
  1.0, 0.7, 0.8,
] );
geometry.setAttribute( 'color', new THREE.BufferAttribute(
     colors, 3 ) );
```

Matériaux diffus

```
material = new THREE.MeshBasicMaterial( { color: 0x80fc66,
     shading: THREE.FlatShading } );
material.color.setRGB( red, greenIUT, blue );
var newRed = material.color.r * 0.7;
```

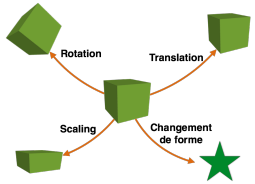
Ka, Kd, and HSL

```
var color = new Color();
// hue, saturation, et lightness, entre0-1
color.setHSL( 0.117, 0.937, 0.557 ); // orange
```

Transparence dans Three.js

```
var movingBoxMaterial = new THREE.MeshLambertMaterial(
    { color: 0xE53319, opacity: 0.7, transparent: true } );
```

Les transformations



Translation

```
sphere.position.x = xPosition;
sphere.position.y = yPosition;
sphere.position.z = zPosition;
```

Rotation

```
cube.rotation.x = xRotationInRadians;//-70*Math.PI/180
cube.rotation.y = yRotationInRadians;
cube.rotation.z = zRotationInRadians;
```

Transformation de corps rigide vs Scaling

```
cube.scale.x = xSize;
cube.scale.y = ySize;
cube.scale.z = zSize;
```

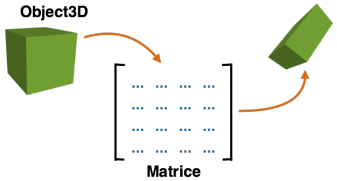
Échelle Rotation Translation

```
// Toujours le meme ordre :
cube.scale.set( xSize, ySize, zSize );
cube.rotation.set( xRotRad, yRotRad, zRotRad);
cube.position.set( xPosition, yPosition, zPosition );
```

Objet 3D (Object3D)

```
var block = new THREE.Mesh(
  new THREE.BoxGeometry(100,4,4), clockHandMaterial);
block.position.x = 40;
// Faire l'objet parent
var clockHand = new THREE.Object3D();
clockHand.add( block );

clockHand.rotation.y = -70 * Math.PI/180;
scene.add( clockHand );
```

Les matrices



La matrice identité

```
var mtx = new THREE.Matrix4();
mtx.identity();
```

Utilisation de matrice

```
var mtx = new THREE.Matrix4(
    1, 0, 0, 12,
    0, 1, 0, 16,
    0, 0, 1, -5,
    0, 0, 0, 1 );

mtx.makeTranslation( x, y, z );
mtx.makeTranslation(12,16,-5 );

forearm.matrix = mtx;
forearm.matrixAutoUpdate = false;
```

Axe de Rotation

```
mtx.makeRotationAxis( axis, theta );
```

Angle de Rotation

```
// get deux coins diametralement opposes au cube
// et calcul la direction et longueur du cylindre
var maxCorner = new THREE.Vector3( 1, 1, 1 );
var minCorner = new THREE.Vector3( -1,-1,-1 );
var cylAxis = new THREE.Vector3();
cylAxis.subVectors( maxCorner, minCorner );
var cylLength = cylAxis.length();

// prend le produit scalaire de cylAxis et du vecteur up
// pour avoir le cosinus de angle
cylAxis.normalize();
var theta = Math.acos(
    cylAxis.dot( new THREE.Vector3(0,1,0) ) );

// version alternative :
var theta = Math.acos( cylAxis.y );
```

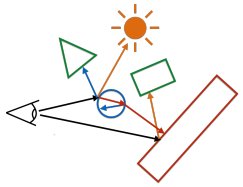
Produit vectoriel

```
var rotationAxis = new THREE.Vector3();
rotationAxis.crossVectors( cylAxis, new THREE.Vector3
     (0,1,0) );

var negRotationAxis = new THREE.Vector3();
negRotationAxis.crossVectors( new THREE.Vector3(0,1,0),
     cylAxis );

// cas special : si rotationAxis est environ zero, le
     mettre a l'axe X,
// pour que l'angle soit 0 ou PI
if ( rotationAxis.length() == 0 )
{
    rotationAxis.set( 1, 0, 0 );
}
rotationAxis.normalize();
```

Multiplication de Rotations

```
airplane.rotation.x = effectController.ex * Math.PI/180;
airplane.rotation.y = effectController.ey * Math.PI/180;
airplane.rotation.z = effectController.ez * Math.PI/180;
```

L'éclairage



Lumiere directionnelle dans three.js

```
var light = new THREE.DirectionalLight( 0xFFFAAD, 0.7 );
light.position.set( 200, 500, 600 );
scene.add( light );

light.position.set( 2, 5, 6 );
light.position.set( 0.02, 0.05, 0.06 );
```

Lumiere ambiante

```
scene.add( new THREE.AmbientLight( 0x222222 ) );

var someMaterial = new THREE.MeshLambertMaterial( );
someMaterial.color.setRGB( 0.8,0.2,0.1);
```

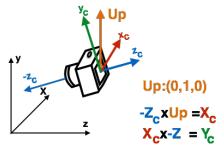
Head Light

```
var light = new THREE.PointLight( 0xFFFFFF, 1.0 );
light.position.set( 1000, 1000, 1000 );
scene.add( light );

function render() {
    var delta = clock.getDelta();
    cameraControls.update(delta);

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

Ombres dans three.js

```
renderer.shadowMap.enabled = true;

spotlight.castShadow = true;

cube.castShadow = true;
cube.receiveShadow = true;

bbird.traverse( function ( object ) {
    if ( object instanceof THREE.Mesh ) {
        object.castShadow = true;
        object.receiveShadow = true;
    }
} );

// controle du shadow bias:
spotlight.shadowBias = 0.0001;
```

Le point de vue



Camera orthographique dans Three.js

```
viewSize = 900;
aspectRatio = canvasWidth/canvasHeight;
// OrthographicCamera( left, right, top, bottom, near, far
    )
camera = new THREE.OrthographicCamera(
    -aspectRatio*viewSize / 2, aspectRatio*viewSize / 2,
    viewSize / 2, -viewSize / 2,
    -1000, 1000 );

camera.position.set( -890, 600, -480 );
cameraControls = new THREE.OrbitControls(camera, renderer.
    domElement);
cameraControls.target.set(0,310,0);
```

Camera perspective dans three.js

```
// PerspectiveCamera( angle, aspectRatio, near, far )
camera = new THREE.PerspectiveCamera( 30, aspectRatio, 1,
    10000 );
camera.position.set( -170, 170, 40 );
cameraControls = new THREE.OrbitControls(camera, renderer.
    domElement);
cameraControls.target.set(0,50,0);

camera.updateProjectionMatrix();
```

Antialiasing

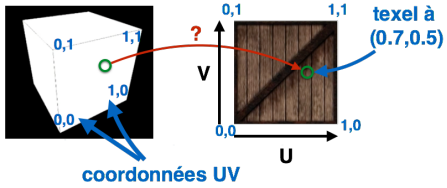
```
renderer = new THREE.WebGLRenderer( { antialias: true } );
```

Quatre Viewports

```
// Ne pas nettoyer quand plusieurs viewports sont dessines
renderer.autoClear = false;

// OrthographicCamera( left, right, top, bottom, near, far
    )
topCam = new THREE.OrthographicCamera(
    -aspectRatio*viewSize / 2, aspectRatio*viewSize / 2,
    viewSize / 2, -viewSize / 2,
    -1000, 1000 );
// X defini l'axe up
topCam.up.set( 1, 0, 0 );

render();

// top view
```

```
topCam.position.copy( cameraControls.target );
// monte d'une unite et regarde vers la cible
topCam.position.y +=1 ;
topCam.lookAt( cameraControls.target );
```

Les textures



Principe des textures

```
var crateTxr = new THREE.TextureLoader().load( 'textures/
    crate.gif' );
var material = new THREE.MeshBasicMaterial( { map: crateTxr
    } );
```

UVs in three.js

```
var triangle = new THREE.BufferGeometry();

const vertices = new Float32Array( [
    0.0, 0.0, 0.0,
    1.0, 0.0, 0.0,
    1.0, 1.0, 0.0,
] );

const uv = new Float32Array( [
    0.0, 0.0,
    1.0, 0.0,
    1.0, 1.0,
] );

triangle.setAttribute( 'position', new THREE.
    BufferAttribute( vertices, 3 ) );
triangle.setAttribute( 'uv', new THREE.BufferAttribute( uv,
    2 ) );
```

Modes Wrap

```
var texture = new THREE.Texture();
texture.wrapS = texture.wrapT = THREE.RepeatWrapping;
texture.wrapS = texture.wrapT = THREE.
    MirroredRepeatWrapping;
texture.wrapS = texture.wrapT = THREE.ClampToEdgeWrapping;
```

Transformation de Texture

```
var texture = new THREE.Texture();
texture.repeat.set( 1, 1 );
texture.offset.set( 0, 0 );
```

Magnification de Texture

```
var texture = new THREE.Texture();
texture.magFilter = THREE.NearestFilter; // un "tap"
texture.magFilter = THREE.LinearFilter; // quatre "taps"
```

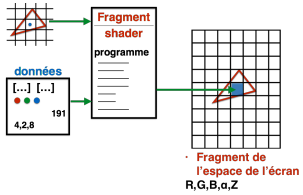
Échantillonnent et filtres

```
var texture = new THREE.Texture();
texture.magFilter = THREE.NearestFilter;
texture.magFilter = THREE.LinearFilter; // la norme

texture.minFilter = THREE.NearestFilter;
texture.minFilter = THREE.LinearFilter;
texture.minFilter = THREE.LinearMipMapLinearFilter; // la
    norme

texture.anisotropy = 1;
texture.anisotropy = renderer.getMaxAnisotropy();
```

Particules

```
var disk = new THREE.TextureLoader().load( 'disc.png' );
  var material = new THREE.SpriteMaterial( { map: disk} );
  material.color.setHSL( 0.9, 0.2, 0.6 )

  for ( var i = 0; i < 8000; i ++ ) {
    var particles = new THREE.Sprite( material );
    // accept the point only if it's in the sphere
    do {
      particles.position.x= 2000 * Math.random() - 1000;
      particles.position.y= 2000 * Math.random() - 1000;
      particles.position.z = 2000 * Math.random() - 1000;
    } while ( vertex.length() > 1000 );
    particles.scale.set(35, 35, 35);
    scene.add( particles);
  }
```

Les shaders



Exemple de Vertex Shader

```
uniform vec3 uMaterialColor;
uniform vec3 uDirLight;

varying vec3 vColor;

void main() {
    // Transform the vertex from model space to clip
        coordinates
    gl_Position = projectionMatrix * modelViewMatrix * vec4
        ( position, 1.0 );
    vec3 light = normalize( uDirLight );

    // Compute a diffuse color using Lambertian reflection,
        N * L
    float diffuse = max( dot( normal, light ), 0.0);

    vColor = uMaterialColor * diffuse;
}
```

Exemple de Fragment Shader

```
varying vec3 vColor;

void main() {
    gl_FragColor = vec4(vColor, 1.0);
}


vec3 uSpecularColor;
float specular;
...
gl_FragColor.rgb += specular * uSpecularColor;
```

Cartoon Shading

```
float diffuse = max(
    dot( normal, light ), 0.0);
```

Debugger les Shaders

```
gl_FragColor = vec4( uKd * uMaterialColor * uDirLightColor
    * diffuse, 1.0 );
```

Matériaux Anisotropic

```
for ( int i = 0; i < 2; i++ ) {
   ... do things ...
}
```

La Correction Gamma

```
renderer.gammaInput = true;
renderer.gammaOutput = true;
```

Texturage and Post-Processing

```
vec4 texelColor = texture2D( map, vUv );

vec4 cubeColor = textureCube( tCube,
    vec3( -vReflect.x, vReflect.yz ) );


varying vec2 vUv;

void main() {
    vUv = uv;
    gl_Position = projectionMatrix * modelViewMatrix * vec4
        ( position, 1.0 );
}
```

```
uniform sampler2D tDiffuse;
varying vec2 vUv;

void main() {
    vec4 cTextureScreen = texture2D( tDiffuse, vUv );

    // luma, for non-gamma-corrected computations
    vec3 lumaColor = vec3(
        cTextureScreen.r * 0.3 +
        cTextureScreen.g * 0.59 +
        cTextureScreen.b * 0.11 );

    gl_FragColor.rgb = vec4( lumaColor, 1.0 );
}
```

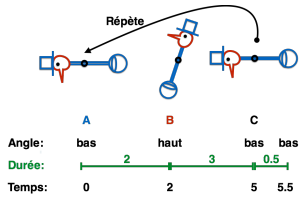
Faire une Flashlight mouvante

```
if ( length( vViewPosition.xy ) > uFlashRadius ) {
    return;
}

uniform vec2 uFlashOffset;
```

Model de deformation

```
varying vec3 vNormal;
varying vec3 vViewPosition;

uniform float uSphereRadius2;

void main() {
    gl_Position = projectionMatrix * modelViewMatrix * vec4
        ( position, 1.0 );
    vNormal = normalize( normalMatrix * normal );
    vec4 mvPosition = modelViewMatrix * vec4( position, 1.0
        );
    vViewPosition = -mvPosition.xyz;
}
```

Les interactions



Evenements

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

```
function onDocumentMouseDown( event ) {
    even.preventDefault();
```

Picking

```
function onDocumentMouseDown( event ) {
    var mouseVector = new THREE.Vector3(
        2 * ( event.clientX / canvasWidth ) - 1,
        1 - 2 * ( event.clientY / canvasHeight ));
    var projector = new THREE.Projector();
    var raycaster = projector.pickingRay( mouseVector.clone
        (), camera );

// test de d'intersection
var intersects = raycaster.intersectObjects( objects );
if ( intersects.length > 0 ) {
    intersects[ 0 ].object.material.color.setRGB(
        Math.random(), Math.random(), Math.random() );

    var sphere = new THREE.Mesh( sphereGeom, sphereMaterial
        );
    sphere.position = intersects[ 0 ].point;
    scene.add( sphere );
}

// plusieurs valeurs sont retournees
intersects[ 0 ].object
intersects[ 0 ].face
intersects[ 0 ].faceIndex
intersects[ 0 ].distance
intersects[ 0 ].point
```

La boucle de rendu

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

function animate() {
    window.requestAnimationFrame(animate);
    render();
}

function render() {
    var delta = clock.getDelta();
    cameraControls.update(delta);
    renderer.render(scene, camera);
}
```

Animation Incrementale

```
var bodyhead = new THREE.Object3D();
bodyhead.add(body);
bodyhead.add(head);

// ajout d'un champ pour la partie animation
bbird.animated = bodyhead;

bbird.add(support);
bbird.add(bodyhead);


var tiltDirection = 1;

function render() {
    bird.animated.rotation.z += tiltDirection * 0.5 * Math.
        PI/180;
    if ( bird.animated.rotation.z > 103 * Math.PI/180 ) {
        tiltDirection = -1;
        bird.animated.rotation.z = 2*(103 * Math.PI/180) -
            bird.animated.rotation.z;
    } else if ( bird.animated.rotation.z < -22 * Math.PI
        /180 ) {
        tiltDirection = 1;
        bird.animated.rotation.z = 2*(-22 * Math.PI/180) -
            bird.animated.rotation.z;
    }

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

Animation minutee

```
var clock = new THREE.Clock();

function render() {
    var delta = clock.getDelta();


// Dimension du pas fixee
bird.animated.rotation.z +=
    tiltDirection * 0.5 * Math.PI/180;

// controle de l'horloge:
bird.animated.rotation.z +=
    tiltDirection * 30 * delta * Math.PI/180;


// code final
var angle = (30 * clock.getElapsedTime()) % 250;
if ( angle < 125 ) {
    // aller de -22 a 103 degres
    bird.animated.rotation.z =
        (angle - 22) * Math.PI/180;
} else {
    // aller de 103 a -22 degres
    bird.animated.rotation.z =
        ((250-angle) - 22) * Math.PI/180;
}
```


PF Villard
email :
pierre-frederic.villard@univ-lorraine.fr