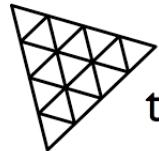




dat.GUI



SAT.js



three.js

Université Paris 8

Moteurs de Jeu¹

Nicolas JOUANDEAU
n@up8.edu

septembre 2022

1. A la découverte des fonctions proposées par les moteurs et les jeux possibles.

5 Objets 3D avec three.js

La librairie `three.js` permet de manipuler simplement des objets et des scènes en 3D ; on place le fichier `three.min.js` dans le répertoire `js` et on ajoute la ligne suivante dans le fichier `html`.

```
1 <script src=".js/three.min.js"></script>
```

5.1 Systèmes de coordonnées

On utilise un système de coordonnées cartésien avec trois axes Ox , Oy et Oz orientés comme présenté en Fig. 36.

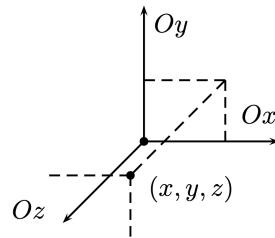


FIG. 36 – Coordonnées cartésiennes.

Dans un système de coordonnées cylindrique, x et y sont définis en fonction d'un rayon r et d'un angle θ ; on a une hauteur z comme présenté en Fig. 63.

$$\begin{cases} x = r.\cos(\theta) \\ y = r.\sin(\theta) \\ z \end{cases}$$

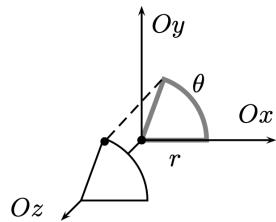


FIG. 37 – Coordonnées cylindriques.

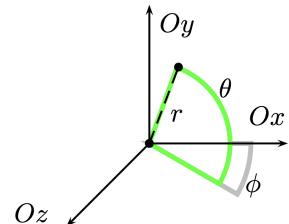


FIG. 38 – Coordonnées sphériques.

Dans un système de coordonnées sphérique, chacune des coordonnées est fonction des deux angles ϕ et θ et du rayon r de la sphère comme présenté en Fig. 64.

$$\begin{cases} x = r.\sin(\theta).\cos(\phi) \\ y = r.\sin(\theta).\sin(\phi) \\ z = r.\cos(\theta) \end{cases}$$

5.2 Dessiner une sphère avec une texture

Le programme suivant dessine une sphère qui tourne sur elle-même présentée en Fig. 39 ; le `renderer` calcule le rendu de la scène du point de vue de la caméra ; la sphère est définie par une géométrie associée à une texture.

```
1 let cnv = document.querySelector('#myCanvas');
2 let renderer = new THREE.WebGLRenderer({canvas: cnv, antialiasing: true});
3 renderer.setSize(window.innerWidth, window.innerHeight);
4 let scene = new THREE.Scene();
5 let camera = new THREE.PerspectiveCamera(75,
6   window.innerWidth/window.innerHeight, 1, 1000 );
7 camera.position.set(0, 0, 4);
8 camera.lookAt(0, 0, 0);
9 scene.add(camera);
10 let geometry = new THREE.SphereGeometry(1.0, 20, 20);
11 let texture = new THREE.TextureLoader().load("assets/namek.png");
12 let material = new THREE.MeshBasicMaterial({ map: texture});
13 let sphere = new THREE.Mesh(geometry, material);
14 scene.add(sphere);
15 function onWindowResize() {
16   camera.aspect = window.innerWidth/window.innerHeight;
17   camera.updateProjectionMatrix();
18   renderer.setSize(window.innerWidth, window.innerHeight);
19 }
20 let previousTimeStamp = undefined;
21 let updateTime = 20, elapsed = updateTime+1;
22 function update(timestamp) {
23   if(previousTimeStamp != undefined) { elapsed = timestamp-previousTimeStamp; }
24   if(elapsed > updateTime) {
25     previousTimeStamp = timestamp;
26     sphere.rotation.x += 0.01; sphere.rotation.z += 0.01;
27   }
28   renderer.render(scene, camera);
29   requestAnimationFrame(update);
30 }
31 window.addEventListener('resize', onWindowResize, false);
32 requestAnimationFrame(update);
```

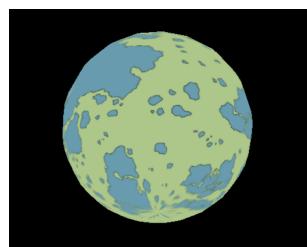


FIG. 39 – Une sphère avec une texture (planète Namek).

5.3 Dessiner plusieurs sphères en fil de fer

Le programme suivant dessine des sphères qui tournent sur elles-mêmes présentées en Fig. 40 ; la fonction `addSphere` ajoute une sphère de rayon `radius` dans le tableau `spheres` ; la géométrie d'une sphère est définie par un nombre de segments en hauteur et en largeur ; ici on crée 5 sphères de hauteur 2 à 6 qui tournent à des vitesses différentes ; on fixe la largeur à 100 segments ; on place les sphères de plus en plus vers la droite.

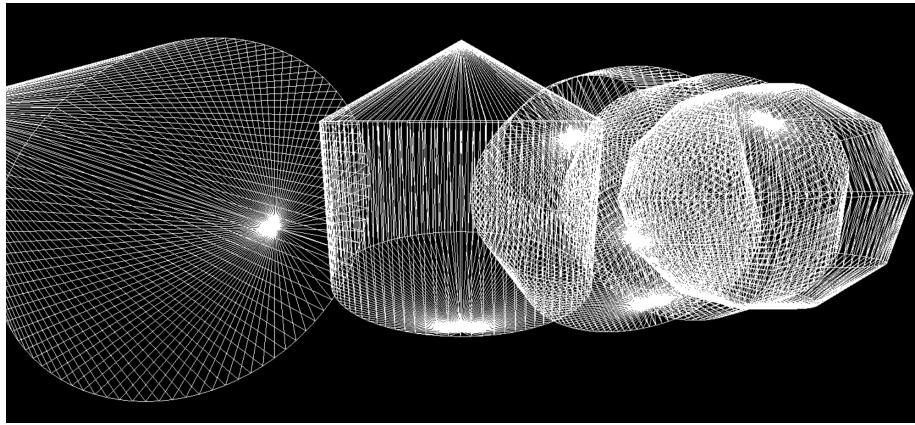


FIG. 40 – Des sphères avec 100 segments en largeur et 2 à 6 segments en hauteur.

L'option `wireframe` permet de dessiner en mode fil de fer.

```

1  let cnv = document.querySelector('#myCanvas');
2  let renderer = new THREE.WebGLRenderer({canvas: cnv,
3                                         antialiasing: true});
4  let width = window.innerWidth;
5  let height = window.innerHeight;
6  renderer.setSize(width, height);
7  let scene = new THREE.Scene();
8  let camera = new THREE.PerspectiveCamera(75, width/height, 1, 1000);
9  scene.add(camera);
10 let spheres = [];
11 function addSphere(radius, nbStack, xc, yc, zc){
12     let geometry = new THREE.SphereGeometry(radius, 100, nbStack);
13     let material = new THREE.MeshBasicMaterial({color: 0xffffffff,
14                                               wireframe: true});
15     let newSphere = new THREE.Mesh(geometry, material);
16     newSphere.position.set(xc, yc, zc);
17     spheres.push(newSphere);
18     scene.add(newSphere);
19 }
```

```

20 for (let i = 0; i < 5; i++) {
21   addSphere(1.0+0.2*i, 2+i, -1.5+1.5*i, 0.0, -1*i);
22 }
23 camera.position.set(0, 0, 2);
24 camera.lookAt(0, 0, 0);
25 function onWindowResize(){
26   width = window.innerWidth;
27   height = window.innerHeight;
28   camera.aspect = width/height;
29   camera.updateProjectionMatrix();
30   renderer.setSize(width, height);
31 }
32 let previousTimeStamp = undefined;
33 let updateTime = 20, elapsed = updateTime+1;
34 function update(timestamp) {
35   if(previousTimeStamp != undefined) {
36     elapsed = timestamp-previousTimeStamp;
37   }
38   if(elapsed > updateTime) {
39     previousTimeStamp = timestamp;
40     for (let i = 0; i < 5; i++) {
41       spheres[i].rotation.x += 0.01+0.001*i;
42       spheres[i].rotation.y += 0.01+0.001*i;
43     }
44   }
45   renderer.render(scene, camera);
46   requestAnimationFrame(update);
47 }
48 window.addEventListener('resize', onWindowResize, false);
49 requestAnimationFrame(update);

```

5.4 Dessiner une sphère à partir de segments

Le programme suivant dessine une sphère (présentée en Fig. 41) sans utiliser la structure `Sphere` de `three.js`; on utilise des coordonnées sphériques.

```
1 let cnv = document.querySelector('#myCanvas');
2 let renderer = new THREE.WebGLRenderer({canvas: cnv, antialiasing: true});
3 let width = window.innerWidth;
4 let height = window.innerHeight;
5 renderer.setSize(width, height);
6 let scene = new THREE.Scene();
7 let camera = new THREE.PerspectiveCamera(75, width/height, 1, 1000);
8 scene.add(camera);
9 let nbSector = 100;
10 let nbStack = 5;
11 let points = [];
12 let sectorStep = 2*Math.PI/nbSector;
13 let stackStep = Math.PI/nbStack;
14 let theta = -1*Math.PI;
15 for (let i=0; i<nbSector; i++) {
16     let phi = -1*Math.PI/2;
17     for (let j=0; j<nbStack; j++) {
18         let x = Math.cos(theta)*Math.cos(phi);
19         let y = Math.sin(theta)*Math.cos(phi);
20         let z = Math.sin(phi);
21         let x2 = Math.cos(theta+sectorStep)*Math.cos(phi);
22         let y2 = Math.sin(theta+sectorStep)*Math.cos(phi);
23         let z2 = Math.sin(phi);
24         let x3 = Math.cos(theta)*Math.cos(phi+stackStep);
25         let y3 = Math.sin(theta)*Math.cos(phi+stackStep);
26         let z3 = Math.sin(phi+stackStep);
27         let x4 = Math.cos(theta+sectorStep)*Math.cos(phi+stackStep);
28         let y4 = Math.sin(theta+sectorStep)*Math.cos(phi+stackStep);
29         let z4 = Math.sin(phi+stackStep);
30         points.push(new THREE.Vector3(x,y,z));
31         points.push(new THREE.Vector3(x2,y2,z2));
32         points.push(new THREE.Vector3(x,y,z));
33         points.push(new THREE.Vector3(x3,y3,z3));
34         phi += stackStep;
35     }
36     theta += sectorStep;
37 }
38 let geometry = new THREE.BufferGeometry().setFromPoints( points );
39 let material = new THREE.MeshBasicMaterial( {color: 0xffffffff} );
40 let mySphere = new THREE.Line(geometry, material );
41 scene.add(mySphere);
```

```

42 camera.position.set(0, 0, 2);
43 camera.lookAt(0, 0, 0);
44 function onWindowResize(){
45   width = window.innerWidth;
46   height = window.innerHeight;
47   camera.aspect = width/height;
48   camera.updateProjectionMatrix();
49   renderer.setSize(width, height);
50 }
51 let previousTimeStamp = undefined;
52 let updateTime = 20, elapsed = updateTime+1;
53 function update(timestamp) {
54   if(previousTimeStamp != undefined) {
55     elapsed = timestamp-previousTimeStamp;
56   }
57   if(elapsed > updateTime) {
58     previousTimeStamp = timestamp;
59     mySphere.rotation.x += 0.001;
60     mySphere.rotation.y += 0.001;
61   }
62   renderer.render(scene, camera);
63   requestAnimationFrame(update);
64 }
65 window.addEventListener('resize', onWindowResize, false);
66 requestAnimationFrame(update);

```

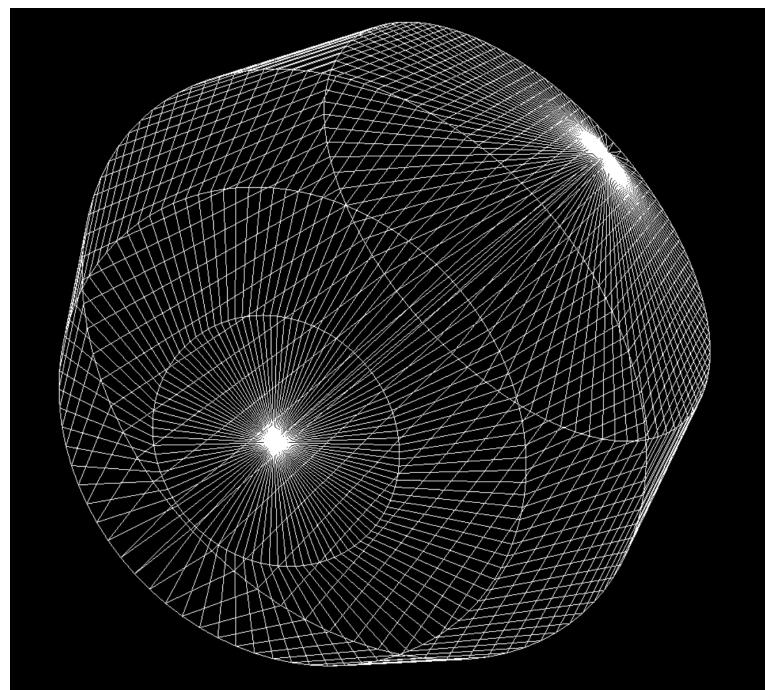


FIG. 41 – Une sphère avec 100 segments en largeur et 5 segments en hauteur.

5.5 Primitives géométriques usuelles

Le programme suivant dessine des objets (présentés en Fig. 42) correspondant aux primitives géométriques usuelles de `three.js`; on fixe ici les niveaux de détails à 0.

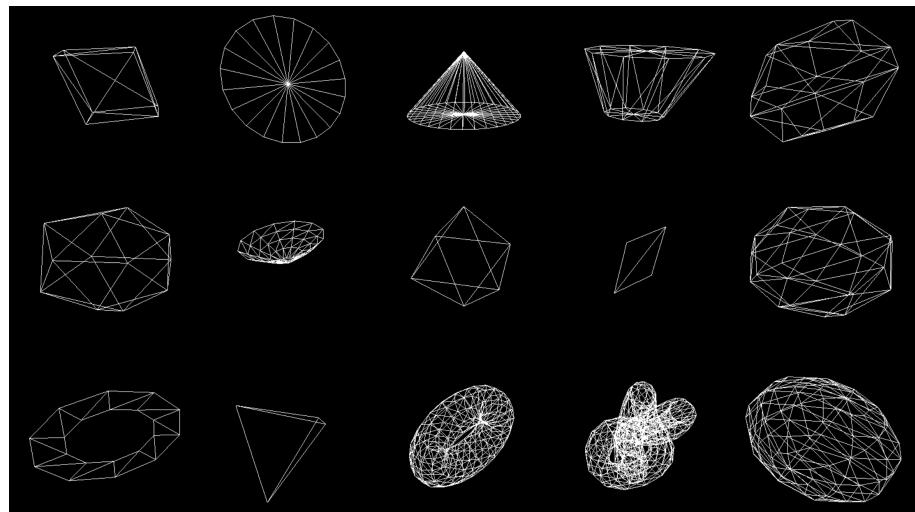


FIG. 42 – Les formes géométriques usuelles de `three.js`.

```

1 let cnv = document.querySelector('#myCanvas');
2 let renderer = new THREE.WebGLRenderer({canvas: cnv,
3                                         antialiasing: true});
4 let width = window.innerWidth;
5 let height = window.innerHeight;
6 renderer.setSize(width, height);
7 let scene = new THREE.Scene();
8 let camera = new THREE.PerspectiveCamera(75, width/height, 1,
9                                         1000);
10 scene.add(camera);
11 let objects = [];
12 function addObject(geom, xc, yc, zc){
13     let material = new THREE.MeshBasicMaterial({color: 0xffffffff,
14                                               wireframe: true} );
15     let newObj = new THREE.Mesh(geom, material);
16     newObj.position.set(xc, yc, zc);
17     objects.push(newObj);
18     scene.add(newObj);
19 }
```

```

20 let widthBox = 1.0, heightBox = 1.0, depthBox = 1.0;
21 addObject(new THREE.BoxGeometry(widthBox, heightBox, depthBox),
22   -6.0, 3.0, 0.0);
23 let radiusCircle = 1.0, sgtCircle = 20;
24 addObject(new THREE.CircleGeometry(radiusCircle, sgtCircle),
25   -3.0, 3.0, 0.0);
26 let radiusCone = 1.0, heightCone = 1, radSgtCone = 32;
27 addObject(new THREE.ConeGeometry(radiusCone, heightCone, radSgtCone),
28   0.0, 3.0, 0.0);
29 let radiusTopCylinder = 1.0, radiusBotCylinder = 0.5;
30 let heightCylinder = 1.0, radialSgtCylinder = 10;
31 addObject(new THREE.CylinderGeometry(radiusTopCylinder,
32   radiusBotCylinder, heightCylinder, radialSgtCylinder),
33   3.0, 3.0, 0.0);
34 let radiusDodecahedron = 1.0, detailDodecahedron = 0;
35 addObject(new THREE.DodecahedronGeometry(radiusDodecahedron,
36   detailDodecahedron), 6.0, 3.0, 0.0);
37 let radiusIcosahedron = 1.0, detailIcosahedron = 0;
38 addObject(new THREE.IcosahedronGeometry(radiusIcosahedron,
39   detailIcosahedron), -6.0, 0.0, 0.0);
40 let ptsLathe = [];
41 for ( let i = 0; i < 5; i ++ ) {
42   ptsLathe.push(new THREE.Vector2(Math.sin(i*0.2)*0.8+0.1, i*0.1));
43 }
44 addObject(new THREE.LatheGeometry(ptsLathe), -3.0, 0.0, 0.0);
45 let radiusOctahedron = 1.0, detailOctahedron = 0;
46 addObject(new THREE.OctahedronGeometry(radiusOctahedron,
47   detailOctahedron), 0.0, 0.0, 0.0);
48 let widthPlane = 1.0, heightPlane = 1.0;
49 addObject(new THREE.PlaneGeometry(widthPlane, heightPlane),
50   3.0, 0.0, 0.0);
51 let verticesPolyhedron = [ -1,-1,-1,  1,-1,-1,  1, 1,-1,  -1, 1,-1,
52                           -1,-1, 1,  1,-1, 1,  1, 1, 1,  -1, 1, 1 ];
53 let indicesFacesPolyhedron = [
54   2,1,0,    0,3,2,
55   0,4,7,    7,3,0,
56   0,1,5,    5,4,0,
57   1,2,6,    6,5,1,
58   2,3,7,    7,6,2,
59   4,5,6,    6,7,4 ];
60 addObject(new THREE.PolyhedronGeometry(verticesPolyhedron,
61   indicesFacesPolyhedron, 1, 1), 6.0, 0.0, 0.0);
62 let innerRadiusRing = 0.6, outerRadiusRing = 1.0, thetaSgtRing = 10;
63 addObject(new THREE.RingGeometry(innerRadiusRing, outerRadiusRing,
64   thetaSgtRing), -6.0, -3.0, 0.0);
65 let radiusTetrahedron = 1.0, detailTetrahedron = 0;
66 addObject(new THREE.TetrahedronGeometry(radiusTetrahedron,
67   detailTetrahedron), -3.0, -3.0, 0.0);

```

```

68 let radiusTorus = 0.7, tubeTorus = 0.3, radialSgtTorus = 10;
69 let tubularSgtTorus = 20;
70 addObject(new THREE.TorusGeometry(radiusTorus, tubeTorus,
71     radialSgtTorus, tubularSgtTorus), 0.0, -3.0, 0.0);
72 let radiusTorusKnot = 0.6, tubeTorusKnot = 0.2;
73 let tubularSgtTorusKnot = 40, radialSgtTorusKnot = 10;
74 addObject(new THREE.TorusKnotGeometry(radiusTorusKnot, tubeTorusKnot,
75     tubularSgtTorusKnot, radialSgtTorusKnot), 3.0, -3.0, 0.0);
76 let radiusSphere = 1.0, widthSphere = 10, heightSphere = 10;
77 addObject(new THREE.SphereGeometry(radiusSphere, widthSphere,
78     heightSphere), 6.0, -3.0, 0.0);
79 camera.position.set(0, 0, 7);
80 camera.lookAt(0, 0, 0);
81 function onWindowResize(){
82     width = window.innerWidth;
83     height = window.innerHeight;
84     camera.aspect = width/height;
85     camera.updateProjectionMatrix();
86     renderer.setSize(width, height);
87 }
88 let previousTimeStamp = undefined;
89 let updateTime = 20, elapsed = updateTime+1;
90 function update(timestamp) {
91     if(previousTimeStamp != undefined) {
92         elapsed = timestamp-previousTimeStamp;
93     }
94     if(elapsed > updateTime) {
95         previousTimeStamp = timestamp;
96         for (let i = 0; i < objects.length; i++) {
97             objects[i].rotation.x += 0.01;
98             objects[i].rotation.y += 0.01;
99         }
100    }
101    renderer.render(scene, camera);
102    requestAnimationFrame(update);
103 }
104 window.addEventListener('resize', onWindowResize, false);
105 requestAnimationFrame(update);

```

5.6 Light et Material

La réaction des objets à la lumière est une composition de trois composantes vectorielles nommées ambiante, diffuse et spéculaire, avec en plus selon les modèles de matériaux, la brillance, la rugosité, l'effet métallique ; la rugosité est l'inverse de la brillance ; selon le modèle d'éclairage choisi, le résultat diffère :

- La composante ambiante définit la couleur reflétée par un objet sous un éclairage ambiant.
- La composante diffuse définit la couleur reflétée par un objet sous un éclairage diffus.
- La composante spéculaire définit la couleur reflétée par un objet.
- La brillance (*i.e. shininess*) définit la netteté et la taille de la tâche lumineuse de la composante spéculaire.
- Le `MeshBasicMaterial` n'est pas affecté par la lumière.
- Le `MeshLambertMaterial` calcule la lumière pour les sommets.
- Le `MeshPhongMaterial` calcule la lumière pour les pixels et ajoute la brillance.
- Le `MeshStandardMaterial` calcule la lumière pour les pixels et ajoute la rugosité (*i.e. roughness*) et l'effet métallique (*i.e. metalness*).
- Le `MeshPhysicalMaterial` ajoute à `MeshStandardMaterial` un vernis (*i.e. clearcoat*)
- Le `MeshToonMaterial` ajoute un effet cartoon au `MeshPhongMaterial`.
- Le `MeshDepthMaterial` permet de visualiser la profondeur de chaque pixel (les plus proches en clair et les plus éloignés en foncé).
- Le `MeshNormalMaterial` permet de visualiser les normales de chaque pixel (selon la direction : rouge en *x*, vert en *y* et bleu en *z*).
- La propriété de `flatShading` uniformise l'éclairage sur chaque face.
- La propriété de `side` indique le côté pris en compte pour l'éclairage : `THREE.FrontSide`, `THREE.BackSide`, ou `THREE.DoubleSide`.

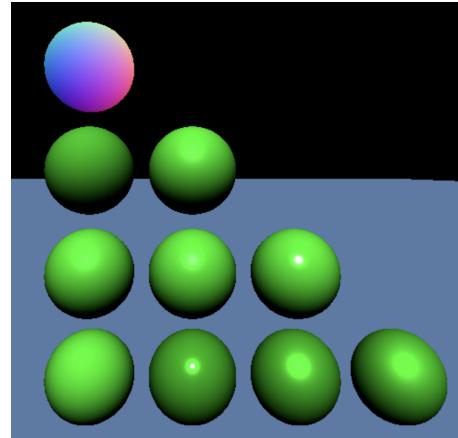


FIG. 43 – Quelques `material` : `MeshNormalMaterial` à la ligne 1, puis `MeshLambertMaterial` et `MeshPhongMaterial` à la ligne 2, puis 3 `MeshPhongMaterial` avec `shininess` croissante à la ligne 3 et 4 `MeshStandardMaterial` avec `roughness` croissante et `metalness` constante à 0.3 à la ligne 4.

La propriété de `side` est très importante : pour tous les objets solides, être en `THREE.FrontSide` suffit souvent ; pour les objets plans et les objets non solides, être en `THREE.BackSide` suffit souvent également.

On détaille les effets obtenus dans Fig. 43 ; pour toutes les sphères, on utilise la fonction `addObjectTo` et les variables `radiusSphere`, `widthSphere` et `heightSphere` ; on utilise un éclairage directionnel ; on ajoute un plan bleu proche d'un bleu horizon qui est assimilé au sol.

```

1 let objects = [];
2 function addObjectTo(obj, root, xc, yc, zc){
3     obj.position.set(xc, yc, zc);
4     objects.push(obj);
5     root.add(obj);
6 }
7 let dirLight = new THREE.DirectionalLight(0xffffffff);
8 dirLight.position.set(0, 10, 10);
9 scene.add(dirLight);
10 let mesh = new THREE.Mesh(new THREE.PlaneGeometry(100, 200),
11     new THREE.MeshPhongMaterial( { color: 0x79aceb, depthWrite: false } ) );
12 mesh.rotation.x = - Math.PI/2;
13 scene.add(mesh);
14 let radiusSphere = 0.3, widthSphere = 30, heightSphere = 30;
```

Pour obtenir la première sphère, qui présente les normales de chaque pixel :

```

1 let obj0 = new THREE.Mesh(
2     new THREE.SphereGeometry(radiusSphere, widthSphere, heightSphere),
3     new THREE.MeshNormalMaterial ({flatShading: false}));
4 addObjectTo(obj0, scene, -.7, 2.7, 0.0);
```

Pour obtenir les deux sphères suivantes :

```

1 let obj1 = new THREE.Mesh(
2     new THREE.SphereGeometry(radiusSphere, widthSphere, heightSphere),
3     new THREE.MeshLambertMaterial({color: 0x00cc00, flatShading: false,});
4 addObjectTo(obj1, scene, -0.7, 2., 0.0);
5 let obj2 = new THREE.Mesh(
6     new THREE.SphereGeometry(radiusSphere, widthSphere, heightSphere),
7     new THREE.MeshPhongMaterial({color: 0x00ff00, flatShading: false});
8 addObjectTo(obj2, scene, 0.0, 2., 0.0);
```

Pour obtenir les trois sphères, qui présente une variation de `MeshPhongMaterial` :

```
1 let obj2a = new THREE.Mesh(
2   new THREE.SphereGeometry(radiusSphere, widthSphere, heightSphere),
3   new THREE.MeshPhongMaterial({color: 0x00ff00, flatShading: false,
4     shininess: 30}));
5 addObjectTo(obj2a, scene, -.7, 1.3, 0.0);
6 let obj2b = new THREE.Mesh(
7   new THREE.SphereGeometry(radiusSphere, widthSphere, heightSphere),
8   new THREE.MeshPhongMaterial({color: 0x00ff00, flatShading: false,
9     shininess: 60}));
10 addObjectTo(obj2b, scene, 0.0, 1.3, 0.0);
11 let obj2c = new THREE.Mesh(
12   new THREE.SphereGeometry(radiusSphere, widthSphere, heightSphere),
13   new THREE.MeshPhongMaterial({color: 0x00ff00, flatShading: false,
14     shininess: 150}));
15 addObjectTo(obj2c, scene, .7, 1.3, 0.0);
```

Pour obtenir les quatre sphères, qui présente une variation de `MeshStandardMaterial` :

```
1 let obj3 = new THREE.Mesh(
2   new THREE.SphereGeometry(radiusSphere, widthSphere, heightSphere),
3   new THREE.MeshStandardMaterial({color: 0x00ff00, flatShading: false}));
4 addObjectTo(obj3, scene, -.7, .6, 0.0);
5 let obj3a = new THREE.Mesh(
6   new THREE.SphereGeometry(radiusSphere, widthSphere, heightSphere),
7   new THREE.MeshStandardMaterial({color: 0x00ff00, flatShading: false,
8     roughness: 0.2, metalness: 0.3}));
9 addObjectTo(obj3a, scene, 0.0, .6, 0.0);
10 let obj3b = new THREE.Mesh(
11   new THREE.SphereGeometry(radiusSphere, widthSphere, heightSphere),
12   new THREE.MeshStandardMaterial({color: 0x00ff00, flatShading: false,
13     roughness: 0.4, metalness: 0.3}));
14 addObjectTo(obj3b, scene, 0.7, .6, 0.0);
15 let obj3c = new THREE.Mesh(
16   new THREE.SphereGeometry(radiusSphere, widthSphere, heightSphere),
17   new THREE.MeshStandardMaterial({color: 0x00ff00, flatShading: false,
18     roughness: 0.6, metalness: 0.3}));
19 addObjectTo(obj3c, scene, 1.4, .6, 0.0);
```

Pour la lumière, il existe différents types de sources lumineuses ; il importe de les combiner pour produire une belle lumière dans la scène. Les propriétés sont génériquement les suivantes :

- `color` définit la couleur associée.
- `intensity` définit l'intensité de la lumière ; si elle décroît, c'est l'intensité initiale.
- les éclairages ambients : les objets sont éclairés indépendamment de leur position :
 - `AmbientLight` : la lumière est omnidirectionnellement identique en couleur et en intensité ; sans dégradé, l'éclairage ne fournit aucune information de profondeur sur l'objet éclairé.
 - `HemisphereLight` : la lumière s'estompe d'une couleur `color` coté ciel vers une couleur `groundColor` coté terre.
- les éclairages directes : la lumière va vers les objets éclairés, changer leur position modifie le résultat :
 - `DirectionalLight` : la lumière part d'une position `position` et va vers une autre position `target` ; le comportement résultant est similaire à celui de la lumière du jour ; la source réelle de lumière est considérée à une distance infiniment grande et produit des rayons lumineux parallèles.
 - `SpotLight` : la lumière part d'une position et va vers une autre position en s'élargissant en forme de cône ; `distance` définit la portée maximale de la lumière ; `angle` définit l'angle de dispersion (qui est majoré par $\pi/2$) ; `penumbra` définit le pourcentage d'atténuation dans l'ombre ; `decay` définit la perte de lumière le long de la propagation de la lumière.
 - `PointLight` : la lumière est émise dans toutes les directions à partir d'une position ; le comportement résultant est similaire à celui d'une ampoule ; `distance` définit la portée maximale de la lumière ; `decay` définit la perte de lumière le long de la propagation de la lumière.
 - `RectAreaLight`⁶ : la lumière est émise dans toutes les directions à partir d'un rectangle ; le comportement résultant est similaire à celui d'une fenêtre ou d'un néon ; `width` et `height` définissent la taille du rectangle.

Les Fig 44 à 52 présentent 3 sphères de couleur respective rouge, vert, bleu, et de géométrie $(1.0, 20, 20)$ avec différents `material` :

- une avec `MeshBasicMaterial` en $(-2.0, 0.0, 0.0)$
- une avec `MeshPhongMaterial` en $(0.0, 0.0, 0.0)$
- une avec `MeshPhysicalMaterial` en $(2.0, 0.0, 0.0)$

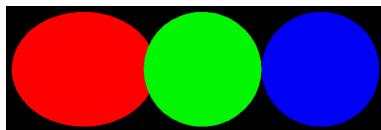


FIG. 44 – Avec `AmbientLight`.

6. Ce type d'éclairage est compatible uniquement avec `MeshStandardMaterial` et `MeshPhysicalMaterial`.

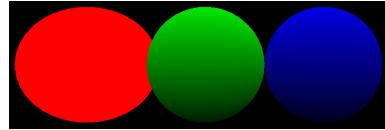


FIG. 45 – Avec HemisphereLight.

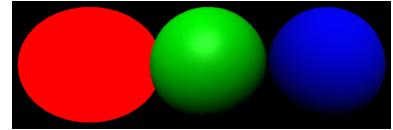


FIG. 46 – Avec DirectionalLight.

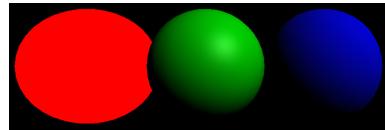


FIG. 47 – Avec PointLight.

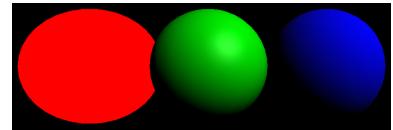


FIG. 48 – Avec SpotLight.



FIG. 49 – Avec AreaRectLight.

En les combinant, on obtient des effets plus réalistes présentés en Fig 50 à 52.

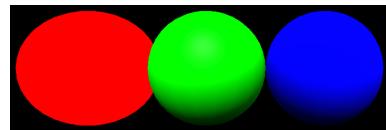


FIG. 50 – Avec HemisphereLight et DirectionalLight.

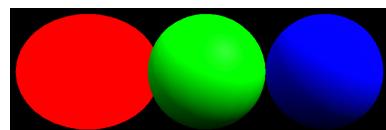


FIG. 51 – Avec HemisphereLight et PointLight.

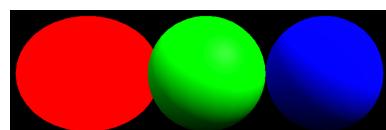


FIG. 52 – Avec HemisphereLight et SpotLight.

5.7 Texture

Avec la propriété `map` d'un `MeshBasicMaterial`, il est possible d'associer une texture à chacune des formes primitives ; selon la forme, le résultat diffère ; pour certaines formes, il est possible de placer plusieurs textures :

- pour `ConeGeometry`, associer deux textures permet d'avoir une texture sur le cône et une sur le disque
- pour `CylinderGeometry`, associer trois textures permet d'avoir une texture sur le tour, le dessus et le dessous
- pour `BoxGeometry`, associer six textures permet d'avoir une texture par face

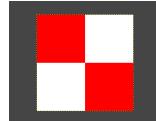


FIG. 53 – Simple texture rouge et blanc présentée sur fond gris.

En associant la texture présentée en Fig. 53 sur chacune des formes primitives usuelles présentées en Fig. 42, on obtient Fig. 54 ; la fonction `addObject` est modifiée comme ci-dessous ; pour éviter de faire disparaître certaines formes quand elles sont vues à partir de leur face arrière, la propriété `side` est fixée à `THREE.DoubleSide` (dans une scène usuelle, pour un objet correspondant au sol, `THREE.FrontSide` est suffisant).

```

1 let objects = [];
2 let texture = new THREE.TextureLoader().load("assets/red-white-32.png");
3 function addObject(geom, xc, yc, zc){
4     let newObj = new THREE.Mesh(geom,
5         new THREE.MeshBasicMaterial({ map: texture, side: THREE.DoubleSide }));
6     newObj.position.set(xc, yc, zc);
7     objects.push(newObj);
8     scene.add(newObj);
9 }
```

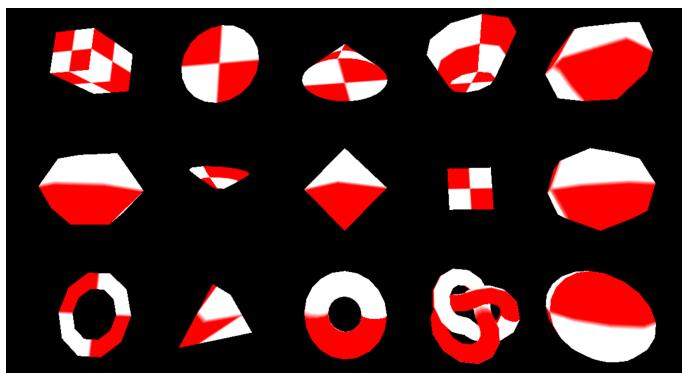


FIG. 54 – Texture sur formes primitives.

Selon les valeurs fixées pour les propriétés de la texture, il est possible de définir le plaquage de la texture ; quand la texture est plus petite ou plus grande qu'elle n'est en réalité, le `renderer` utilise des valeurs mip, correspondants aux pixels de la texture à une résolution proche de la taille actuelle d'affichage de la texture :

- `minFilter` indique comment dessiner une texture vue plus petite que ce qu'elle est
 - `THREE.NearestFilter` choisit le pixel le plus proche
 - `THREE.LinearFilter` choisit la moyenne des 4 pixels les plus proches
 - `THREE.NearestMipMapNearestFilter` choisit la valeur mip la plus proche
 - `THREE.NearestMipMapLinearFilter` choisit la moyenne des 2 valeurs mip les plus proches
 - `THREE.LinearMipMapNearestFilter` choisit la moyenne des 4 valeurs mip les plus proches
 - `THREE.LinearMipMapLinearFilter` choisit la moyenne des 8 pixels des 2 valeurs mip les plus proches
- `magFilter` indique comment dessiner une texture vue plus grande que ce qu'elle est
 - `THREE.NearestFilter` choisit le pixel le plus proche
 - `THREE.LinearFilter` choisit la moyenne des 4 pixels les plus proches
- `wrapS` indique comment elle est plaquée horizontalement
 - `THREE.RepeatWrapping` répète simplement la texture
 - `THREE.MirroredRepeatWrapping` répète en miroir la texture
 - `THREE.ClampToEdgeWrapping` répète le dernier pixel de la texture jusqu'à la fin de la forme
- `wrapT` indique comment elle est plaquée verticalement avec les mêmes valeurs possibles que `wrapS`
 - `repeat.x` indique le nombre de répétitions de la texture horizontalement
 - `repeat.y` indique le nombre de répétitions de la texture verticalement
 - `offset.x` indique le décalage horizontal de la texture
 - `offset.y` indique le décalage vertical de la texture
 - `rotation` indique la rotation appliquée à la texture

Les textures font parties de la mémoire utilisée par le programme lors de son exécution ; il sera toujours important de considérer l'espace occupé en fonction de la taille (w, h) de la texture (sans considération de compression) en utilisant l'approximation suivante : $w \times h \times 4 \times 1.33$ octets.

Pour la suite, on utilise la texture de caisse présentée en Fig. 55.



FIG. 55 – Texture de caisse.

On utilise un objet de forme `PlaneGeometry` de 10 par 5, sur lequel on applique une texture ; les résultats sont présentés dans les Fig. 56 à 59 ; on fixe `minFilter` à `THREE.NearestMipmapNearestFilter` et `magFilter` à `THREE.NearestFilter` pour conserver un effet de pixelisation sans flou, proche de la texture initiale.

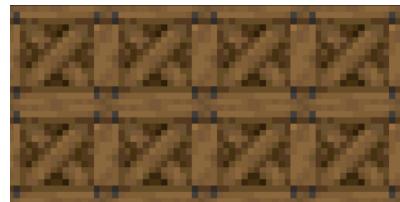


FIG. 56 – `wrapS` à `THREE.RepeatWrapping`, `wrapT` à `THREE.RepeatWrapping`, `repeat.x` à 4 et `repeat.y` à 2.

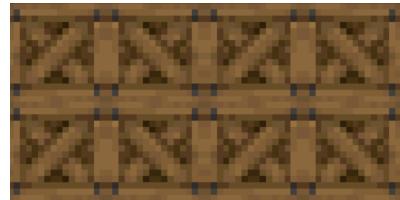


FIG. 57 – `wrapS` à `THREE.MirroredRepeatWrapping`, `wrapT` à `THREE.RepeatWrapping`, `repeat.x` à 4 et `repeat.y` à 2.

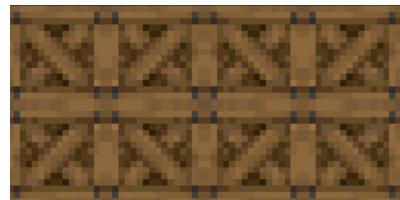


FIG. 58 – `wrapS` à `THREE.MirroredRepeatWrapping`, `wrapT` à `THREE.MirroredRepeatWrapping`, `repeat.x` à 4 et `repeat.y` à 2.



FIG. 59 – `wrapS` à `THREE.RepeatWrapping`, `wrapT` à `THREE.RepeatWrapping`, `repeat.x` à 4, `repeat.y` à 2 et `rotation` à 16.

Il est possible de définir une couleur pour le fond de la scène.

```
1 scene.background = new THREE.Color('white');
```

Il est possible d'avoir de la transparence sur un material, avec les propriétés `opacity` et `transparent` comme présenté en Fig. 60; Cela fonctionne sur les couleurs et sur les textures, avec les material `MeshBasicMaterial`, `MeshStandardMaterial`, `MeshPhongMaterial`, `MeshPhysicalMaterial`

```
1 let widthBox = .8, heightBox = .8, depthBox = .8;
2 let texture = new THREE.TextureLoader().load("block/barrel_side.png");
3 let cube1 = new THREE.Mesh(new THREE.BoxGeometry(widthBox, heightBox, depthBox),
4   new THREE.MeshBasicMaterial({map: texture}));
5 addObjectTo(cube1, scene, -1, 2, 0.0);
6 let textureTop = new THREE.TextureLoader().load("block/barrel_top.png");
7 let textureSide = new THREE.TextureLoader().load("block/barrel_side.png");
8 const cMaterials = [
9   new THREE.MeshBasicMaterial({map:textureTop, opacity:0.5, transparent:true}),
10  new THREE.MeshBasicMaterial({map:textureTop, opacity:0.5, transparent:true}),
11  new THREE.MeshBasicMaterial({map:textureSide, opacity:0.5, transparent:true}),
12  new THREE.MeshBasicMaterial({map:textureSide, opacity:0.5, transparent:true}),
13  new THREE.MeshBasicMaterial({map:textureSide, opacity:0.5, transparent:true}),
14  new THREE.MeshBasicMaterial({map:textureSide, opacity:0.5, transparent:true}),
15];
16 let cube2 = new THREE.Mesh(
17   new THREE.BoxGeometry(widthBox, heightBox, depthBox), cubeMaterials);
18 addObjectTo(cube2, scene, 1, 2, 0.0);
19 let cube3 = new THREE.Mesh(
20   new THREE.BoxGeometry(widthBox, heightBox, depthBox),
21   new THREE.MeshStandardMaterial({color:0xff0000}));
22 addObjectTo(cube3, scene, -1, 2, 2);
23 let cube4 = new THREE.Mesh(
24   new THREE.BoxGeometry(widthBox, heightBox, depthBox),
25   new THREE.MeshPhysicalMaterial({color:0x0000ff, opacity:0.5, transparent:true}));
26 addObjectTo(cube4, scene, 1, 2, 2);
```



FIG. 60 – Transparence sur des cubes colorés et texturés.

Pour dessiner franchement les ombres des formes, il est nécessaire d'utiliser un éclairage directionnel avec `DirectionalLight`, `PointLight` ou `SpotLight`; ensuite il s'agit d'activer le calcul des ombres à différents niveaux :

- sur le `renderer`, il faut mettre `shadowMap.enabled` à `true`
- sur l'éclairage, il faut mettre `castShadow` à `true`
- sur l'objet supposé recevoir une ombre, il faut mettre `receiveShadow` à `true`

Considérant que le calcul des ombres est très consommateur en temps de calcul, il est avisé d'activer ce calcul pour le moins d'objets possible ; Fig. 61 présente le résultat.

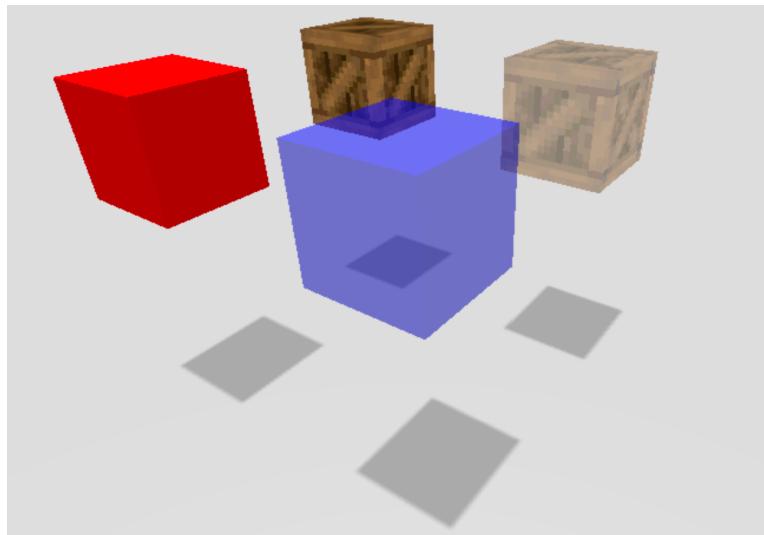


FIG. 61 – Ombres des cubes sur le sol.

5.8 Graphe de scène

Le graphe de scène de `three.js` permet de hiérarchiser les objets selon un arbre ; le noeud `scene` est considéré comme la racine de l'arbre ; chaque noeud est placé dans le repère local de son noeud parent ; une opération sur un noeud est faite dans le repère local associé ; appliquer une transformation à un noeud se propage automatique dans les noeuds enfants.

A la scène, on ajoute deux noeuds, correspondants à un cube rouge en (0,0,0) et à une sphère rouge en (1.5,0,0).

A la sphère rouge, on ajoute deux noeuds, correspondants à une petite sphère verte en (2,0,0) et à une petite sphère bleue en (-2,0,0).

La scène correspond au noeud `scene`, le cube rouge au noeud `box0`, la sphère rouge au noeud `obj0`, la sphère verte au noeud `obj1` et la sphère bleue au noeud `obj2`.

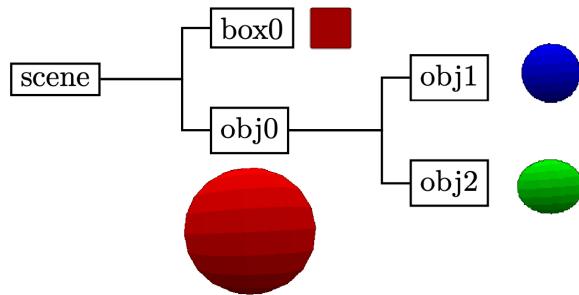


FIG. 62 – Graphe de scène.

A chaque instant, on applique une rotation autour de l'axe Oz sur chaque sphère ; la sphère rouge (`obj0`) tourne sur elle-même ; la sphère verte (`obj1`) et la sphère bleue (`obj2`) tournent autour de la sphère rouge.

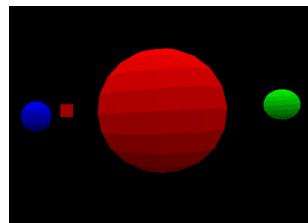


FIG. 63 – Position initiale.

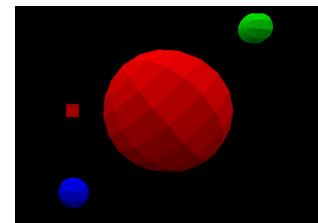


FIG. 64 – Rotation autour de `obj0`.

```

1 let cnv = document.querySelector('#myCanvas');
2 let renderer = new THREE.WebGLRenderer({canvas: cnv, antialiasing: true});
3 renderer.setSize(window.innerWidth, window.innerHeight);
4 let scene = new THREE.Scene();
5 let camera = new THREE.PerspectiveCamera(75,
6   window.innerWidth/window.innerHeight, 1, 1000);
7 camera.position.set(0, 0, 5);
  
```

```

8   camera.lookAt(0, 0, 0);
9   scene.add(camera);
10  let hemiLight = new THREE.HemisphereLight(0xfffffff, 0x444444);
11  hemiLight.position.set(0, 20, 0 );
12  scene.add(hemiLight);
13  let objects = [];
14  function addObjectTo(obj, root, xc, yc, zc){
15    obj.position.set(xc, yc, zc);
16    objects.push(obj);
17    root.add(obj);
18  }
19  let widthBox = 0.2, heightBox = 0.2, depthBox = 0.2;
20  let box0 = new THREE.Mesh(
21    new THREE.BoxGeometry(widthBox, heightBox, depthBox),
22    new THREE.MeshPhongMaterial({color: 0xff0000, flatShading: true,}));
23  addObjectTo(box0, scene, 0.0, 0.0, 0.0);
24  let radiusSphere = 1.0, widthSphere = 10, heightSphere = 10;
25  let obj0 = new THREE.Mesh(
26    new THREE.SphereGeometry(radiusSphere, widthSphere, heightSphere),
27    new THREE.MeshPhongMaterial({color: 0xff0000, flatShading: true,}));
28  addObjectTo(obj0, scene, 1.5, 0.0, 0.0);
29  let obj1 = new THREE.Mesh(
30    new THREE.SphereGeometry(radiusSphere/4, widthSphere, heightSphere),
31    new THREE.MeshPhongMaterial({color: 0x00ff00, flatShading: true,}));
32  addObjectTo(obj1, obj0, 2.0, 0.0, 0.0);
33  let obj2 = new THREE.Mesh(
34    new THREE.SphereGeometry(radiusSphere/4, widthSphere, heightSphere),
35    new THREE.MeshPhongMaterial({color: 0x0000ff, flatShading: true,}));
36  addObjectTo(obj2, obj0, -2.0, 0.0, 0.0);
37  function onWindowResize(){
38    camera.aspect = window.innerWidth/window.innerHeight;
39    camera.updateProjectionMatrix();
40    renderer.setSize(window.innerWidth, window.innerHeight);
41  }
42  function update(timestamp) {
43    if(previousTimeStamp != undefined) {
44      elapsed = timestamp-previousTimeStamp;
45    }
46    if(elapsed > updateTime) {
47      previousTimeStamp = timestamp;
48      for (let i = 1; i < objects.length; i++) {
49        objects[i].rotation.z += 0.001;
50      }
51    }
52    renderer.render(scene, camera);
53    requestAnimationFrame(update);
54  }
55  window.addEventListener('resize', onWindowResize, false);
56  requestAnimationFrame(update);

```

5.9 Modèle 3D glTF

Le programme suivant charge un modèle glTF présenté en Fig. 65 ; on place le fichier `Fox.glb` dans le répertoire `assets`, on ajoute le fichier `GLTFLoader.js` dans le répertoire `js` et on ajoute la ligne suivante dans le fichier `html`.

```
1 <script src=".//js/GLTFLoader.js"></script>
```

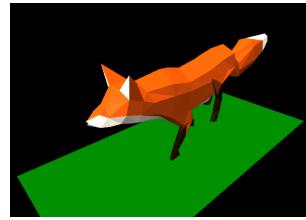


FIG. 65 – Modèle de renard.

```
1 //... canvas, renderer, scene, camera
2 //... HemisphereLight et DirectionalLight
3 let loader = new THREE.GLTFLoader();
4 let model, skeleton, mixer, numAnimations;
5 let modelSpeed = 0.001, modelReady = false;
6 let loader = new THREE.GLTFLoader();
7 loader.load('assets/Fox.glb',
8   function(gltf) {
9     model = gltf.scene;
10    scene.add(model);
11    const animations = gltf.animations;
12    mixer = new THREE.AnimationMixer(model);
13    numAnimations = animations.length;
14    for ( let i = 0; i !== numAnimations; ++ i ) {
15      let clip = animations[i];
16      let name = clip.name;
17      if(name=="Run") {
18        const action = mixer.clipAction( clip );
19        const clip2 = action.getClip();
20        action.enabled = true;
21        action.setEffectiveTimeScale( 1 );
22        action.setEffectiveWeight( 1 );
23        action.play();
24      }
25    }
26    animate();
27    modelReady = true;
28  },
29  function(xhr) { console.log((xhr.loaded/xhr.total*100)+"% loaded"); },
30  function(error) { console.log('An error happened'); }
31 );
```

```
33 function onWindowResize() {
34     camera.aspect = window.innerWidth/window.innerHeight;
35     camera.updateProjectionMatrix();
36     renderer.setSize(window.innerWidth, window.innerHeight);
37 }
38 function animate(now) {
39     if(modelReady) mixer.update(modelSpeed);
40     renderer.render(scene, camera);
41     requestAnimationFrame(animate);
42 }
43 window.addEventListener('resize', onWindowResize, false);
44 requestAnimationFrame(animate);
```