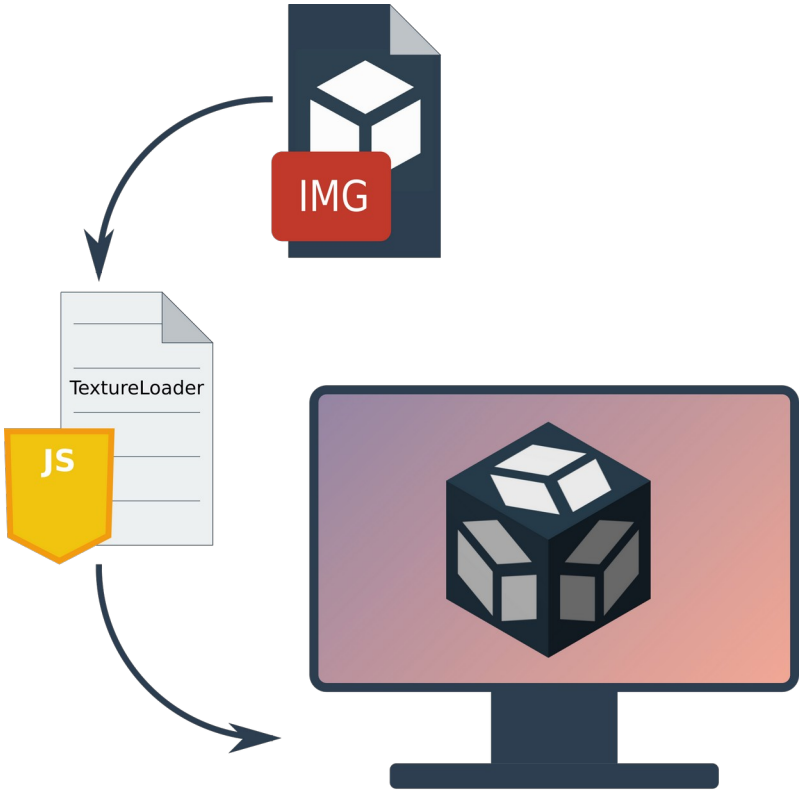
Three.JS University

Utilisation de textures dans Three.js

Utilisons des textures dans Three.js !

- Dans le monde de la 3D, une texture représente généralement une image appliquée sur la surface d’un objet 3D
- Ces images sont, pour la plupart, créées avec des outils externes
- Pour appliquer une texture sur un objet 3D, nous utiliserons les classes de la famille des Material

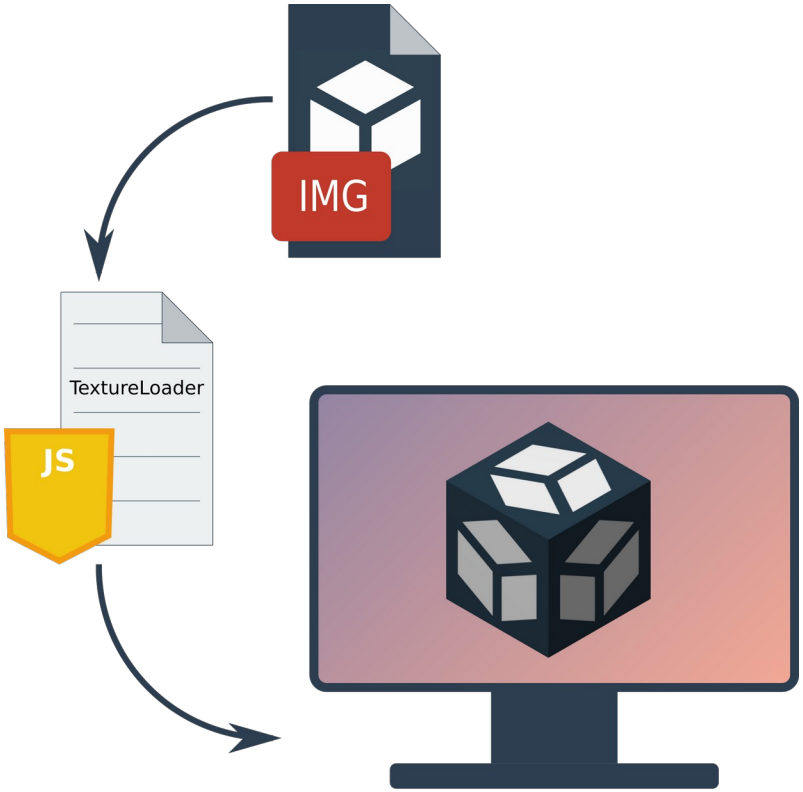
Charger une texture – La classe TextureLoader



- La classe **TextureLoader** est utilisée pour charger une image externe dans notre code
- La méthode **load** permet de charger un fichier
- Nous déployons l'image sur un objet 3D grâce a la propriété **map** de son **Material**

```
const loader = new THREE.TextureLoader();

const material = new THREE.MeshPhongMaterial({
  map: loader.load('cube_texture.png')
});
```

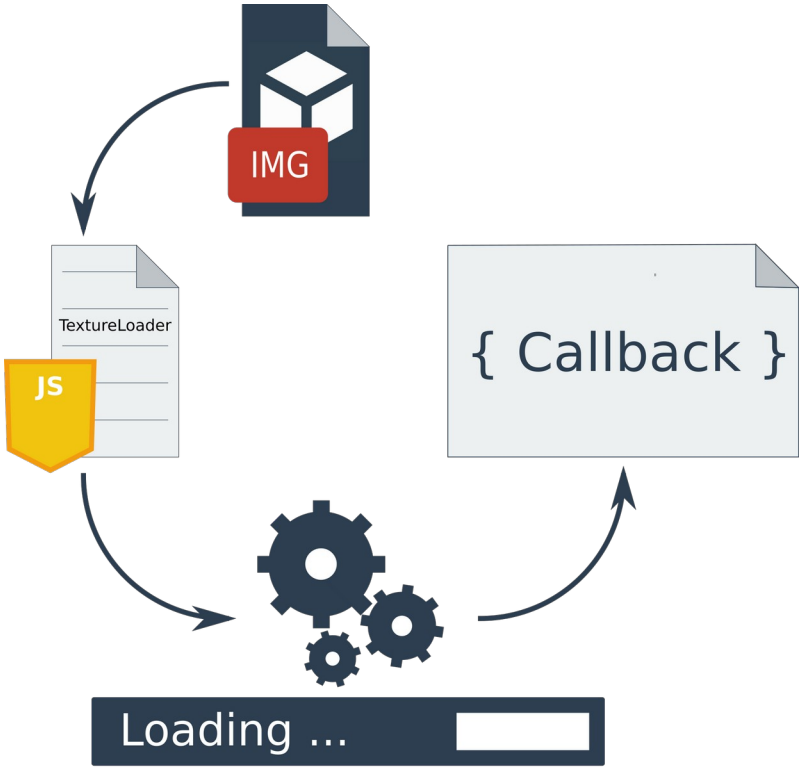
Charger une texture dans une variable



- Il est également possible de charger une texture dans une variable
- C’est une option intéressante si nous souhaitons réutiliser plusieurs fois la même texture

```
const loader = new THREE.TextureLoader();
var texture = loader.load('cube_texture.png')

[...]

const material = new THREE.MeshLambertMaterial({
    map: texture
});
const material_2 = new THREE.MeshLambertMaterial({
    map: texture
});
```

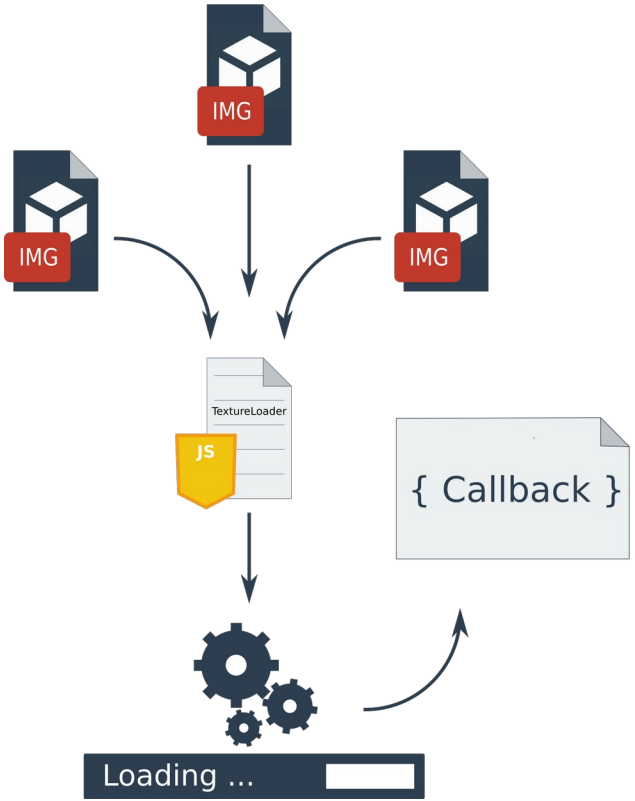
La fonction *callback* de *load*



- Il est nécessaire de vérifier que la texture est bien chargée avant de l'utiliser !
- Pour cela, il est possible d'utiliser la fonction *callback* de **load**
- Le code *callback* sera exécuté une fois le chargement terminé

```
const loader = new THREE.TextureLoader();
loader.load('cube_texture.png', (texture) => {
    // ---------------- callback ----------------
    const material = new THREE.MeshLambertMaterial(
    {
      map: texture,
    });

    const geometry = new THREE.BoxGeometry( 2, 2, 2 );
    var cube = new THREE.Mesh( geometry, material );
    scene.add( cube );
});
```

LoadingManager – Surveiller le chargement de plusieurs textures



- Si nous devons surveiller plusieurs chargements, il est possible d'utiliser la classe **LoadingManager**
- Cette classe fournis un callback unique, exécuté lorsque tous les chargements sont terminés

```
const loadMngr = new THREE.LoadingManager();
const loader = new THREE.TextureLoader(loadMngr);

texture = loader.load('cube_texture.png');
texture2 = loader.load('cube_texture2.png');
texture3 = loader.load('cube_texture3.png');
texture4 = loader.load('cube_texture4.png');

loadMngr.onLoad = () => {

    // code
};
```

Les options de filtrage – magFilter

Lorsque la texture projetée est plus grande que l'image d'origine, il existe deux options : **NearestFilter** et **LinearFilter**

- Ces deux options sont des valeurs possibles de la propriété **magFilter** des textures
- **NearestFilter** : Chaque pixel de la texture projetée est calculé à partir du pixel proportionnellement le plus proche sur l'image d'origine
- **LinearFilter** : Chaque pixel de la texture projetée est calculé à partir d'une moyenne de pixels proportionnellement les plus proches sur l'image d'origine

```
loader.load('pixel_art.png', (texture_linear) => {

    texture_linear.magFilter = THREE.LinearFilter;

    const material = new THREE.MeshPhongMaterial(
    {
      map: texture_linear,
    });
```

```
loader.load('pixel_art.png', (texture_nearest) => {

    texture_nearest.magFilter = THREE.NearestFilter;

    const material = new THREE.MeshPhongMaterial(
    {
      map: texture_nearest,
    });
```

Les options de filtrage – les Mips

- Les *Mips* sont les copies d’une texture dont la taille a été réduite
- Chaque *Mip* est deux fois plus petite que la *Mip* précédente
- Les pixels composant une *Mip* sont calculés à partir de la moyenne des pixels de la *Mip* précédente
- Les *Mips* sont utilisables par Three.js lorsqu’une texture est projetée à une taille inférieure aux dimensions de l’image d’origine

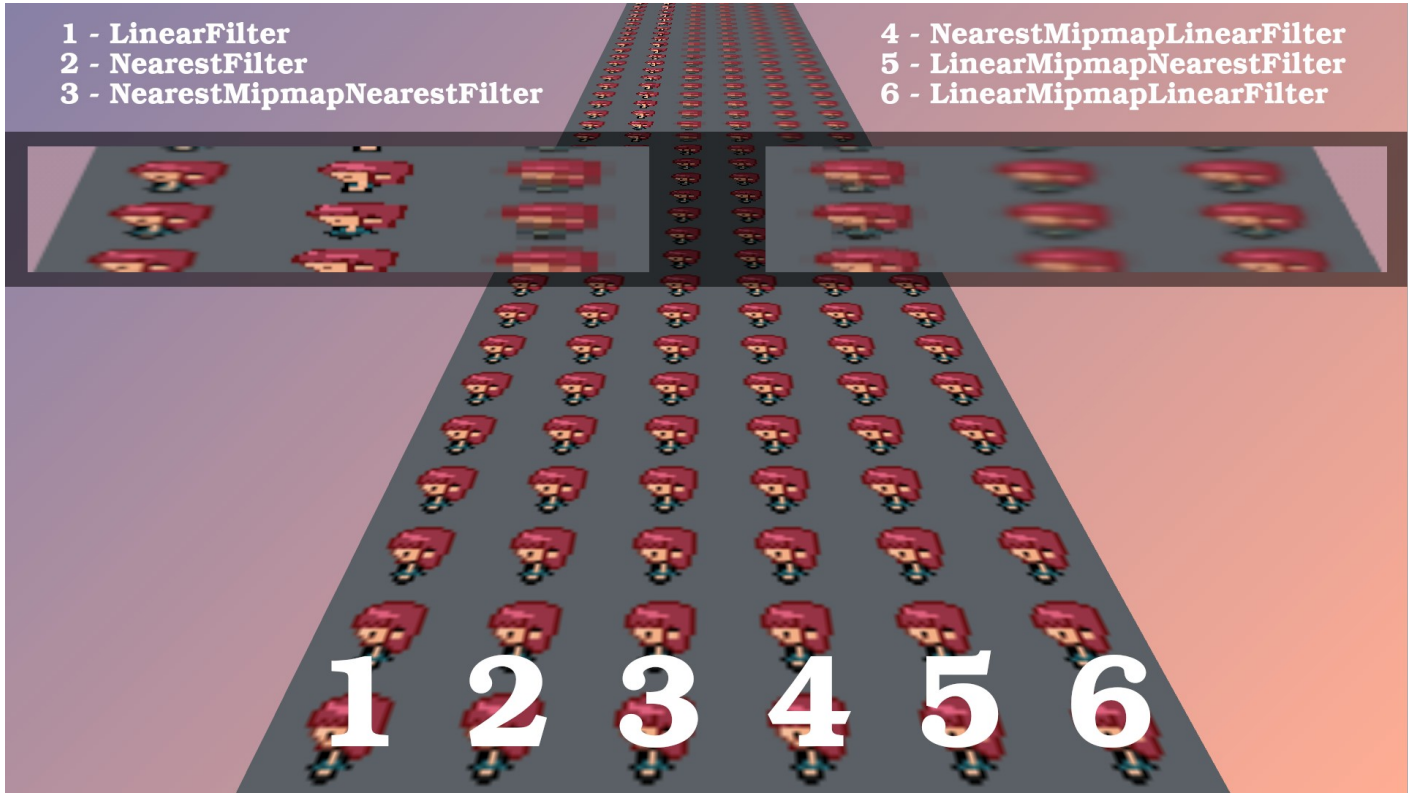
Les options de filtrage – minFilter

Lorsque la texture projetée est plus petite que l'image d'origine, il existe six options

Méthodes de calcul pour chaque pixel de la texture projetée :

- **NearestFilter**: À partir du pixel proportionnellement le plus proche sur l'image d'origine
- **LinearFilter** : À partir d'une moyenne de pixels proportionnellement les plus proches sur l'image d'origine
- **NearestMipmapNearestFilter** : À partir du pixel proportionnellement le plus proche sur la *Mip* appropriée
- **NearestMipmapLinearFilter** : Deux *Mips* sont sélectionnées, le pixel le plus proche sur chaque *Mip* est choisi, puis, nous calculons la moyenne des deux pixels
- **LinearMipmapNearestFilter** : À partir de la moyenne des quatre pixels proportionnellement les plus proches sur la *Mip* appropriée
- **LinearMipmapLinearFilter** : Deux *Mips* sont sélectionnées, les quatre pixels les plus proches sur chaque *Mip* sont choisis, puis, nous calculons la moyenne de ces huit pixels

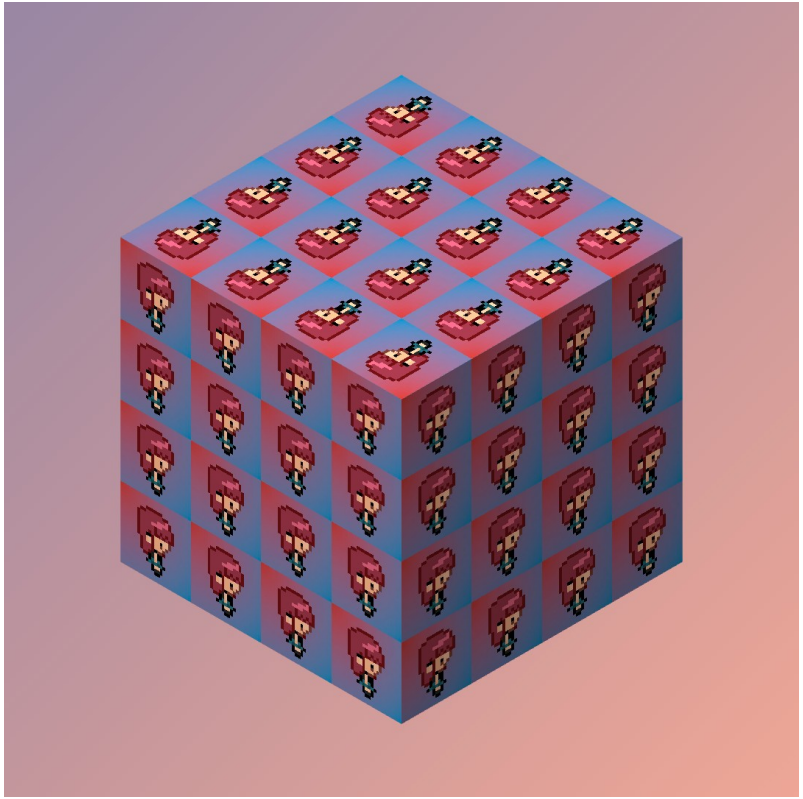
Les options de filtrage – minFilter



Nous utilisons la propriété **minFilter** de notre texture :

```
const loader = new THREE.TextureLoader();

loader.load('pixel_art.png', (texture_linear) => {

    texture_linear.minFilter = THREE.LinearFilter;

    const material = new THREE.MeshBasicMaterial(
    {
      map: texture_linear,
      side : THREE.DoubleSide
    });

    // ---------------- 3D CUBE ----------------
    const geometry = new THREE.PlaneGeometry( 2, 2 );
    var plane = new THREE.Mesh( geometry, material );
```

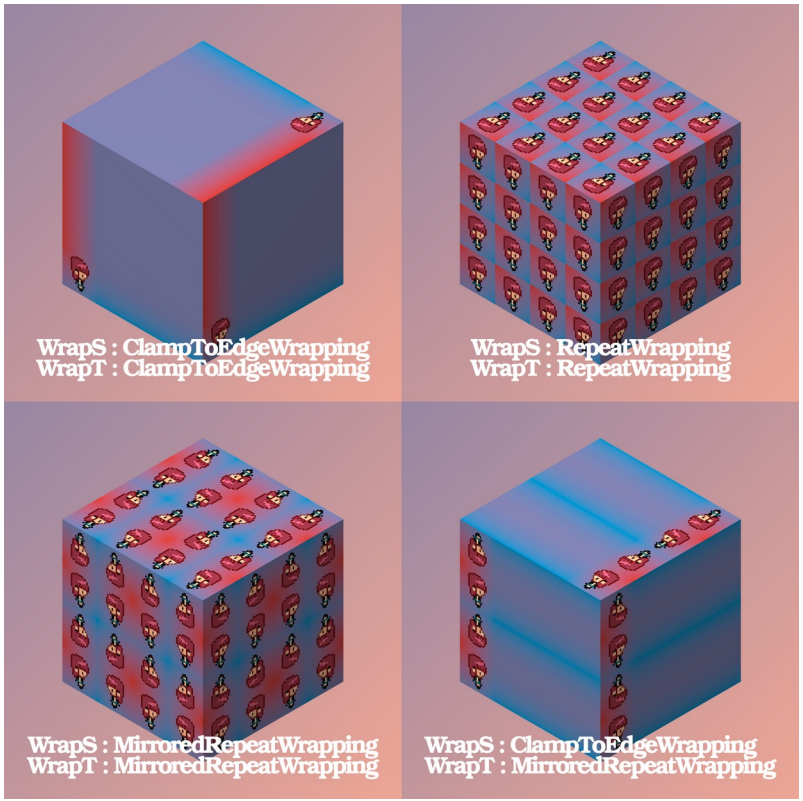
Répétition d’une texture



- La propriété **wrapS** est utilisée pour définir la règle de répétition horizontale d’une texture
- La propriété **wrapT** est utilisée pour définir la règle de répétition verticale d’une texture
- Il existe trois règles de répétition : **ClampToEdgeWrapping**, **RepeatWrapping** et **MirroredRepeatWrapping**

```
const loader = new THREE.TextureLoader();

loader.load('pixel_art.png', (texture) => {

    texture.wrapS = THREE.RepeatWrapping;
    texture.wrapT = THREE.ClampToEdgeWrapping;
```

Répétition d’une texture

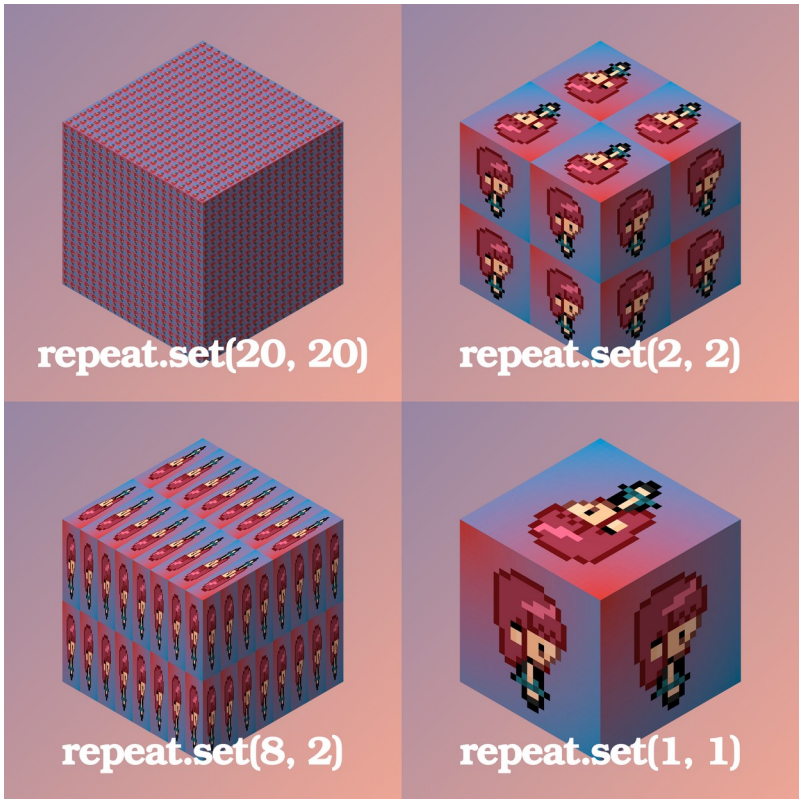


- **ClampToEdgeWrapping** : Pas de répétition, le dernier pixel est répété à l’infini
- **RepeatWrapping** : La texture est répétée
- **MirroredRepeatWrapping** : La texture est répétée en miroir

Par défaut, les textures ne sont pas répétées
(**ClampToEdgeWrapping**)

```
const loader = new THREE.TextureLoader();

loader.load('pixel_art.png', (texture) => {

    texture.wrapS = THREE.RepeatWrapping;
    texture.wrapT = THREE.ClampToEdgeWrapping;
```

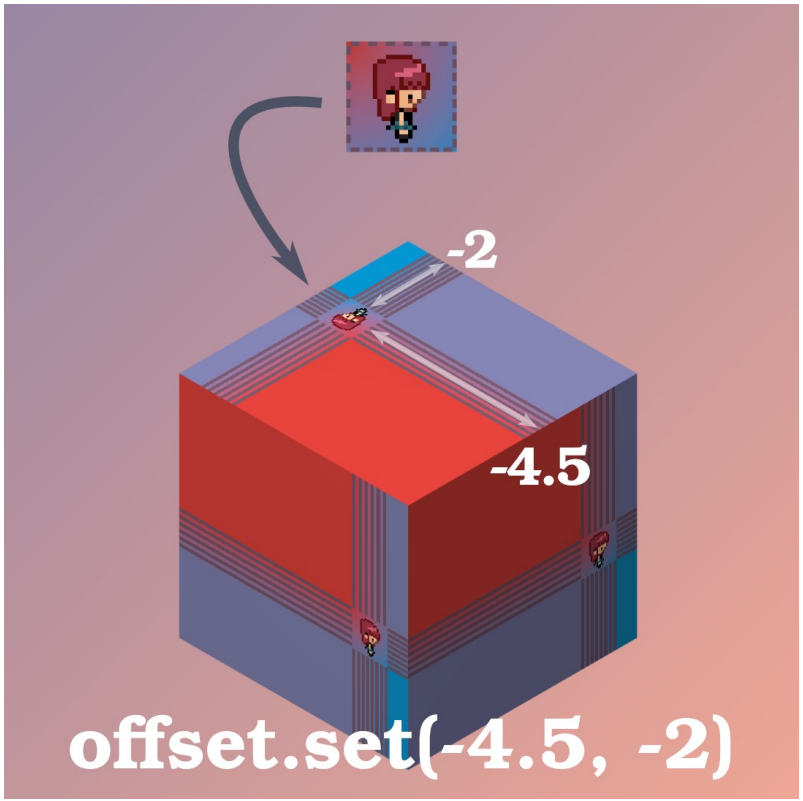
Répétition d’une texture



- Le nombre de répétitions est défini avec la méthode **set** de la propriété **repeat** d’une texture
- Cette méthode accepte deux paramètres : Le nombre de répétitions horizontales et verticales

```
const horizontally = 8;
const vertically = 2;
texture.repeat.set(horizontally, vertically);

const material = new THREE.MeshLambertMaterial(
{
  map: texture,
});
```

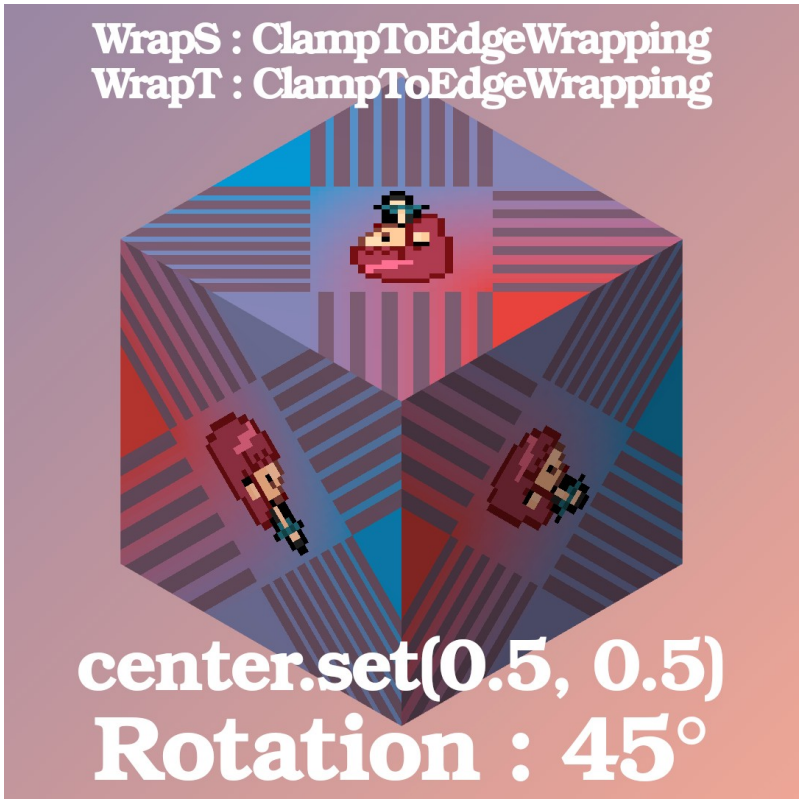
Décalage d’une texture



- Il est possible définir un décalage de texture avec la méthode **set** de la propriété **offset** d’une texture
- Cette méthode accepte deux paramètres : Le décalage horizontal et vertical
- Le décalage est exprimé en « *taille d’une unité de texture* »
- L’exemple de gauche à été réalisé avec une répétition de (**6, 6)** et la règle de répétition **ClampToEdgeWrapping** pour les deux axes

```
const horizontally = 6;
const vertically = 6;
texture.repeat.set(horizontally, vertically);

const offset_x = -4.5;
const offset_y = -2;
texture.offset.set(-4.5, -2);
```

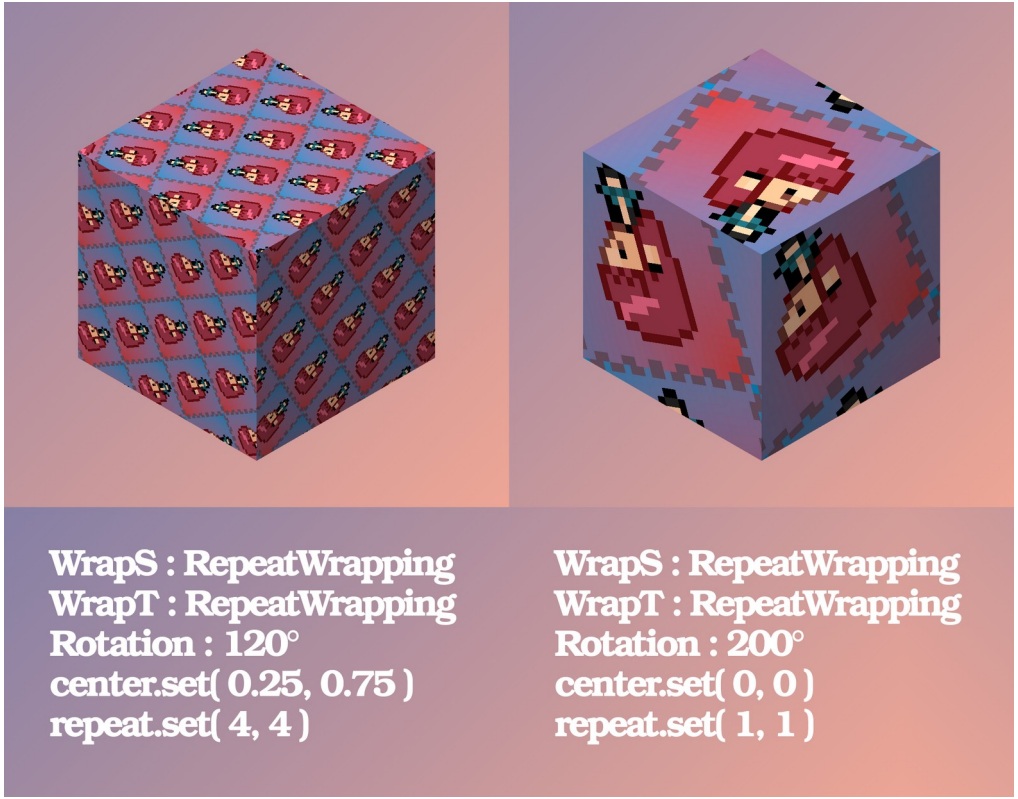
Rotation d’une texture



- Il est possible définir une rotation de texture avec la propriété **rotation** d’une texture
- La valeur de la propriété **rotation** est exprimée en radians
- I est possible définir le centre de rotation de la texture avec la méthode **set** de la propriété **center**
- Le centre de rotation est également exprimé en « *taille d’une unité de texture* »
- L’exemple de gauche à été réalisé avec une répétition de (**2, 2)** et la règle de répétition **ClampToEdgeWrapping** pour les deux axes

```
texture.wrapS = THREE.ClampToEdgeWrapping;
texture.wrapT = THREE.ClampToEdgeWrapping;

texture.repeat.set(2, 2);

texture.center.set(0.5, 0.5);
texture.rotation = THREE.MathUtils.degToRad(45);
```

Rotation d’une texture



Toutes les propriétés de textures sont fortement liées entre elles !

Utilisation de map avec les autres propriétés de Material

- Comme expliqué dans ce chapitre, les textures s'utilisent dans les matériaux avec la propriété **map**
- **map** est également exploitable aux cotés d'autres propriétés de matériaux

Utilisation de textures dans Three.js

Si vous êtes prêts, passons à la pratique !