

# **Rapport projet Réalité Virtuelle**

Clothilde Haristoy et Ewen Jegouzo

|   |           |
|---|-----------|
| <b>Introduction.....</b>                                    | <b>3</b>  |
| <b>Organisation du projet.....</b>                          | <b>3</b>  |
| <b>Construction du musée.....</b>                           | <b>3</b>  |
| Les salles.....   | 3         |
| Texture des murs.....                                       | 4         |
| Les portes.....   | 4         |
| <b>Fichiers 3D, détails et animations.....</b>              | <b>5</b>  |
| Ray pointer détection d'objet dans le champs de vision..... | 5         |
| Descriptions textuelles.....                                | 6         |
| Importation de fichier 3D extérieur.....                    | 7         |
| Animation des meshs.....                                    | 8         |
| Création d'Amer.....  | 9         |
| <b>Organisation des fichiers.....</b>                       | <b>9</b>  |
| <b>Remarque.....</b>  | <b>10</b> |

## **Introduction**

Le musée est consacré à la renaissance appliquée à Star Wars. Nous avons 3 salles en bas qui représentent chacune un univers de différentes planètes de Star Wars. La salle sur la droite représente la planète Tatooine, c'est sur cette planète que le Jedi Obi Wan Kenobi renaît de ses cendres, il redevient un jedi pour s'occuper du jeune Luke Skywalker. La salle de milieu représente la planète Naboo, c'est sur cette planète que l'on pourra voir une scène du mariage de Padmé vêtu d'une longue robe. Cette robe ainsi que certains costumes de star wars ont été inspiré de la renaissance et de son style drapé. La dernière salle à gauche représente quant à elle l'étoile noire, base de Dark Vador et naissance de l'ennemi. La mezzanine représente quant à elle une scène mythique de star wars où Luke voit apparaître plusieurs Jedi pourtant morts.

## **Organisation du projet**

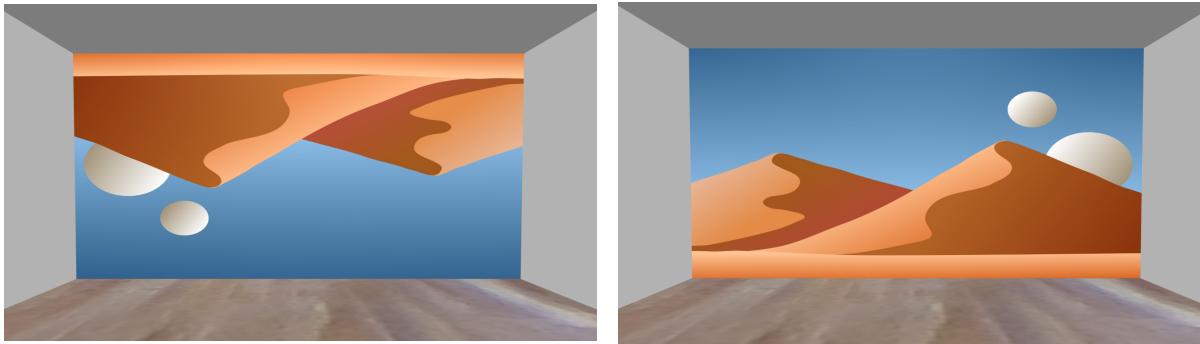
Ewen étant alternant nous avions créé un git depuis le début. Nous avons travaillé ensemble durant le P1 sur la construction du musée et en P2 j'ai avancé de mon côté, Ewen reprendra l'avancée de son côté lorsqu'il reviendra à l'Enib en P2 de printemps.

## **Construction du musée**

### **Les salles**

Nous souhaitions créer des univers différents pour chaque salle. Nous avons créé une fonction par salle, chaque salle est composée de 6 murs: 4 murs (Nord, Sud, Est, Ouest lorsque vous entrez dans une salle le mur Nord est situé devant vous) plus un sol et un plafond.

Chaque mur est associé à un faceUV\_x (avec x compris entre 1 et 6) et chaque faceUV\_x est un tableau de 6 caractères, un par face. Ainsi on peut appliquer une texture sur seulement une face, on applique une couleur par défaut à chaque face et ensuite on superpose la texture sur la face qui apparaît dans la salle. Cela permet d'éviter de mettre des textures qui ne servent à rien. En travaillant avec des BABYLON.Vector4 nous pouvions positionner le coin bas gauche et le coin haut droit de la texture. Pour des soucis de réalisme nous avons fini par les remplacer par des textures.



```
faceUV_2[i] = new BABYLON.Vector4(
    1, 0, 0, 1);
```

```
faceUV_2[i] = new BABYLON.Vector4(
    0, 1, 1, 0);
```

Sur les images ci-dessus on applique notre texture sur la face 0 du mur associé à `faceUV_2`. On peut voir que si on change les coordonnées des coins, l'image change de sens.

Nous avons ensuite abandonné le fonctionnement des `faceUV` et avons simplement appliqué les textures aux murs.

### Texture des murs

La texture des murs est créée par une superposition d'images. La `diffuseTexture` est la couleur de base de la texture. Le `bumpTexture` permet quant à lui de donner du relief à la texture, pour accentuer le relief de la texture on peut aussi appliquer un Parallax. (Nous n'appliquons pas de parallax sur les textures utilisées car elles ne s'y prêtent pas.) La `specTexture` modifie la réflexion de la lumière sur les textures. Enfin `l'ambiantTexture` modifie l'éclairage global de la texture. On applique également de la roughness pour avoir un rendu moins brillant qui réfléchit moins la lumière. Grâce au `uScale` et `vScale` on peut régler l'échelle des textures sur la surface, `u` étant sur l'axe des `x` et `v` sur l'axe des `y`. Plus le nombre est grand, plus la texture apparaît petite.

### Les portes

Les ouvertures des portes sont faites avec la fonction `creuser()`, mais pour pouvoir appliquer un booléen il faut travailler avec deux mesh or la fonction `PRIMS.wall()` retourne un `TransformNode`. Nous avons donc créer les murs ou nous devions créer des portes avec la fonction `PRIMS.box`.

Pour l'ouverture des portes nous avons utilisé les notions de trigger et d'interpolation. Les portes s'ouvrent de manière coulissante lorsque le volume englobant de la caméra `Boxcamera` entre en contact avec le mesh de la porte grâce à la fonction

`OnIntersectionEnterTrigger`. Ensuite les portes sont animées grâce à la fonction `InterpolateValueAction()` avec laquelle on règle la position finale et la durée de l'animation.

```
//Animation de la porte
porte_s1.actionManager = new BABYLON.ActionManager(scene);

let ouverture_s1 = new BABYLON.InterpolateValueAction(
{
    trigger : BABYLON.ActionManager.OnIntersectionEnterTrigger,
    parameter : {
        8,
        1000,
    };
    let fermeture_s1 = new BABYLON.InterpolateValueAction(
    {
        trigger : BABYLON.ActionManager.OnIntersectionExitTrigger,
        parameter : {
            10,
            1000,
        };
    }

    porte_s1.actionManager.registerAction(ouverture_s1);
    porte_s1.actionManager.registerAction(fermeture_s1);
```

## Fichiers 3D, détails et animations

### Ray pointer détection d'objet dans le champs de vision

La première solution à été d'utiliser un cône 3D invisible comme détecteur de champ de vision lorsque que celui-ci intersecte avec n'importe quel autre objet de l'environnement. Cependant le cône pouvait intersecter plusieurs objets dus à son volume et au fait qu'il traverse les murs et détecte les objets de l'autre côté.

La solution à été d'utiliser une fonction `castRay()` qui permet de tracer un rayon face à la caméra. Ce rayon est très utile puisqu'il permet de détecter quels objets nous visons avec la caméra. Il est bien plus pratique d'utiliser ce rayon plutôt que d'utiliser l'intersection de 2 mesh 3D puisqu'avec le rayon, un seul et unique objet peut être pointé à la fois. Évidemment ce rayon est attaché à la caméra, il pointe vers l'avant et sa longueur est ajustable.

```

function castRay() {
    var origin = camera.position;
    var direction = camera.getDirection(BABYLON.Vector3.Backward())
    var length = 5;

    var ray = new BABYLON.Ray(origin, direction, length);

    var hit = scn.pickWithRay(ray);

    if (hit.pickedMesh) {
        /* object detected */
    }
}

/* run every engine loop */
scn.registerBeforeRender(function () {
    castRay();
});

```

La détection d'objet dans le champ visuel est réalisé juste avant chaque tour de boucle d'affiche du moteur 3D avec le:

```
scn.registerBeforeRender();
```

### **Descriptions textuelles**

Maintenant que nous avons une détection d'objet fiable, nous pouvons rajouter l'affichage d'une description lorsqu'un objet déterminé est visé par le Ray pointer. Pour faire cela un élément html/css est ajouté en "dur" dans le index.html avec des valeurs d'opacités et textuelles nulles.

A la création d'un objet avec description, des valeurs de description sont entrées. Celles-ci sont ensuite récupérées en javascript par le moteur de jeu et insérées dans l'élément html.

```

const poster_x = POSTER.withDescription(
    "poster_x",
    murOuest,
    "Description Title1 &#10;&#10;Augue ac adipiscing quis, arcu auctor!
Elementum. Non vel vel augue odio et in et est, integer, porta sed parturient
rhoncus habitasse! Et porttitor duis pulvinar pulvinar proin ac augue ac sagittis
scelerisque, elementum integer eros. Sed, nec! Porta, dapibus in quis elementum
penatibus adipiscing, nec adipiscing adipiscing purus lacus odio dolor diam a.",
    new BABYLON.Vector3(5, 2, -0.2),

```

```

        {hauteur:2, largeur:1.5},
        scene
    );

```



## Importation de fichier 3D extérieur

Nous avons dans un premier temps essayé d'importer des fichiers .babylon, malheureusement impossible d'exporter le fichier au format babylon avec des textures, même en les plaçant dans le même dossier. Sous blender seul les fichiers sans textures s'exportent correctement en babylon. Nous avons ensuite essayé d'importer des fichiers .gltf mais ça ne fonctionnait pas. Nous avons donc opté pour des fichiers .glb peu volumique et qui s'affichent correctement sous babylon.

Chaque objet importé possède une hitbox. Un volume englobant qui permet de ne pas passer au travers de l'objet. Nous pouvons régler les paramètres des hitbox (taille, offset par rapport à l'origine du modèle 3D, visibilité) indépendamment des paramètres des objets 3D importés.

```

MESH.create(
    "clay_pot",
    { scale: 1 },
    "clay_pot1.glb",
    new BABYLON.Vector3(-14, 0, -6),
    {x: 0, y:0, z:0},
    { width: 0.5, height: 0.5, depth: 0.5 },

```

```
    new BABYLON.Vector3(0, 0.5, 0),  
    false,  
    scene  
) ;
```

## Animation des meshs

Pour animer les meshs nous utilisons plusieurs méthodes différentes.

Pour les portes les animations se déclenchent lorsque deux volume entrent en collision, comme expliqué plus haut. Pour les mesh qui ont une animation continue on définit une fonction dans `.registerBeforeRender()` par exemple l'incrémentation à l'infinie d'une rotation.

```
scn.registerBeforeRender(function () {  
    mesh.rotation.x = mesh.rotation.x + rotation_animation.rot_x;  
    mesh.rotation.y = mesh.rotation.y + rotation_animation.rot_y;  
    mesh.rotation.z = mesh.rotation.z + rotation_animation.rot_z;  
});
```

Pour des animations plus compliquées, on met en place un système de keyFrame. On va définir à des temps précis la valeur du paramètre à animer. Ensuite grâce à des modes on peut choisir de faire tourner cette animation en boucle ou non. (Cette méthode a été utilisée au début du projet puis remplacée par la première).

La dernière méthode utilisée est de réaliser un tracé en plaçant des points à des endroits fixes et de faire évoluer les mesh sur ces tracés.

```
const points = [];  
//list of track points  
points.push(position);  
points.push(position.x + 10);  
points.push(position.z + 7);  
points.push(position.x - 10);  
points.push(points[0]);  
  
//draw track  
BABYLON.MeshBuilder.CreateLines("square", {points: points});  
//after covering dist apply turn  
const slide = function (turn, dist) {  
    this.turn = turn;  
    this.dist = dist;
```

Certains éléments sont quant à eux déjà animés depuis le logiciel blender et n'ont pas besoin d'être animés dans le code.

### Création d'Amer

Le musée comporte deux amers. Un dans le hall et un au fond de la salle Tatooine face au tableau d'Obi wan.

Les amer sont représentés par des sphères grises semi transparentes. Lorsque le curseur passe sur un amer, la position de la caméra est téléportée jusqu'à l'amer. Pour connaître la position de notre curseur nous avons installé au milieu de l'écran un réticule. Lors du premier clic sur la page du musée, il faut cliquer sur le réticule. Ainsi nous connaissons la position de notre curseur à tout moment.

```
sphere.actionManager.registerAction(  
    new BABYLON.SetValueAction(BABYLON.ActionManager.OnPointerOverTrigger,  
        cam,  
        "position",  
        new BABYLON.Vector3(sphere.position.x, 2.1, sphere.position.z)));
```

### Organisation des fichiers

Pour mieux se retrouver dans le code, le segmenter en objets et éviter de se retrouver avec un fichier prim d'une longueur phénoménale, nous avons créé des fichiers objets supplémentaires tel que POSTER.js, MESH.js, DOOR.js, ou AMER.js .

POSTER.js contient une fonction `Create()` basique qui permet de créer un poster d'une certaine taille avec une image précise. Ensuite la fonction `WithDescription()` reprend les paramètres de la fonction `create` et permet de positionner le poster et d'y associer un texte de description. Pour le fichier MESH.js nous ne pouvons pas retourner notre mesh à cause de la fonction `meshTask.onSuccess()` . Nous avons alors dupliqué la fonction en ajoutant des fonctionnalités, bien que ça ne soit pas la meilleure façon de faire cela évite de devoir programmer des fonctionnalités non utilisées pour certains mesh.

Les fichiers DOOR et AMER fonctionnent sur le même principe que le fichier POSTER, nous créons une fonction `Create()` et ensuite ajoutons d'autres fonctions avec des fonctionnalités supplémentaires qui reprennent les bases de la fonction `Create()`.

## **Remarque**

La moitié des salles du musée ont été réalisées. Le projet reste lourd à cause des textures et des éléments 3D importés. Le lancement du programme peut buggé il suffit de réactualiser la page pour qu'il fonctionne correctement.