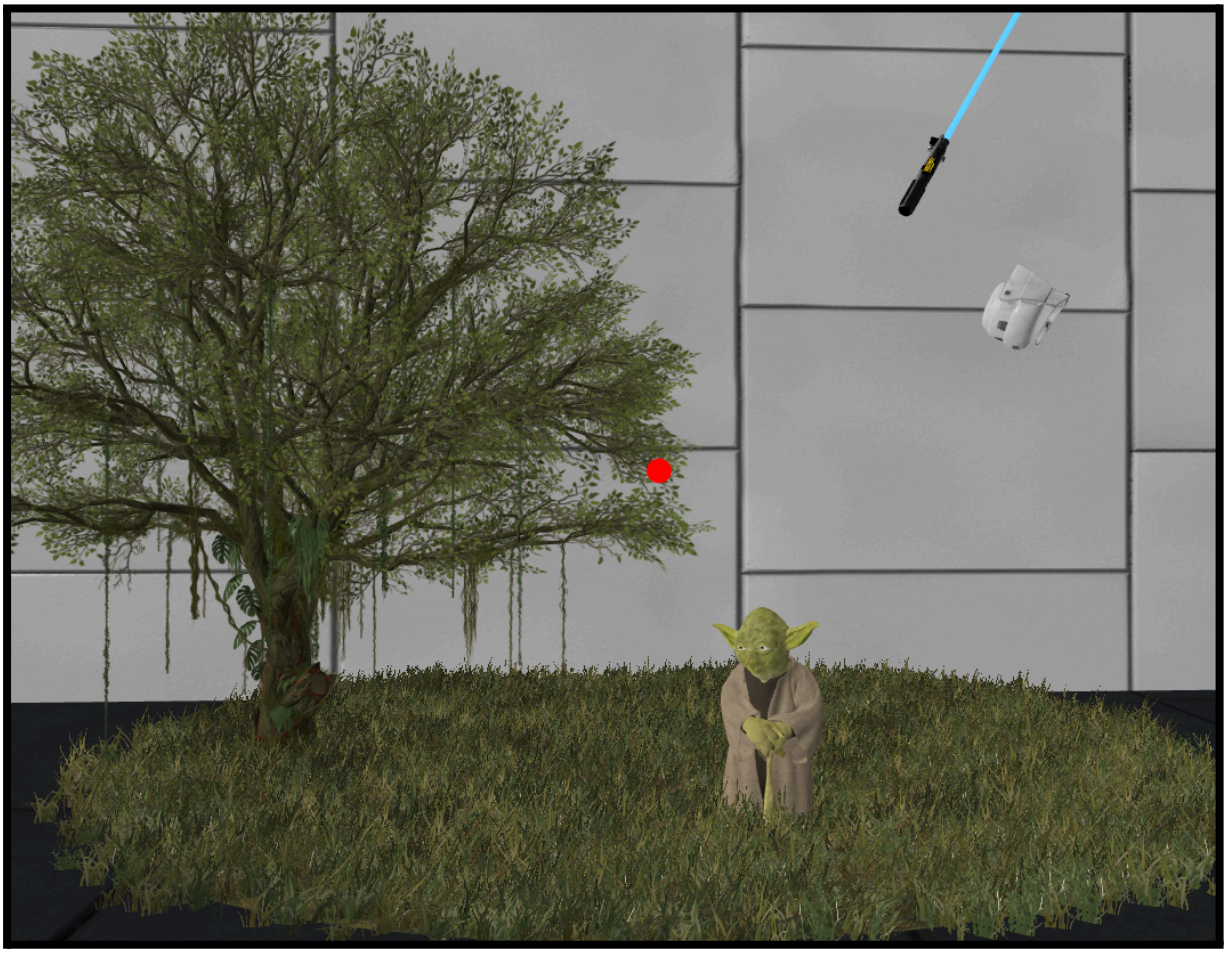
Rapport de projet de Réalité Virtuelle

Environnement virtuel dans un musée



BAGUIAN Harouna
JEGOUZO Ewen
(HARISTOY Clothilde en P1)

20/06/2024

Sommaire

Introduction

Pour notre projet de réalité virtuelle de 5e année nous avons dû réaliser un musée. Pour traiter du thème de la Renaissance, nous avons utilisé l'univers de Star Wars. En effet, dans la saga, nous pouvons retrouver plusieurs type de Renaissance :

- Sur Tatooine - la renaissance de la force.
- Le lac de Côme - les tenues de la Renaissance

Notre musée est constitué d'un hall et de trois salles au rez-de-chaussée. La salle à droite est dédiée à la planète Tatooine, où le Jedi Obi-Wan Kenobi fait son retour pour veiller sur le jeune Luke Skywalker. La salle centrale met en scène la planète Naboo, avec une représentation du mariage de Padmé dans une robe longue, inspirée de la Renaissance. La dernière salle, à gauche, est consacrée à l'Étoile Noire, la base de Dark Vador et l'origine de l'ennemi.

A l'étage, nous pourrons retrouver la scène mythique de la fin de l'épisode 6 lorsque Luke voit ses différents maîtres pourtant morts.

Étant alternant, j'ai commencé ce projet avec clothilde en P1 du semestre d'automne. Je suis ensuite partie en entreprise en P2, mais sur mon temps libre, j'ai pu continuer à l'aider sur le développement de certaines fonctionnalités.
Au semestre de printemps, Harouna m'a rejoint et nous avons pu continuer le développement du musée.

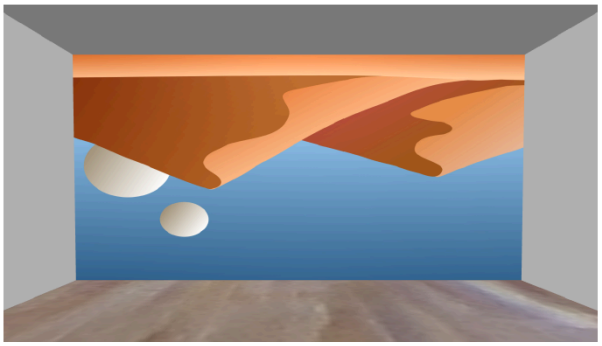
1. La construction des salles

- Les salles:

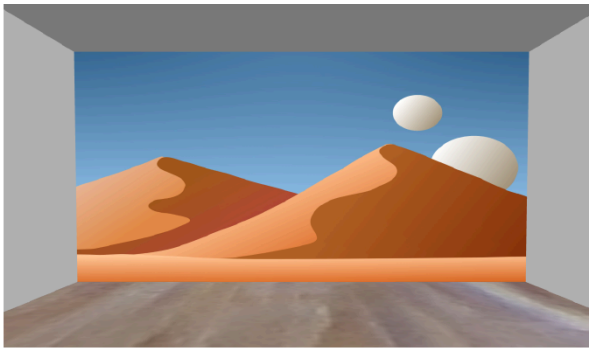
Nous avions pour objectif de créer des environnements distincts pour chaque salle, ce qui nous a amenés à développer une fonction dédiée à chaque salle. Chacune de ces salles est structurée avec quatre murs principaux : le Nord, le Sud, l'Est et l'Ouest. Lorsque l'on pénètre dans une salle, le mur Nord se trouve directement en face de nous.

Chaque mur est associé à une **faceUV_x**, où x varie de 1 à 6. Chaque **faceUV_x** correspond à un tableau de 6 caractères, représentant chacun une face. Cette méthode nous permet de gérer l'application de textures uniquement sur les faces nécessaires, évitant ainsi le gaspillage de ressources inutiles. Par défaut, chaque face reçoit une couleur de base, sur laquelle nous superposons la texture appropriée selon la configuration de la salle.

Pour ce faire, nous avons utilisé les fonctionnalités de BABYLON.Vector4 pour déterminer avec précision les coins inférieur gauche et supérieur droit de chaque texture. Cependant, afin de renforcer le réalisme visuel, nous avons progressivement remplacé ces vecteurs par des textures directement applicables. Ce choix nous a permis d'améliorer significativement l'immersion et l'aspect esthétique des environnements que nous avons créés.



```
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, nous avons illustré l'application de notre texture sur la face 0 du mur associé à **faceUV_2**. Ce choix était crucial car il nous permettait de contrôler l'orientation de l'image en ajustant les coordonnées des coins, assurant ainsi que la texture s'affiche correctement dans le jeu.

Cependant, après avoir évalué différents aspects et pour simplifier notre processus de développement, nous avons décidé d' abandonner l'utilisation des **faceUV**. À la place, nous avons opté pour une approche plus directe où nous appliquons les textures directement sur les murs. Cette décision a simplifié notre pipeline de création d'environnements tout en maintenant la qualité visuelle et en réduisant la complexité technique.

- Texture des murs

La création des textures des murs implique une superposition d'images qui combine différents effets pour enrichir le rendu visuel. La **diffuseTexture** définit la couleur de base de la texture, tandis que la **bumpTexture** ajoute du relief pour accentuer les détails. Si nécessaire, un effet de **Parallax** peut être appliqué pour renforcer le relief, bien que dans notre cas spécifique, nous avons choisi de ne pas l'utiliser car il n'était pas adapté aux textures en question.

La **specTexture** ajuste la réflexion de la lumière sur la texture, tandis que **l'ambiantTexture** modifie l'éclairage global de la surface texturée. Pour atténuer la brillance excessive et obtenir un rendu plus réaliste, nous appliquons également une **roughness** qui réduit la réflexion de la lumière.

Les paramètres **uScale** et **vScale** permettent de régler l'échelle des textures sur la surface : u correspond à l'axe des **x** et **v** à l'axe des y. Augmenter ces valeurs réduit la taille apparente de la texture, tandis que les diminuer la fait paraître plus grande. Cette flexibilité nous permet d'ajuster précisément l'apparence des textures pour correspondre aux dimensions et à l'ambiance de chaque salle créée.

- 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 créé 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.

```
//Ouverture et fermeture de la porte
door.actionManager = new BABYLON.ActionManager(scn);

let ouverture = new BABYLON.InterpolateValueAction(
    {
        trigger : BABYLON.ActionManager.OnIntersectionEnterTrigger,
        parameter : {
            mesh : boxCamera
        }
    },
    door,
    'position.z',
    positionDoor.z-2,
    1000,
);

let fermeture = new BABYLON.InterpolateValueAction(
    {
        trigger : BABYLON.ActionManager.OnIntersectionExitTrigger,
        parameter : {
            mesh : boxCamera
        }
    },
    door,
    'position.z',
    positionDoor.z,
    1000,
);

door.actionManager.registerAction(ouverture);
door.actionManager.registerAction(fermeture);
```

2. Les fonctionnalitées

- **Importation de fichier 3D extérieur**

Chaque objet importé est doté d'une hitbox, un volume englobant destiné à empêcher les collisions non souhaitées avec l'objet. La hitbox joue un rôle crucial en garantissant que les interactions physiques respectent les limites de l'objet, évitant ainsi que les personnages ou autres objets ne traversent les surfaces.

Nous avons la possibilité de régler divers paramètres des hitbox de manière indépendante par rapport aux paramètres des objets 3D importés. Cela inclut la taille de la hitbox, son décalage par rapport à l'origine du modèle 3D, et sa visibilité. En ajustant ces

paramètres, nous pouvons affiner la précision des collisions et optimiser le comportement physique des objets dans notre environnement virtuel, tout en conservant une flexibilité maximale pour l'intégration et la personnalisation des modèles 3D.

```
MESH.create(
  "yoda_levitation",
  { scale: 0.02 },
  "master_yoda.glb",
  new BABYLON.Vector3(8, 0, 10),
  { x: 0, y: 0, z: 0 },
  { width: 0.5, height: 0.5, depth: 0.5 },
  new BABYLON.Vector3(0, 3, 0.5),
  false,
  scene
);
```

- **Ajout d’un agent virtuel**

Ce code définit une fonction ***TurningMesh()*** qui crée et anime le bébé yoda dans une scène. Il initialise les propriétés du personnage, comme sa position, sa rotation, et sa hitbox. Ensuite, il trace un chemin en utilisant des points et des rotations prédéfinis. Le personnage suit ce chemin en se déplaçant et en tournant à chaque point de virage. Le personnage vérifie constamment la proximité de la caméra, et si elle est proche, il affiche un message approprié selon son orientation. À une position spécifique (-9.93, 4.8, 7.41), cette position se trouve devant le tableau de Yoda à la mezzanine, le personnage s'arrête pendant 5 secondes et dit "Regarde le tableau" si la caméra est également proche. Après cette pause, il reprend son mouvement le long du chemin défini.

En gros, le code fait en sorte que Baby Yoda suive une trajectoire, s'arrête à une certaine position pour dire un message, puis continue son chemin après une pause de 5 secondes. Voici un exemple ci-dessous :

et dans la console :

```
boite_camera
Hello, je suis Baby Yoda et je parle
Regarde le tableau
```

L'ensemble du code intéressant se trouve dans le mesh.js.

- **Amers de téléportations**

Il y a plusieurs amers de téléportation dans le musée, il y a 4 dans le hall, 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 vertes. 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.

```
export function Create(
    name,
    opts,
    scn,
    cam,
    ret)
{
    let options     = opts || {} ;
    let diameter    = options.diameter || 0.5 ;
    let position    = options.position;
    let visibility  = options.visibility || 1 ;

    const sphere = BABYLON.MeshBuilder.CreateSphere(name, {diameter: diameter}, scn);
    sphere.position = position;
    sphere.visibility = visibility;

    var mat_sphere = new BABYLON.StandardMaterial(name, scn);
    mat_sphere.specularColor = new BABYLON.Color3(0,0,0);
    mat_sphere.diffuseColor = new BABYLON.Color3(0.4, 1, 0.4);
    sphere.material = mat_sphere;

    sphere.actionManager = new BABYLON.ActionManager(scn);

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

    return sphere;
}

```

- **Jouer du son à l’ouverture de la porte**
  Pour ajouter une fonctionnalité de son à l'ouverture de la porte dans la scène Babylon.js, nous avons créé et configuré un son en utilisant BABYLON.Sound avec le fichier audio souhaité et des paramètres adaptés, comme l'activation du son spatial et la désactivation de la lecture automatique. Ensuite, nous avons défini une action de collision en utilisant BABYLON.ExecuteCodeAction, qui déclenche la lecture du son lorsque la caméra (boxCamera) entre en collision avec la porte. Enfin, nous avons enregistré cette action dans le gestionnaire d'actions de la porte. Voici le code complet pour cette fonctionnalité:

```
const doorOpenSound = new BABYLON.Sound(
    "doorOpen",
    "./assets/sound/sliding-noise-v2-83483.mp3",
    scene,
    null,
    { spatialSound: true, autoplay: false }
  );

  const playSoundAction = new BABYLON.ExecuteCodeAction(
    {
      trigger: BABYLON.ActionManager.OnIntersectionEnterTrigger,
      parameter: {
        mesh: boxCamera,
      },
    },
    function () {
      doorOpenSound.play();
    }
  );

  door.actionManager.registerAction(playSoundAction);
```

3. Remarques

Le musée est assez lourd et nous avons préféré améliorer les fonctionnalités plutôt que le visuel. Ainsi, nous avons retravaillé sur la téléportation, l'ajout d'une fonctionnalité avec la musique (qui n'est pas parfaite), complexification dans le déplacement des objets, ajout d'un agent, etc…

4. Conclusion

Notre projet de réalité virtuelle, un musée immersif sur le thème de la Renaissance via l'univers de Star Wars, a été un défi stimulant. Nous avons créé des environnements captivants avec des textures précises et des fonctionnalités interactives telles que des portes animées et des téléportations. Bien que nous ayons simplifié certaines méthodes pour optimiser la création des salles, nous avons privilégié les fonctionnalités interactives au détriment de l'esthétique pure. Ce projet nous a permis de développer nos compétences techniques et de démontrer notre capacité à collaborer efficacement, créant ainsi une expérience utilisateur riche et engageante.