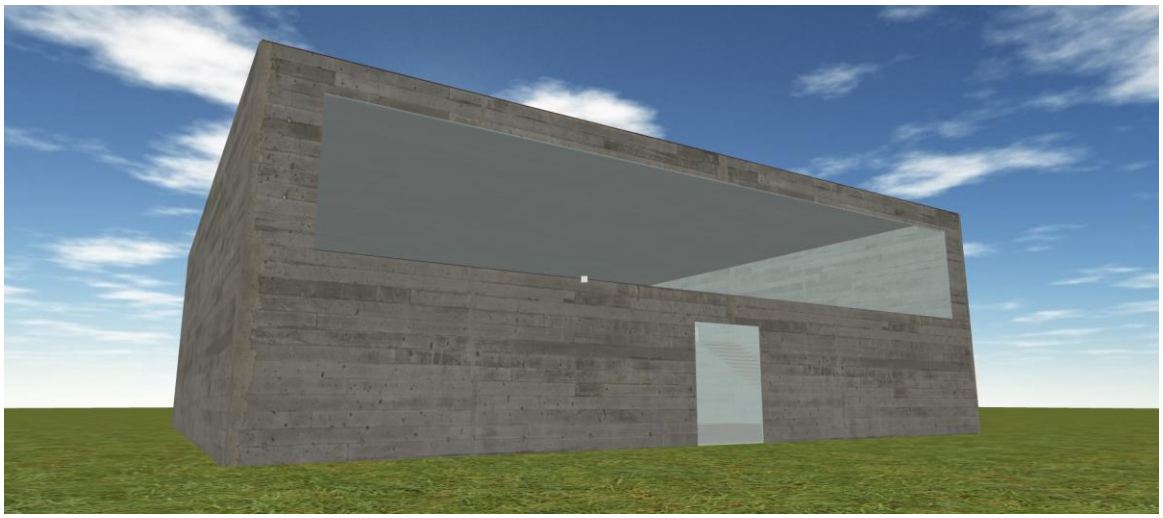


Musée Virtuel

PRESENTATION, FUNCTIONNEMENT ET REMARQUES

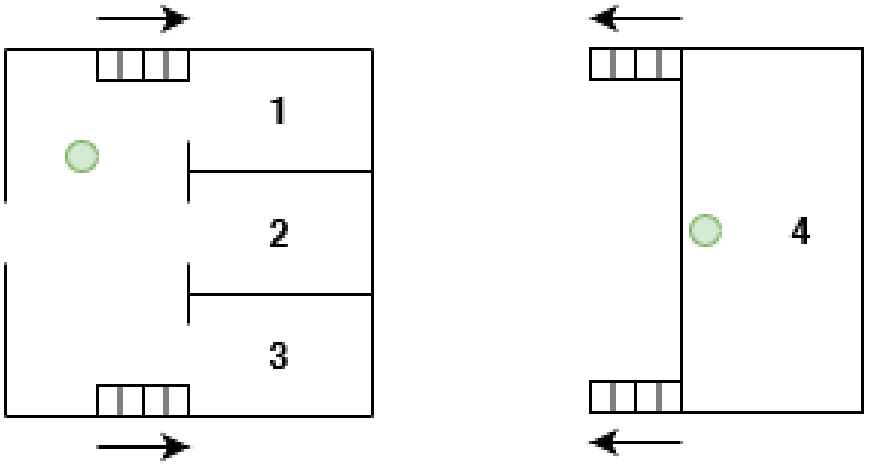
Hernio | REV | 17/06/22

Table des matières

Carte des lieux

RDC

1ER Étage



Légende :

- 1 : exposition Niklas Mäckle
- 2 : exposition SublimePoulpe
- 3 : exposition Kristoffer Zetterstrand
- 4 : chien géant en terre cuite qui tourne
- ○ : Telesphere™
- ▭ : Escalier

Les expositions

NIKLAS MÄCKLE

Niklas Mäckle est un artiste de 21 ans s'exprimant au travers de scènes réalisées en 3D dans le style voxel, comme si l'univers décrit était régit par une immense grille tridimensionnelle qui pouvait être rempli de cube unicolore.

Il s'est notamment fait connaitre au travers du jeu « Teardown » dont il réalise des environnements aux atmosphères uniques.

SUBLIMEPOULPE

C'est moi ! Il y a un an de ça, j'ai commencé à apprendre les rudiments de la modélisation, du rigging et de l'animation 3D.

Je créé des scènes inspirées des premiers jeu 3D de la Nintendo 64 ou de la PS1.

KRISTOFFER ZETTERSTRAND

Kristoffer Zetterstand est un artiste surréaliste suédois dont les œuvres mélangent art de la Renaissance et le début des arts numérique.

On peut parfois voir dans ses œuvres des collages, des personnages en reliefs peint, des formes 3D primitives et autres personnages en « pixel art » sur des plans en 2D.

Fonctionnement des salles du RDC

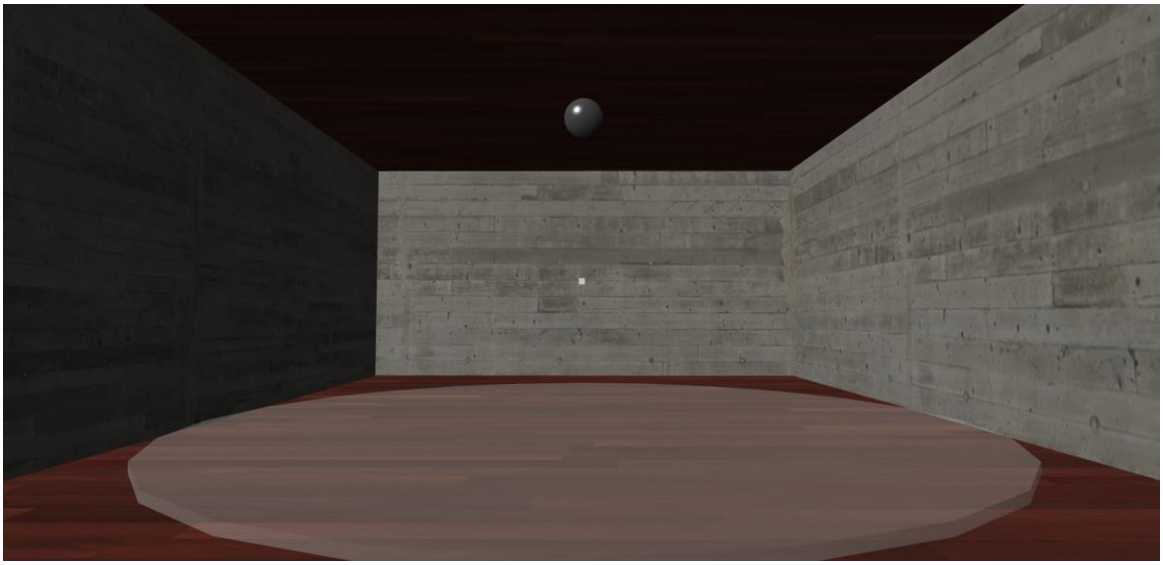


Figure 1 : A première vue, la salle semble bien vide…

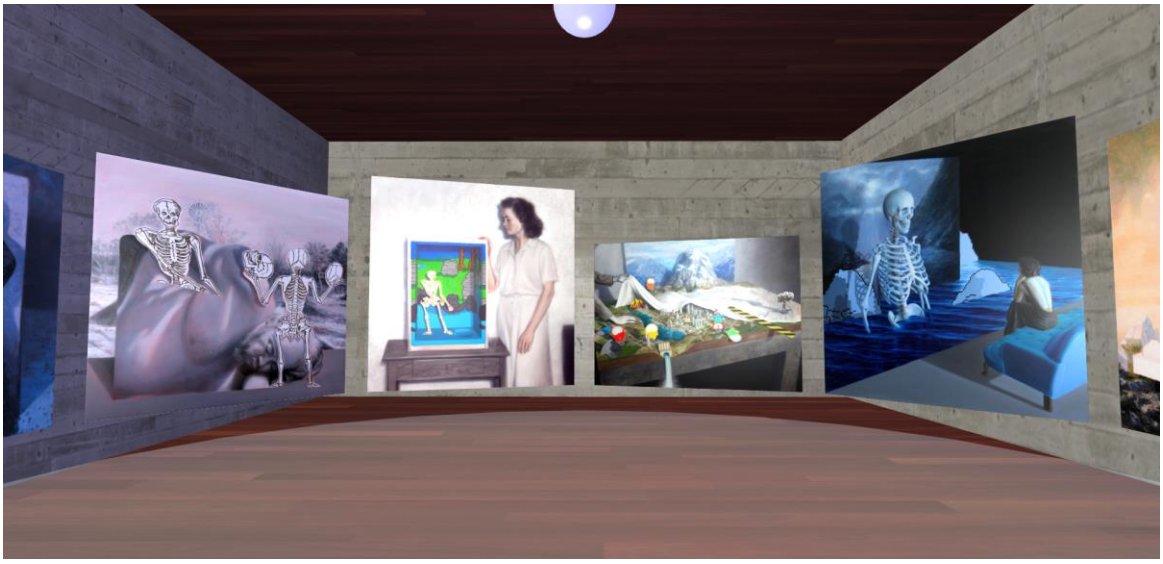


Figure 2 : Mais une fois à l'intérieur, tout apparaît !

LE « MAGNIFIQUE » CODE DERRIERE CE COMPORTEMENT

```
    zone.actionManager.registerAction(
        new BABYLON.CombineAction(
            {
                trigger:
BABYLON.ActionManager.OnIntersectionEnterTrigger,
                parameter: {
                    mesh: playerCol,
                }
            },
            [
                new BABYLON.InterpolateValueAction(
                    BABYLON.ActionManager.NothingTrigger,
                    paintingAnchor,
                    'position.y',
                    paintingAnchor.position.y + distanceInGround,
                    500
                ),
                    new BABYLON.PlaySoundAction(
                    BABYLON.ActionManager.NothingTrigger,
                    clic
                )
            ]
        )
    );

    zone.actionManager.registerAction(
        new BABYLON.CombineAction(
            {
                trigger:
BABYLON.ActionManager.OnIntersectionExitTrigger,
                parameter: {
                    mesh: playerCol,
                }
            },
            [
                new BABYLON.InterpolateValueAction(
                    BABYLON.ActionManager.NothingTrigger,
                    paintingAnchor,
                    'position.y',
                    paintingAnchor.position.y,
                    500
                ),
                    new BABYLON.PlaySoundAction(
                    BABYLON.ActionManager.NothingTrigger,
                    clac
                )
            ]
        )
    );
```

Figure 3 : La définition de l'enfer

Ce morceau de code sert simplement à faire : « Quand un collider entre dans la zone, la position des tableaux change et un son est déclenché » et « Quand un collider quitte la zone, la position des tableaux se déplace dans l'autre sens et un autre son est joué ».

C'est en pas moins de **46 lignes** de code horrible à relire que ce comportement est défini. Pour rappel, implémenter ce type de mouvement est quelque chose de très fréquent pour toute expérience interactive.

Pourquoi est-il nécessaire de définir une « CombineAction » au lieu de simplement ajouter plusieurs actions à un trigger ? Pourquoi est-il nécessaire d'indiquer un « NothingTrigger » pour une action combinée ? Pourquoi avoir un type d'action par comportement (« PlaySoundAction », « InterpolateValueAction »), au lieu d'un bout de code générique avec un accès en référence vers l'objet ayant le trigger ? Chaque type

d’action devant être paramétrée correctement et ne pouvant faire varier qu’un seul paramètre à la fois. Pourquoi dois-tu fonctionner aussi bizarrement, Ô BabylonJS ?!

Je ne suis pas du genre à cracher sur un outil. Je pense plutôt qu’un outil a un contexte dans lequel il brille vraiment.

Mais pour le cas de BabylonJS… Peut-être que si on se trouve sur une machine très peu puissante et qu’on aime se faire du mal, c’est l’outil idéal.

Aujourd’hui nous avons la chance d’avoir accès à des programmes fantastiques comme Unity, voir même Godot si jamais on aime être à contrecourant, qui permettent de générer des expériences en 3D pour le web. Et qui on la décence de proposer une interface pour voir ce que l’on fait, au lieu de devoir deviner des coordonnées, de retaper du code, de rafraichir la page web, de se rendre compte que les coordonnées ne sont pas bonnes, de retaper du code, rincer et répéter.

Et je ne parle même pas de l’intégration d’objets provenant de l’extérieur et de la logique douteuse sur la gestion des armatures…

Sinon on peut voir des informations sur le tableau quand on passe sa souris dessus, c’est sympa. (Un collider est constamment en face de la caméra et est détecté par les tableaux qui affiche un plan avec du texte).

Telesphere™

Afin de faciliter le déplacement dans le musée, deux Telesphere™, sont présentent.

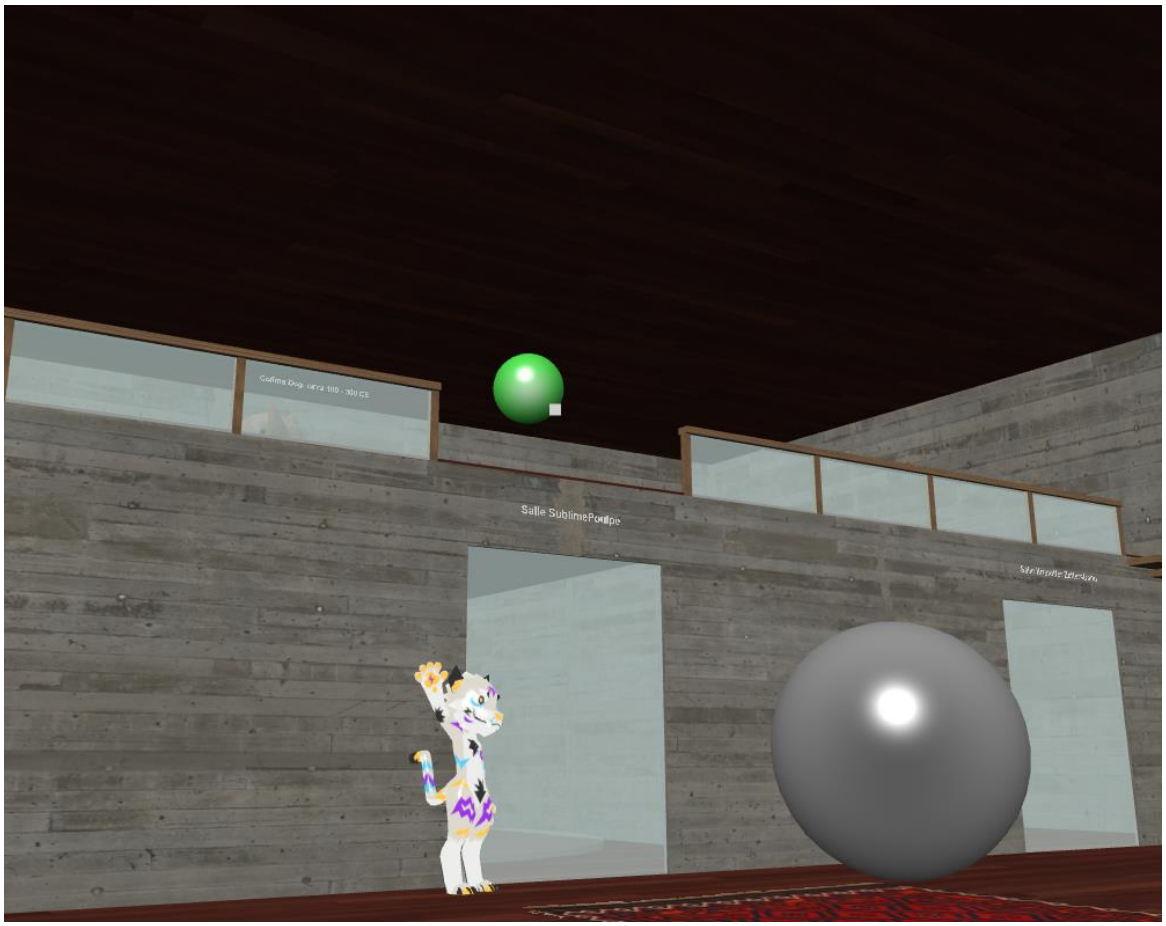


Figure 4 : Deux Telesphere™, représentant la puissance des technologies Franco-Bretonnes

L'utilisateur n'a qu'à placer son curseur sur l'une d'entre-elle, celle-ci va changer de couleur, indiquant à l'utilisateur qu'elle est sélectionnée et, au simple clic de souris, il sera transporté tout en douceur vers sa position. Pas mal non ? C'est Franco-Breton.

```
new BABYLON.ExecuteCodeAction(
                    BABYLON.ActionManager.NothingTrigger,
                    function () {
                        selectedTeleportationSphere = new
BABYLON.Vector3(groupe.position.x, groupe.position.y,
groupe.position.z)
                    }
                )
```

Lorsqu'une Telesphere™ est survolée, on assigne une valeur globale (berk) avec les coordonnées de celle-ci. (On assigne une valeur vide quand l'utilisateur arrête de survoler une Telesphere™.)

```
window.addEventListener ("click", function(){
    if(selectedTeleportationSphere != null && teleportationProgress >=
1.0) {

        get_clic().play();
        teleportationProgress = 0.0
        startPosition = camera.position
        endPosition = selectedTeleportationSphere
    }
})
```

A chaque clic, on vient vérifier que cette valeur globale contient bien une position et si l'utilisateur n'est pas en train d'être téléporté.

Si c'est le cas, on joue un son, on remet à o la progression de la téléportation et on met à jour les paramètre de déplacement : la position de départ et d'arrivé.

```
    engine.runRenderLoop( function(){

        if(teleportationProgress < 1.0) {
            teleportationProgress += 0.02

            camera.position = BABYLON.Vector3.Lerp(startPosition,
endPosition, smoothStep(teleportationProgress))

            if (teleportationProgress >= 1.0) {
                kids.play();
            }
        }

        scene.render();
    } ) ;
```

Dans la boucle de mise à jour du programme, si une téléportation est en cours on vient incrémenter le progrès.

On met à jour la position de la camera en faisant une interpolation linéaire entre la position de départ et celle d'arrivé avec la valeur du progrès. On ajoute une petit smoothStep pour adoucir la transition (on est pas des animaux, c'est bien plus agréable).

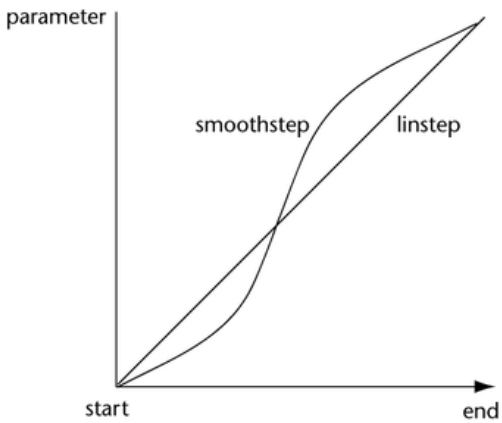


Figure 5 : Comparaison entre l'interpolation linaire et smoothStep

Import de modèle externes

Figure 6 : Un canidé tournant sur lui-même

Figure 7 : Statue animée

L'immense chien provient de SketchFab et est partagé sous licence CCo, après simplification du modèle avec Blender, il a été importé dans la scène.

La statue animé est une création personnelle et est animé, saluant avec sympathie l'utilisateur.

Bilan

L'expérience fut intéressante, malgré l'outil peu agréable à utiliser. J'ai presque envie d'écrire mon propre éditeur par-dessus BabylonJS. La plupart des choses présentées lors du cours m'étais déjà connu au travers d'Unity et de Godot et il était vraiment frustrant de voir la verbosité de BabylonJS face à la simplicité des autres outils. J'ai surtout eu l'impression de taper bien plus de code pour faire bien moins.

Pour finir sur une note plus sympathique, j'ai pu partager deux artistes que j'aime beaucoup et qui sont des sources d'inspiration pour moi.