

TER: Interface de réalité virtuelle pour manipuler des pièces archéologiques

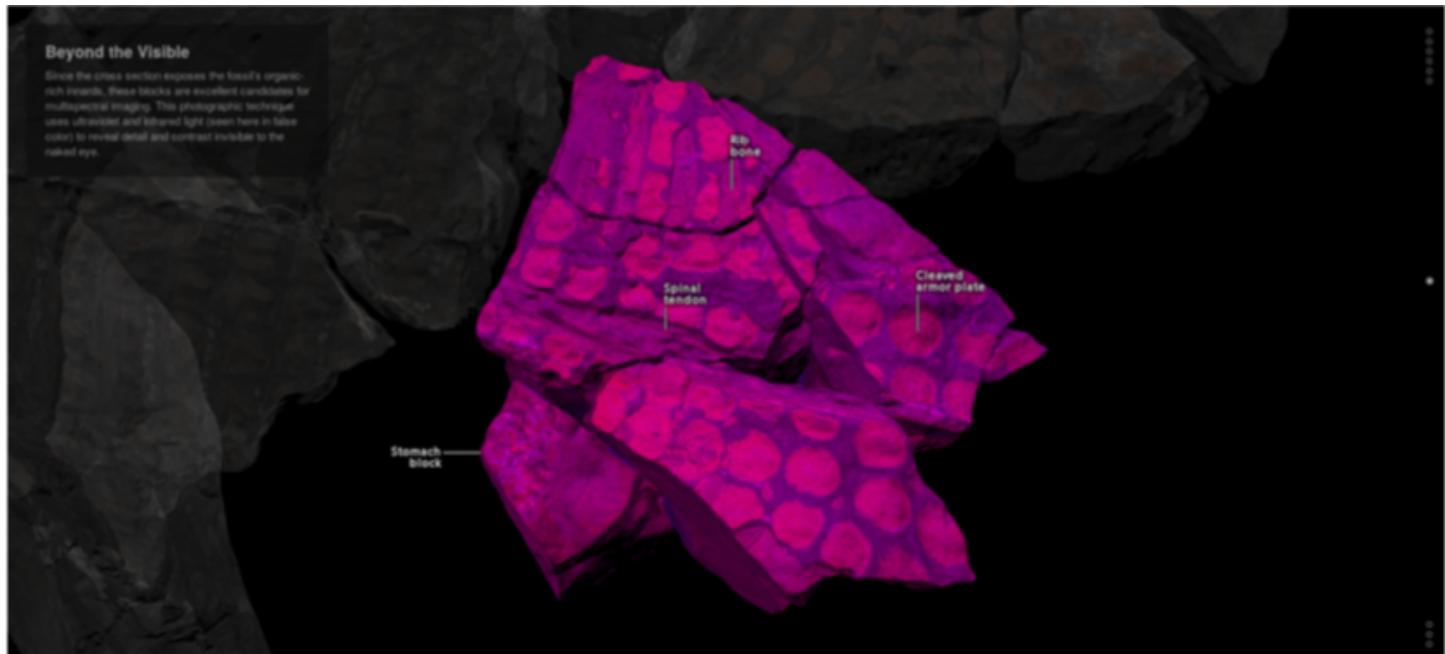
ALLIO David¹ BARRY Jean-Noël¹ DUARTE Florian¹

¹Aix-Marseille

Université

Résumé

Ce projet vous propose une visite virtuelle de musée en appliquant de la réalité virtuelle ainsi que augmentée. Ces technologies nous permettent ainsi de se focaliser sur des points d'intérêts d'objets archéologiques et de les manipuler afin d'accéder à diverses informations sur eux (images, vidéos, ou reconstruction d'une partie brisée par exemple). Les salles de musée ainsi que le placement des pièces 3d et des points d'intérêts sur lesdites pièces sont d'ailleurs générés procéduralement, à partir de fichiers de configuration éditables par tous.

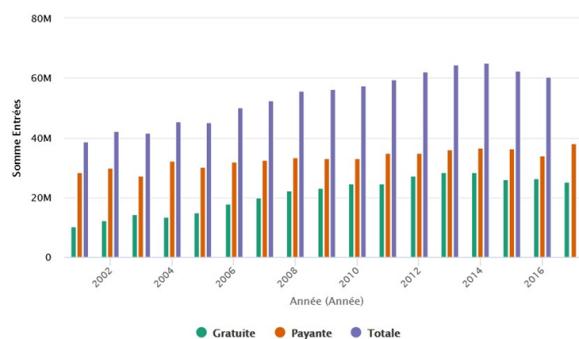


Résumé	2
1 Présentation du projet	3
1.1 Introduction	3
1.2 Etude de l'existant . .	3
1.3 Problématiques	5
1.4 Pistes explorées	5
2 Architecture et Conception	6
2.1 Architecture	6
2.2 Cahier des charges . . .	6
2.2.1 Exigences fonctionnelles	6
2.2.2 Exigences non-fonctionnelles	6
2.3 Répartition des charges .	6
2.4 Travaux effectués . . .	7
2.4.1 Analyse fonctionnelle	7
2.4.2 Analyse non-fonctionnelle	15
2.5 Problèmes rencontrés .	15
3 Application & Tests	16
3.1 Application	16
3.1.1 Analyse	16
3.1.2 Tutoriel pour ajouter un QRCode au projet	17
3.1.3 Critique	19
3.2 Tests	19
4 Conclusion	19

1. Présentation du projet

1.1. Introduction

Les avancées technologiques récentes nous ont poussé à la dématérialisation dans de plus en plus de domaines. La culture, entre autres n'a pas été épargnée par ce phénomène. Par exemple, la démocratisation des plateformes d'écoute et de visionnage en streaming ont sérieusement impacté le marché des supports physiques.



Fréquentation des Musées de France selon le site du ministère de la culture. ([Lien vers les dons](#))

Cependant, malgré l'explosion de la dématérialisation de la culture, l'intérêt des activités culturelles en présentiel n'a pas diminué : concerts, cinéma, expositions, musées, théâtre... la culture étant avant tout un partage entre le public et l'artiste.

Afin de s'adapter à cette évolution, les musées ont d'abord travaillé sur leur identité virtuelle. C'est-à-dire, leur présence au format numérique que ce soit sous forme de site ou de comptes sur les réseaux sociaux.



Le site internet du Louvre en 2001

En 2014, **75 %** des musées français ont une identité virtuelle mais seulement **5 à 10 %** d'entre eux propose des formes d'interactions numériques (bornes interactives, application, audioguides, réalité augmentée...) [\[1.21\]](#)

En 2017, 22 % des musées français mettent l'intégralité de leur collection en ligne. [\[1.22\]](#). Dans ce contexte d'adaptation au numérique, il est important pour un musée de s'interroger sur l'intégration de solutions dématérialisés en son sein.

Afin d'assurer, aussi bien aux néophytes technologiques qu'aux initiés une expérience de visite profitable, il semble optimal que ce brassage entre dématérialisation et musée soit fait sur une technologie connue de tous et maîtrisé par le plus grand nombre : le smartphone.

En partant de ces observations, le projet TER de visite de musée en réalité augmentée (désigné par AR par la suite) proposé par Romain Raffin & Marc Daniel du laboratoire LIS nous a donc convaincu.

Le contexte général de ce projet est la visite virtuelle de musée, en se focalisant sur les objets archéologiques et leur manipulation, en utilisant la Réalité Augmentée. Les pièces sont issues de numérisation par photogrammétrie. On possède donc des images photographiques des pièces, les maillages surfaciques(triangulaires) de reconstruction, ainsi que la description de zone d'intérêt en 3D(ex : une tête, un bras, une main ...).

Après discussions, Il nous semblait alors intéressant d'essayer de pousser l'expérience de visite plus loin via la possibilité de visiter virtuellement le musée avec un ordinateur. Chaque pièce du musée serait reconstitué avec les œuvres et on pourrait interagir avec chaque œuvre de la même manière qu'avec la réalité augmentée.

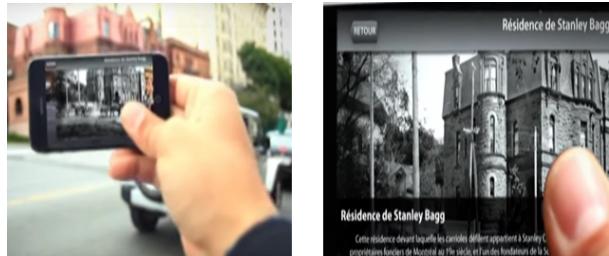
1.2. Etude de l'existant

Il existe quelques projets similaires dans le domaine de la réalité augmentée dans les musées. Parmi lesquels on peut citer :

- [\[1.23\]](#) L'application Sukiennice New Dimension : Lorsque l'on vise un tableau, l'application incruste en temps réel une vidéo mettant en scène le/les personnage(s) de ce tableau sur un des côtés du cadre. (Ci-dessous un aperçu du rendu)



- [1.24] L'application « Montréal – Points de vue » : L'application propose une carte avec différents lieux d'intérêt. Lorsqu'on se rend sur l'un des lieux, et que l'on cible avec l'appareil la zone demandée, une photo du même lieu dans le passé est affichée avec un texte explicatif.



- L'application AR Gallery Explorer : Application proposant une chasse au trésor dans une exposition. Quand une œuvre est visée, un texte et des images sont proposés pour l'expliquer. Parfois un indice en 3D apparaît en réalité augmentée pour faire progresser l'énigme.



- Tango, de Google. Un smartphone équipé de capteurs détectant le mouvement, la distance, la profondeur et qui est capable de reconnaître l'environnement pour réagi face à certaines œuvres avec des animations 3D en réalité augmentée. Mais son développement a été abandonné au profit du kit de développement pour Android : ARCore.[\[1.25\]](#)



Il existe d'autres exemples, dont il n'y a pas d'illustration disponible comme :

- La Conciergerie à Paris qui met à disposition "histopad", une tablette permettant d'observer une reconstitution 3D des anciennes cuisines médiévales en pleine préparation d'un banquet sur 360 degrés. (mais là il s'agit de réalité virtuelle et non pas de réalité augmentée)

- Le château de Versailles qui propose une application qui permet d'afficher des images, des sons en 2D ou 3D se superposant en temps réel sur ce que le visiteur peut voir autour de lui.

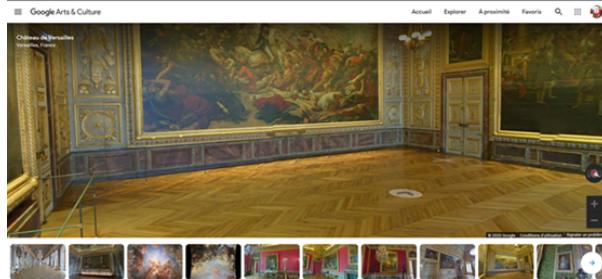
- L'application Overly adaptée pour le musée national d'art de Latvian.[\[1.26\]](#)



Cette application est celle qui ressemble le plus à notre projet. L'utilisateur cible un tableau avec l'application, il apparaît alors sur le tableau des points d'intérêts avec lesquels peut interagir l'utilisateur. Il peut alors obtenir une description précise de la zone et un détail de la peinture agrandi (ou tout autre image/texte associé à l'œuvre).

L'application se veut multi-usage. Elle permet par exemple d'avoir des zones où l'utilisateur peut dévoiler des détails cachés en grattant l'écran, faire apparaître des objets 3D devant un tableau, afficher une vidéo à la place d'un portrait fixe...

Au niveau de la réalité virtuelle, les musées les plus célèbres proposent une visite via Google StreetView, ce qui permet effectivement de visualiser les pièces des musées vues de différents angles, mais de façons très limitées.



Quant à la génération procédurale de salles, aucun autre projet unity à notre connaissance ne réalise ce que nous avons voulu créer.

Tango de Google et Overly sont les 2 seules applications à proposer une structure adaptable à n'importe quel musée. Les autres applications sont développées pour des musées/expositions spécifiques et leur adaptation à d'autres musées nécessiterait beaucoup de modification dans leur code source.

Cependant Tango et Overly ne permettent pas de générer nativement la génération d'un musée pour à la fois une visite en réalité virtuelle (VR) et en réalité augmenté (AR). De plus Overly ne permet de définir que des points d'interactions sur un plan 2D avec des marqueurs 2D, il ne gère pas les interactions avec différentes zones 3D.

[1]]

<https://www.latribune.fr/blogs/le-blog-sur-le-marche-de-l-art/20140313trib000819757/comment-le-numerique-est-en-train-de-metamorphoser-le-monde-de-l-art.html>

[2]]

<https://www.archimag.com/archives-patrimoine/2017/07/10/musees-heure-numerique>

[3]]

<https://ecs-digital.com/culture/realite-augmentee-musee-exposition/>

[4]]

<https://www.lejdd.fr/JDD-Paris/Revolution-numerique-dans-les-musees-841735>

[5]]

<https://www.lesnumeriques.com/mobilite/realite-augmentee-google-abandonne-tango-profit-arcore-n69521.html>

[6]]

<https://overlyapp.com/case-study/museum-offers-augmented-reality-exploration-for-engaging-storytelling/>

1.3. Problématiques

Elles se scindent naturellement en 2 parties : La réalité virtuelle (VR), sur pc, et la réalité augmentée (RA ou AR) sur téléphone.

Contraintes de la VR : générer un musée, décrire l'architecture d'un musée, décrire les œuvres et leurs caractéristiques, gérer une caméra en première personne pour simuler une vraie visite.

Contrainte de la RA : Apprendre à utiliser Vuforia, gérer les interactions avec l'utilisateur, décrire les œuvres et leurs caractéristiques.,

Contraintes des 2 : Que chaque pièce puisse être sélectionnée à différents endroits, afin d'avoir des infos sur cette partie précise, ou alors des reconstructions de morceaux brisés..

1.4. Pistes explorées

Pour la gestion des données, il nous faut une façon simple pour que le musée puisse rajouter des modèles , des points d'intérêts, et des documents (texte, image, vidéo) et leur associer un marqueur (généré ou non) pour la RA.

On peut soit créer un dossier qui contient toutes les infos et les fichiers, structuré avec une hiérarchie spécifique (le rajout d'une pièce de musée se fera par drag & drop du dossier dans les assets de unity), soit avoir une IHM qui permettra de gérer cela automatiquement.

Il faudra également gérer les interactions avec l'AR (quel ensemble d'objets charger pour quel qr code).

Dans tous les cas, l'utilisateur n'aura pas à changer le code.

L'indexation peut se faire avec les fichiers .xyz. Ils représentent une partie d'un objet auquel on peut rajouter un sémantique (main, bras, jambe...)

Quand à la génération procédurale de salles, elle pourrait entièrement s'effectuer par le biais d'un fichier de config (xml, ou .txt), qui contiendra toutes les infos sur les salles , et sur les oeuvres (noms, nombre, placement dans l'espace, etc...).

2. Architecture et Conception

2.1. Architecture



Image agrandie

2.2. Cahier des charges

2.2.1. Exigences fonctionnelles

RA : Le modèle doit apparaître en 3d face à l'utilisateur quand il scanne le qr code correspondant

Optimisation : Les modèles 3d étant nombreux, et quelquefois imposants, la génération de salles de musée et d'infos sur les modèles doivent être les plus efficaces possibles, ou l'on risque des chutes de performance.

Multiplicité : Les informations proposées pour chaque oeuvre devront pouvoir être issues de différents médias (image, vidéo, son).

2.2.2. Exigences non-fonctionnelles

Ergonomie efficace : Les menus doivent être clairs et concis pour que l'utilisateur s'y retrouve facilement

Charte Graphique : Cette application est dédié à tous, elle doit donc être assez "belle" ou l'on risque un désintérêt général à son encontre.

2.3. Répartition des charges

Jean-Noël Barry a géré la réalité augmentée en apprenant à utiliser vuforia

Jean-Noël Barry et Florian Duarte ont réalisés la sélection plus précise des pièces à différents endroits, afin d'avoir des points-clés sur cette partie précise. Ainsi que la gestion du XML.

David Allio a effectué la génération 3D d'un musée, ainsi que la gestion de la caméra

Notons aussi que nous avons employé une méthode de développement agile, via des vidéoconférences tenues chaque semaine avec nos encadrants, Marc Daniel et Romain Raffin.

2.4. Travaux effectués

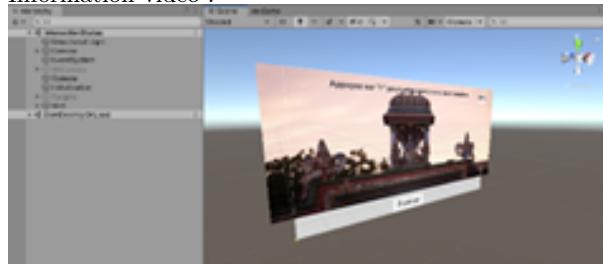
2.4.1. Analyse fonctionnelle

Nous avons tout d'abord cherché à nous familiariser avec Unity en mettant au point les fonctionnalités les plus simples.

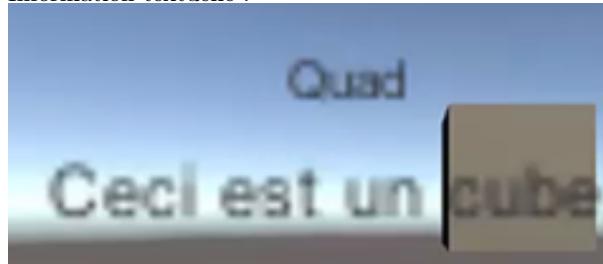
Début de mise en place des interactions

Nos premiers objectifs ont été de faire apparaître les informations souhaitées lors d'une interaction avec des cubes.

- Information vidéo :



- Information textuelle :



Ici, les éléments liés aux informations sont ajouté manuellement dans la scène. Bien entendu, pour simplifier l'utilisation de l'application, nous avons travaillé par la suite à les ajouter automatiquement via un fichier de configuration. Cela permet à l'utilisateur de n'avoir des connaissances que sur le fichier de configuration.(fichier .xml)

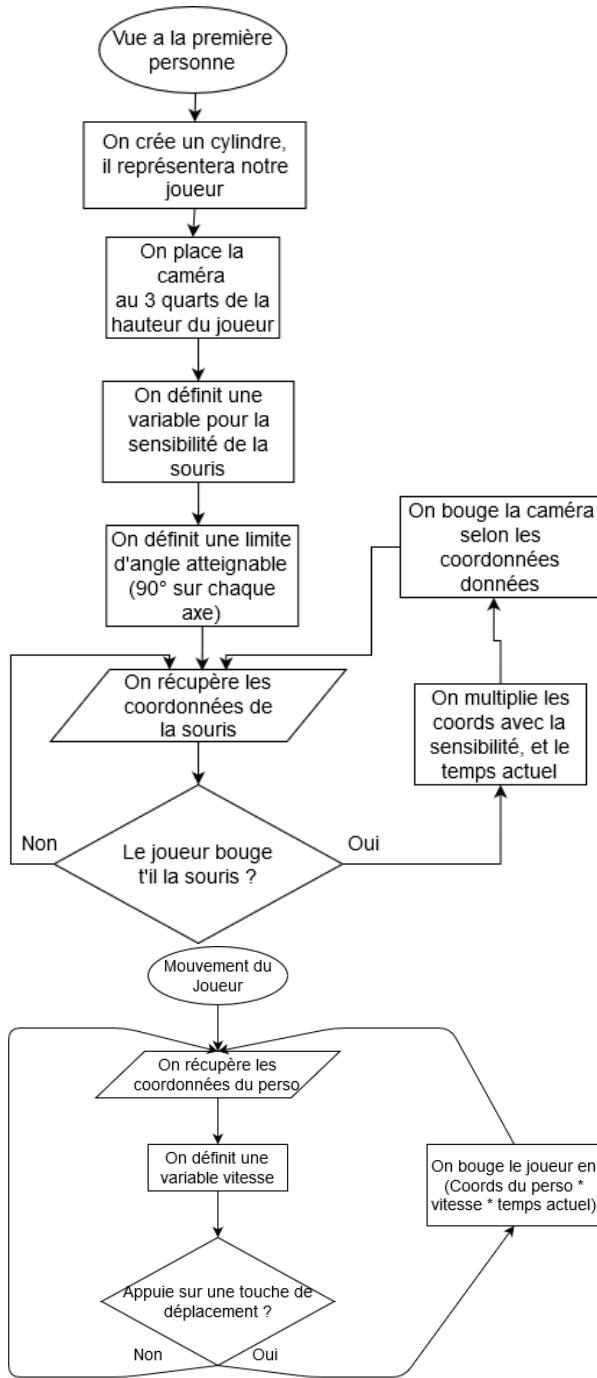
Début de l'environnement 3d

Nous avons essayés de créer un menu principal nous permettant de rejoindre un environnement 3d basique (un simple sol constitué de successions de cube de dimension 1x1) où l'utilisateur pourrait se balader librement (<https://streamable.com/q7ev9f>).

Ce plancher étant bien vide, nous nous sommes alors penchés sur le chargement de pièces de musée en 3d. Unity n'intégrant pas nativement l'obtention dynamique de ressources, j'ai dû me baser sur un plugin libre de droit "ObjLoader" , que j'ai alors légèrement modifié pour l'incorporer au chargement initial de ma scène 3d.

Nous avons alors ajouté à cet environnement l'utilisateur, sous forme d'une vue à la première personne pouvant se déplacer librement dans l'espace.

Algorithmes gérant les mouvements du joueur, et sa vision à la 1ère personne :



Configurer manuellement la génération du musée

La génération du musée était faites au début de façon statique dans le code : on créait une instance de chaque objet à la main. Par la suite, nous avons réfléchi à la création d'un fichier de configuration pour automatiser tout cela.

La génération finale du musée repose sur un fichier xml. Ce fichier décrit entre autre :

- Les pièces du musée, leur emplacement et leur dimensions
- Les œuvres du musée, leur emplacement, leur dimension et leur orientation
- La correspondance entre les œuvres, les zones d'interactions et les ressources à afficher

Voici un exemple de fichier de configuration :

```

<?xml version="1.0"?>
<musee xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xi="http://www.w3.org/2001/XMLSchema-instance" nom="siva museum">
    <salle x="50" y="0" z="75" largeur="100" longueur="150" hauteur="30">
        <portes>true,true,false,true</portes>
        <oeuvre nom="siva" obj="/siva/siva.obj" qrc="sivaTarget" x="50.55" y="2.88" z="30.75" rotx="0" roty="0" rotz="0" taille="0.5">
            <masque nom="mask_048_3" path="/siva/mask_048_3">
                <resource type="img">
                    <value>/siva/la-sculpture-du-champa-tresors-d-art-du-vietnam-ve-xve-siecles.jpg</value>
                </resource>
            </masque>
            <masque nom="mask_048_4" path="/siva/mask_048_4">
                <resource type="img">
                    <value>/siva/manuscritsShiva.mp4</value>
                </resource>
            </masque>
            <masque nom="mask_048_5" path="/siva/mask_048_5">
                <resource type="txt">
                    <value>La tradition shivaïte de l'hindouisme est centrée sur Shiva, considéré dans cinq grandes fonctions : il est le créateur</value>
                </resource>
            </masque>
        </oeuvre>
        <salle x="165" y="0" z="75" largeur="130" longueur="50" hauteur="30">
            <portes>false,true,true,false</portes>
            <oeuvre nom="myson" obj="/myson/myson.obj" qrc="mysonTarget" x="80.72" y="5" z="100.23" rotx="0" roty="150" rotz="0" taille="3">
                <masque nom="masks3D_Myson_00" path="/myson/masks3D_Myson_00">
                    <resource type="img">
                        <value>/myson/DSC_062.JPG</value>
                    </resource>
                </masque>
                <masque nom="masks3D_Myson_02" path="/myson/masks3D_Myson_02">
                    <resource type="txt">
                        <value>Parmi les antiquités en exposition au musée de sculpture Cham de Da Nang, vous serez ravis de contempler trois œuvres p</value>
                    </resource>
                </masque>
            </oeuvre>
        </salle>
    </musee>

```

Lien cliquable pour voir l'image agrandie.

Contexte : Y désigne l'axe vertical, un objet qui pointe en direction de Y pointe vers le ciel/plafond. (0,0,0) est l'origine de toutes les coordonnées. Les coordonnées en X,Y,Z et les dimensions largeur/hauteur/profondeur sont exprimés en mètres. Les angles sont exprimés en degrés. Les chemins sont définies de façon relative par rapport à l'emplacement du XML.

Nous allons le détailler balise par balise :

- **musee** désigne l'ensemble du musée, l'attribut **nom** permet de donner un nom au musée. Ce nom est affiché sur l'écran d'accueil.
- **salle** désigne une salle du musée. Les attributs **x**, **y** et **z** permettent de lui donner une coordonnée par rapport à l'origine. **Largeur**, **longueur** et **hauteur** désignent les attributs respectifs de

la salle.

- **porte** désigne les portes qui font la transition avec les autres salles. Il est composé de 4 booléen séparés par des virgules. Chaque booléen désigne la présence d'un mur sur les 4 côtés de la pièce (`true`=un mur,`false`=une porte)
- **oeuvre** désigne une oeuvre. L'attribut `nom` permet de lui donner un nom afin de pouvoir facilement la retrouver quand il y a beaucoup d'oeuvres dans le xml. `obj` permet de lui associer le modèles 3D qui porte ce nom (précisez le chemin et l'extension du fichier 3D). `qrc` permet de lui associer le qrcode du même nom dans Unity. `x`, `y`, `z` permettent de lui donner une coordonnée dans la salle. `rotx`, `rotY`, `rotZ` permettent de lui donner une rotation par rapport à sa position importée (axe vertical y, l'objet pointe dans l'axe z). `taille` permet de définir sa taille par rapport à celle du modèle importé (taille inf 1 = modèle plus petit, taille sup 1 = modèles plus grands).

- **masque** permet de d'associer un masque à une oeuvre et les interaction qu'il déclenche. `nom` est le nom du fichier de masque qu'il faut utiliser. `ressource` est le nom du fichier de ressource qu'il faut ouvrir en cas d'interaction (il faut mettre l'extension sauf quand il s'agit de texte). `type` est le type de la ressource à ouvrir (img=image, vid=video, txt=texte, mus=musique).

Résumé :

- `<musee>` est la balise qui englobe l'intégralité du musée
- `<salle>` décrit une salle et les œuvres qui la composent
- `<porte>` décrit la présence ou non de portes dans la salle
- `<oeuvre>` désigne une œuvre
- `<masque>` désigne les zones d'intérêts d'une œuvre

Le fichier XML est déserialisé au démarrage de l'application. Nous utilisons pour cela des méthodes incluses dans le langage de base (System.Xml). Pour extraire les données du XML de manière pratique, il nous suffit alors de créer la même hiérarchie que le XML en ayant des classes à la place des balises.

Par exemple, la structure :

```
<musee nom="">
<salle></salle>
<salle></salle>
<salle></salle>
...
</musee>
```

est représentée par :

```
[XmlRoot("musee")]
6 références
public class Musee
{
    [XmlAttribute("nom")]
    public string nom;

    [XmlElement("salle")]
    public List<Salle> salles = new List<Salle>();
```

(Et ainsi de suite pour les balises fils).

- `/XmlRoot("musee")` pour dire que **musee** est la balise englobante qui sera à la racine de la hiérarchie
- `/XmlAttribute("nom")` pour dire qu'un attribut de la balise **musee** est le nom de celui-ci, on peut préciser différents attributs de type différents.
- `/XmlElement("salle")` pour dire que la balise **musee** peut avoir un ensemble de balises **salle** comme fils.

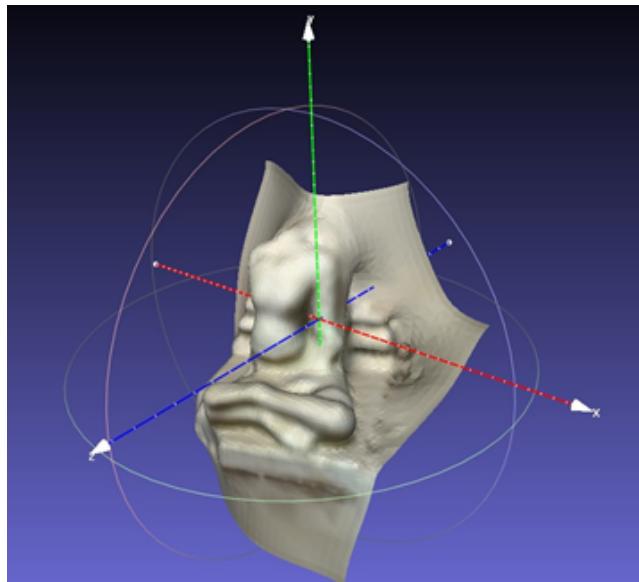
Avantage de cette méthode : une fois que le fichier xml est déserialisé dans une instance de **musee**, il est très simple d'accéder à tous les éléments du musée. Pour obtenir le nom du musée par exemple, il suffit de faire **musee.nom**.

Inconvénient : La structure du XML doit être toujours la même, peu de tolérance aux erreurs de syntaxe, laborieux à mettre en place quand il y a beaucoup d'attributs et de balises différentes.

Normes qui seront admises dans la suite de ce rapport

La réalité augmentée avec Vuforia nécessite de définir une orientation aux objets et aux masques qui leurs sont associés. Afin de faire correspondre l'orientation des objets avec celle des ImageTarget qui leur sont associés, il est nécessaire d'en imposer une.

L'axe Y (vert) est l'axe vertical, l'axe Z (bleu) pointe en direction de la caméra.



Utiliser vuforia pour associer des qr codes à des modèles

Vuforia n'est pas intégré de base à Unity. Il existe cependant un plugin qui permet de l'utiliser facilement dans unity.

Vuforia utilise **2 modules** pour mettre en place un système de réalité augmentée.

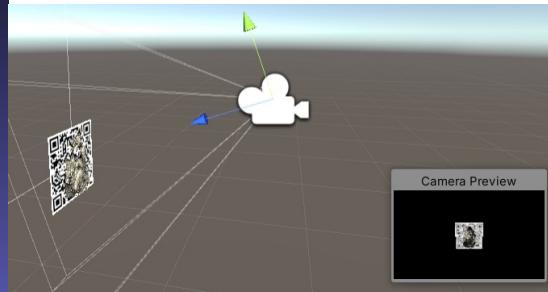


Figure 1: ARCamera



Figure 2: ImageTarget

D'une part une **ARCamera** (capture du haut) qui représente la caméra de l'appareil sur laquelle sera lancée l'application. C'est elle qui affiche un retour vidéo et détecte les images cibles.

Et d'autre part les **ImagesTarget** (capture du bas) qui correspondent aux motifs à détecter.

On associe aux ImagesTarget des fils qui peuvent être des modèles 3D (comme sur la capture), de la musique, des vidéos... Quand Vuforia détecte un motif, il va afficher et activer tous les fils qui lui sont associés.

Note : Dans Unity, un objet fils est contrôlé par son objet père : si le père est déplacé la même transformation est appliquée au fils, si le père est

rendu invisible (désactivé) alors le fils sera invisible également.

Un ImageTarget possède un motif. C'est le nom que l'on donne à l'image qui doit être détectée. Cette image peut être une texture, un qrcode, une photographie d'un objet... L'important est que cette image doit avoir à la fois une certaine complexité et une certaine lisibilité. Nous nous limiterons aux QrCode dans ce projet, mais les premiers tests fait avec des textures étaient très concluant.

Les motifs associés aux ImageTarget peuvent être définis de 2 façon :

Target Name	Type	Rating	Status	Date Modified
qrcode_shiva	Single Image	★★★★★	Active	May 06, 2020 18:45
qrcode_myson	Single Image	★★★★★	Active	May 06, 2020 18:15

- en passant par le site de vuforia qui propose d'uploader les motifs, d'évaluer leur pertinence et d'exporter facilement l'ensemble des motifs dans unity (**From DataBase**)
- en passant par une image définie par une texture (**From Image**)

Nous avons utilisé la méthode **From DataBase** pour créer 1 ImageTarget par oeuvre.

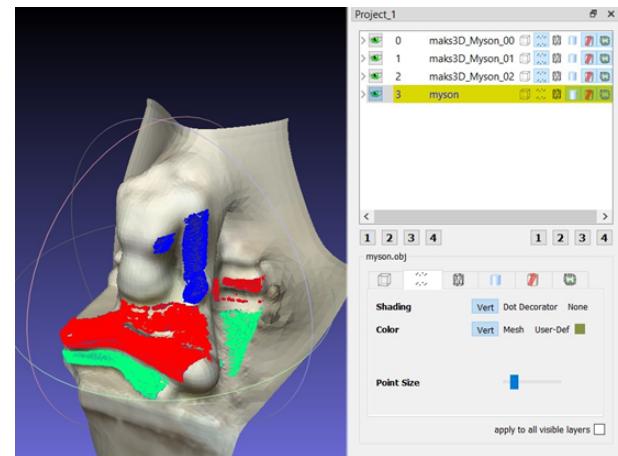
Etant donné que nous n'avons pas eu le temps d'implémenter un éditeur, il est malheureusement nécessaire, pour ajouter de nouveaux qrcode à l'application de faire quelques manipulations dans Unity. Le fichier XML ne fait que décrire à quel qrcode existant ajouter une oeuvre.

Chargement des masques sur unity pour en faire des enveloppe 3D qui correspondent à des parties de l'œuvre.

```
-2.9267500 3.2192200 -12.8112000
-2.9848500 3.2061000 -13.0354000
-2.8954200 3.2596400 -12.7362000
-2.9059700 3.2213200 -12.7608000
-2.9273800 3.2437800 -12.8830000
```

Les masques sont des fichiers textes (.xyz) contenant des coordonnées disposées une par ligne.

Ces coordonnées sont relatives au centre du modèle 3D (son centre de gravité) et non pas à l'origine de la scène.



L'ensemble de ces coordonnées, une fois reliées, forment une zone qui doit pouvoir être sélectionnable par la suite.

Afin de pouvoir transformer ces ensembles de points en objets pouvant être interactifs, il est nécessaire de générer un mesh à partir de ces points. Pour avoir une zone de détection étendue pour faciliter les interactions, nous avons généré des enveloppes convexes de ces ensemble de points.

Ceci a été réalisé grâce à ce package unity :

<https://github.com/Scrawk/Hull-Delaunay-Voronoi>

Il prend en entrée un ensemble de points et renvoie un mesh. Pour de meilleurs résultats, l'ensemble des points en entrée doivent être mélangés. Ce que nous avons fait une fois que les points ont été lus dans le fichier.

Une fois que les meshes sont générés, on leur associe :

- un MeshCollider : pour les interactions avec l'utilisateur
- un MeshRenderer : pour pouvoir personnaliser les zones d'interactions
- un MeshFilter : nécessaire pour utiliser un MeshRender
- un Outline : un plugin de unity permettant de mettre en surbrillance la partie. Lien : <https://assetstore.unity.com/packages/tools/particles-effects/quick-outline-115488>

Un Gameobject va finalement regrouper cet ensemble d'éléments. Il sera nommé par le nom du masque afin de faciliter l'association de future ressources.

Tous ces objets sont des structures existantes dans Unity que l'on instancie avec les propriétés que l'on désire. Soient récupérés à partir du xml, soient définies arbitrairement. Voir 3 pour le pseudo-code de cette partie.

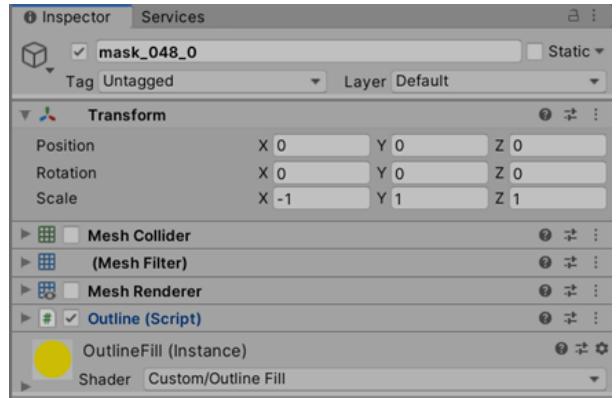


Figure 4: Tous les composants générés par cette méthode pour un masque

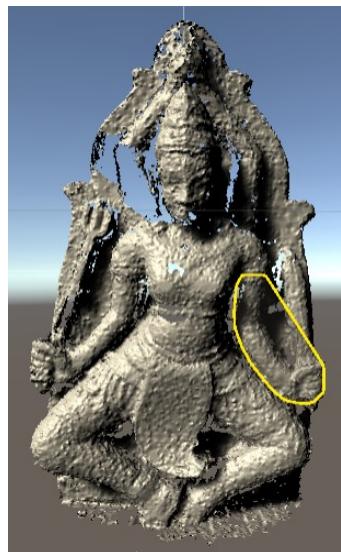


Figure 5: Le masque qui a les composants ci-dessus (zone en jaune)

```

1 Pour chaque œuvres
2   On charge le modèle 3D
3   On crée une instance de GameObject qui le contient
4   Pour chaque masques définis pour le modèle 3D actuel
5     On charge l'ensemble des coordonnées
6     On génère une enveloppe convexe avec
7     On ajoute un MeshCollider, un MeshFilter, un MeshRenderer, un module Outline
8     On regroupe le tout dans un GameObject
9     On rajoute ce GameObject au GameObject qui contient le modèle 3D
10    On modifie l'échelle du GameObject avec celle précisée dans le XML
11    On Associe une ImageTarget au GameObject

```

Figure 3: Pseudo-code de cette partie

Association de ressources avec les masques (manuellement dans un 1er temps). Gestion de l'affichage des ressources (afficher une vidéo, une image, du texte...)

Une fois que les zones d'interactions sont définies, il faut les associer avec des ressources.

Les ressources sont les éléments qui seront affichées lors de la sélection d'une zone (définie précédemment, pour rappel, par un fichier masque). Une ressource peut être un texte, une musique, une vidéo ou une image.

Pour cela, on récupère la structure chargée à partir du fichier xml. Quand une zone est sélectionnée, on récupère son nom (qui a été défini précédemment) et on recherche dans la structure, la ressource qui lui est associée.

```

<masque nom="mask_048_1" path="/siva/mask_048_1">
  <ressource>
    <type>img</type>
    <value>/siva/DN0048.JPG</value>
  </ressource>
</masque>

```

(Par exemple ici, si on sélectionne l'objet qui s'appelle **mask_048_1**, on va ouvrir le fichier au chemin relatif **/siva/DN0048.JPG** qui est de type **image**)

La ressource est alors chargée (sauf pour le texte) et affichée selon son format.

Génération des salles en vr :

Pour une compréhension plus aisée de la suite, il faut noter que unity utilise les axes X et Z pour la longueur et la largeur d'un sol, Y représentant lui l'axe montant jusqu'au ciel

La création de salles devant être la plus simple possible, nous avons pris la décision de la générer à partir de coordonnées basiques. Le fichier de configuration initial était ainsi séparé en 2, la première partie concernant les salles, et la seconde, les œuvres.

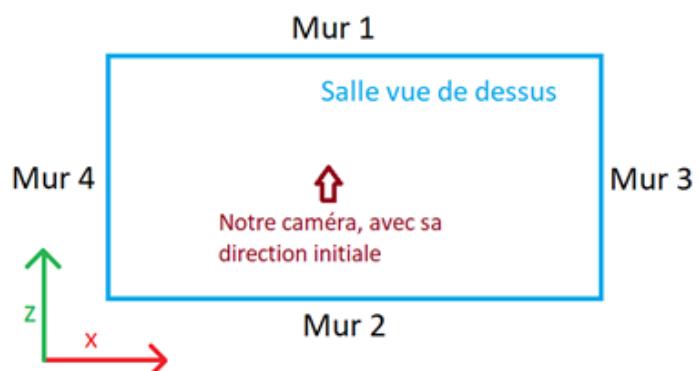
Les salles sont déplacé au coordonnées X,Y,Z suivantes voulues par l'utilisateur

Ces 3 coordonnées reviennent pour définir la rotation de la pièce par rapport à elle-même.

Un seul élément manquait alors, comment relier ces salles. Nous avons alors implémenté un système permettant de "trouver" les murs en leurs milieu, si 2 salles ont un trou sur un mur ou ils étaient autrefois collés, passer à travers nous permet de changer de salle.

Elle se traduit dans le fichier de config final par une suite de 4 booléens, true (mur plein), ou false (mur troué)

Ordre des murs :



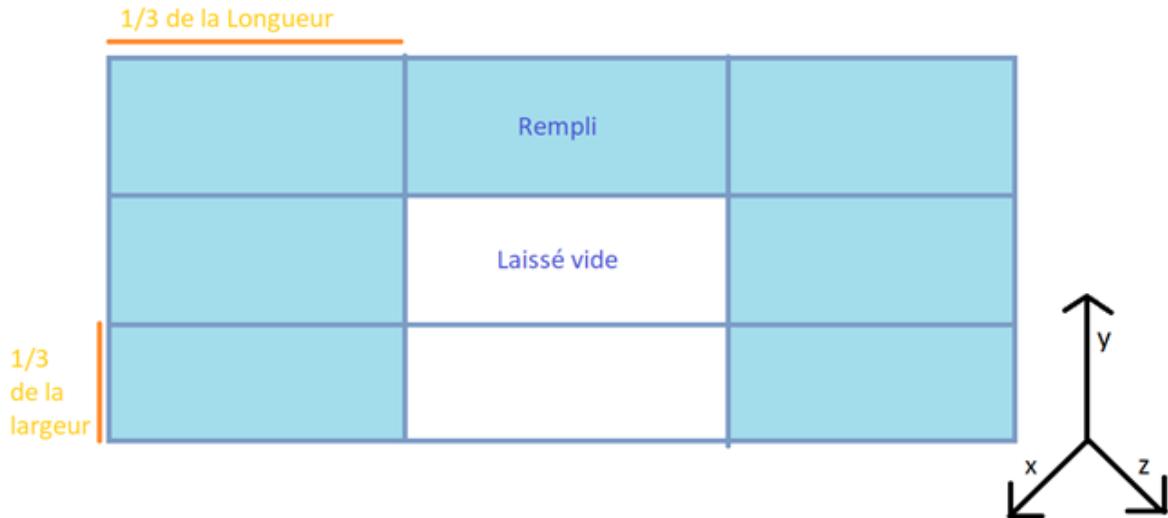


Figure 6: Diagramme expliquant la génération des murs avec ouverture

Cette partie est séparée de l'autre par ////

La seconde nous fournira alors le nom des modèles 3d à implémenter, ainsi que leurs placements dans l'espace.

```

100,150,50,0,75,0,0,0;true,true,false,true
130,50,165,0,75,0,0,0;false,true,true,false
80,110,175,0,155,0,0,0;true,false,true,true
////
kitten/(50,55;2,80;30,75)
gargoyle/(20,31;2,80;60,25)
bunnyLowPoly/(80,72;2,80;100,23)
hexagon/(50,00;2,80;100,23)

```

Figure 7: Exemple initial de fichier de config

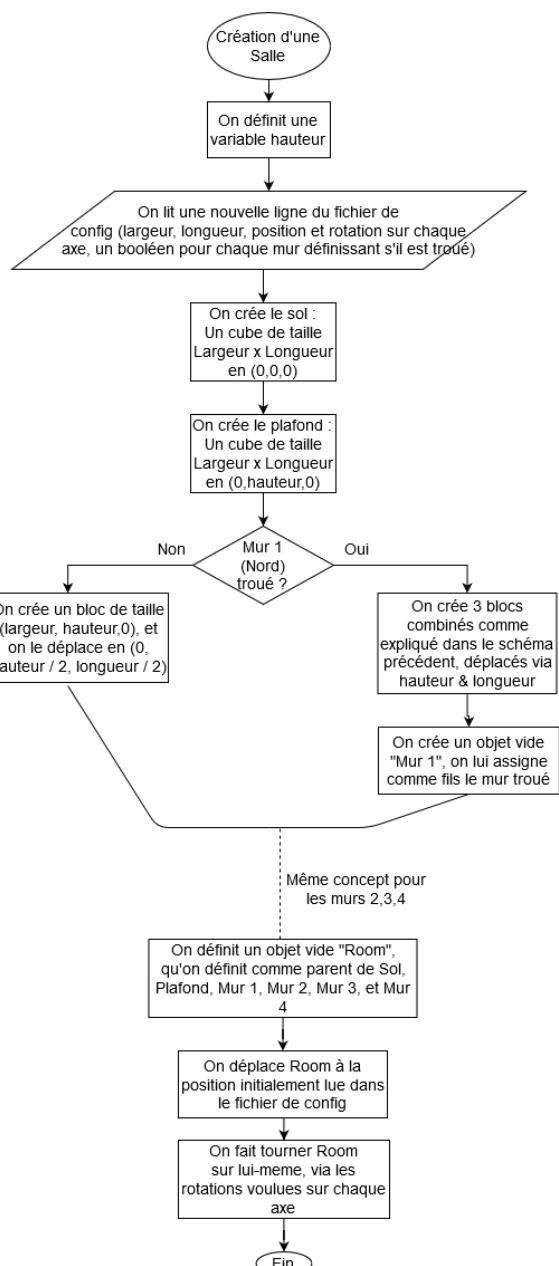


Figure 8: Algorithme gérant la génération de salle automatique via fichier de config

Pour finir, tous ces éléments nécessitent de la lumière pour les afficher proprement, nous utilisons alors conjointement 2 méthodes. La première a été de considérer les plafonds comme “transparent”, permettant à la lumière naturelle provenant du ciel d’unity d’illuminer le tout.

La seconde vient du constat que les oeuvres restaient légerement trop sombres, mais surtout qu’elles étaient difficilement repérables de loin, nous avons donc rajoutés procéduralement des spots colorés sous chaque oeuvre.

2.4.2. Analyse non-fonctionnelle

Les zones d’interactions sont explicitées avec des contours jaunes. Quand on se balade dans le musée, on voit le nom des œuvres que l’on vise avec le curseur.

Les planchers des salles sont d’une texture de bois différente de celle des murs et du plafond. Combinés au système de lumière évoqués précédemment, l’aspect visuel est assez plaisant.

2.5. Problèmes rencontrés

La visualisation des objets en réalité augmentée a nécessité beaucoup d’ajustement sur le positionnement des modèles, leur rotation et l’emplacement des zones d’interactions.

Pour charger les modèles 3d, Unity permet de charger des ressources dynamiquement avec Resources.Load, or ils faut qu’ils soient mis dans un dossier Ressource spécifique, compilé et ainsi inaccessible après compilation, il est donc inutilisable car impossible de charger des modèles externes.

Les “AssetBundle” fonctionnent aussi un peu comme un système de ressource, sauf qu’il n’y a pas de dossier de ressources obligatoire, malheureusement, c’est marqué comme obsolète dans la doc

On s’est donc tournés vers leurs successeurs, les “Addressable Asset”, mais étant plutôt récent et peu utilisé, il n’y avait que très peu de documentation

Notre solution finale utilise un plugin qui lit et reconstruit l’obj dans un format classique de unity (le gameobject) en c# brut. Hors cette solution

ne prenait pas en charge les textures, nous avons donc dû créer une fonction supplémentaire pour leur en assigner.

Pour la génération de salles, nous avons d’abord essayé de créer dans des boucles plusieurs blocs de dimension 1x1. Bien que techniquement fonctionnel, dès qu’une salle était un peu trop grande, le nombre de blocs devenait gargantuesque, entraînant une baisse notable de performances.

On aurait alors pu créer un modèle 3d de salle basique dans blender, puis le modifier dans unity, hors la demande de différentes dispositions de mur troués implique une génération dynamique, d’où le modèle finale, ou puisque unity ne peut creuser dans un élément, chaque mur plein est un cube aplati, et chaque mur troué est la combinaison de 3 cubes aplatis.

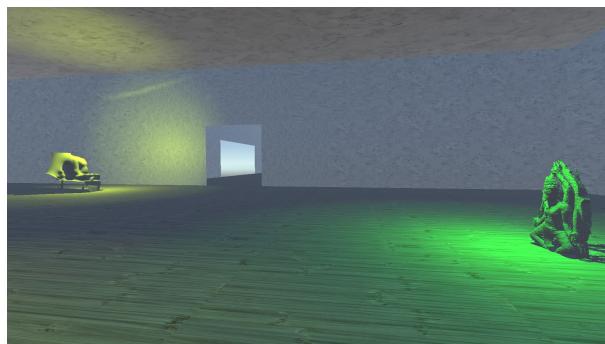
Aperçu final : <https://streamable.com/719nqn>

3. Application & Tests

3.1. Application

3.1.1. Analyse

Aperçu en réalité virtuelle :



Aperçu en réalité augmentée :



Pour voir les 2 en action :
<https://streamable.com/6qnh4e>

<https://streamable.com/di95dp>

Les fichiers pour configurer un musée :

Un musée est un fichier XML qui décrit le contenu du musée (voir pages 12 et 13 pour sa structure). Il doit être dans le même répertoire que les dossier qui contient les ressources.

myson	15/05/2020 20:33	Dossier de fichiers
siva	15/05/2020 20:33	Dossier de fichiers
musee.xml	26/05/2020 17:39	Fichier XML

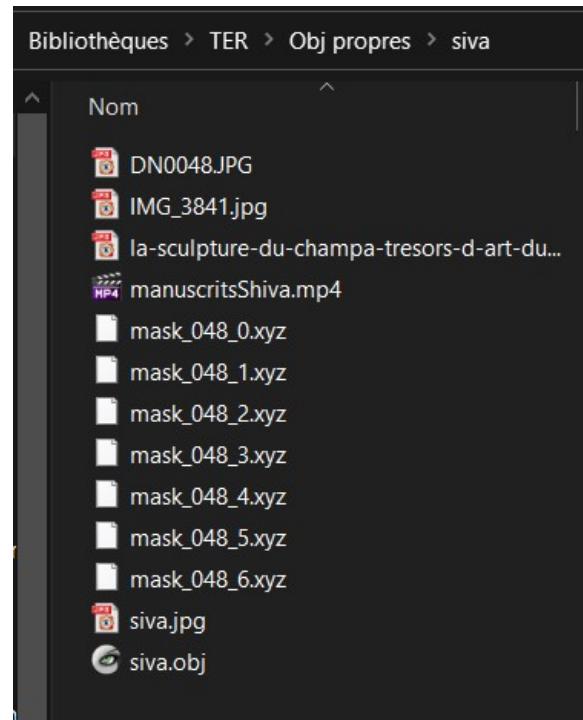


Figure 9: Chaque dossier contient les ressources associées à une oeuvre : son objet 3d, ses masques, les images/vidéos/musiques à afficher pour chaque partie.

Le musée se génère automatiquement en réalité virtuelle et pratiquement automatiquement en réalité augmentée. En réalité augmentée, il va falloir rajouter le qr code manuellement dans Unity. Tutoriel présent à la page suivante.

3.1.2. Tutoriel pour ajouter un QRCode au projet

1) Générer un nombre hexadécimal de 100 caractères

Aller sur : [Un site de génération de nombre hexadécimal](#)

Générez un nombre de 100 caractères. Copiez ce nombre.

2) Générer un qrCode

Aller sur : [Un at de qrcode](#)

Collez le nombre généré précédemment dans le champ “Free Text”

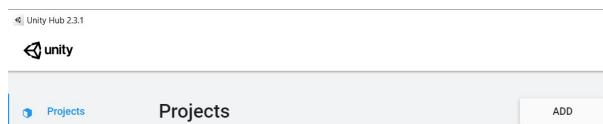
Le QRCode est généré automatiquement. Dans la partie “Size” , remplacez la valeur de “Custom” par 1500. Appuyez sur “Save” . Enregistrez le fichier obtenu avec le même nom que votre objet 3D (par exemple, siva.png pour siva.obj).



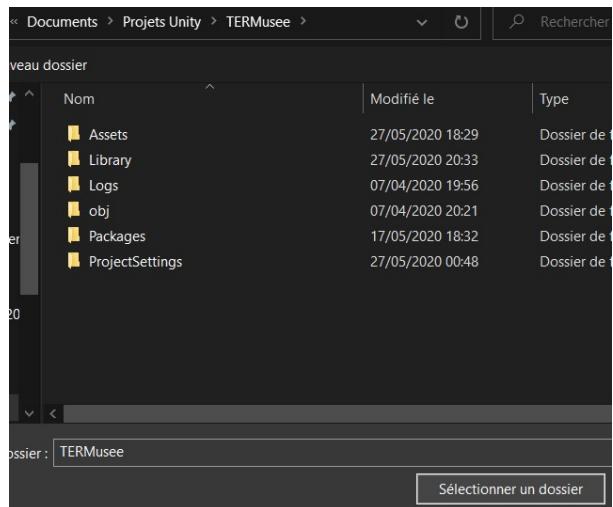
3) Ajouter un ImageTarget dans Unity

[Si Unity n'est pas déjà ouvert avec le projet :](#)

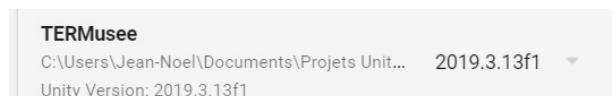
Ouvrez Unity Hub, cliquez sur le bouton Add à droite.



Allez dans le dossier du projet comme sur la capture. Cliquez sur “Sélectionner un dossier” .



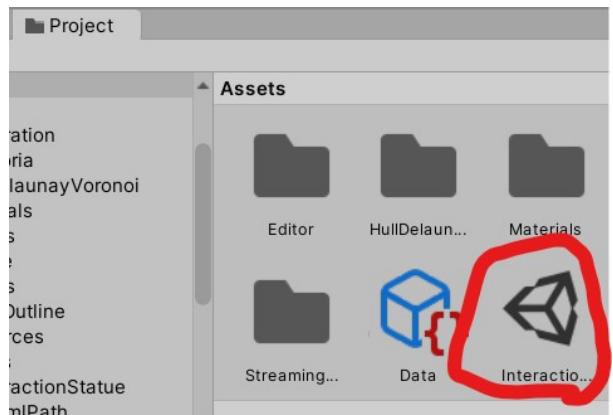
Cliquez sur le nom du projet qui vient d'apparaître dans Unity Hub.



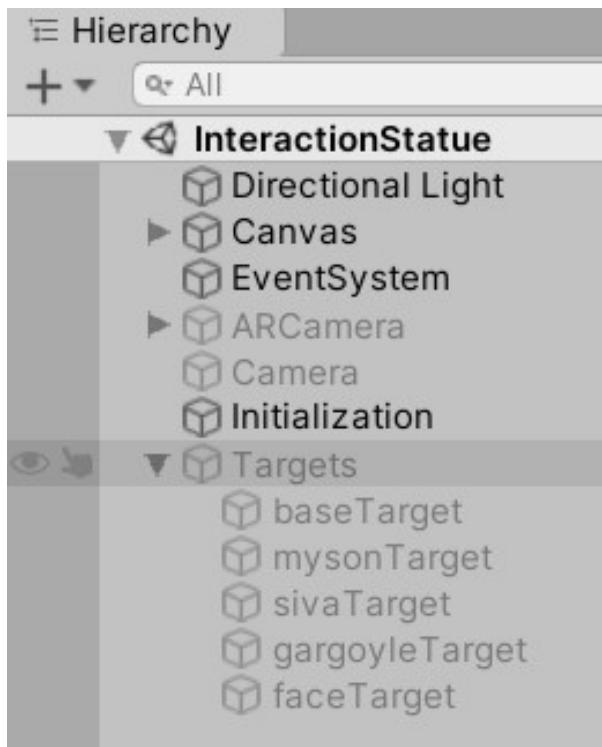
Continuez à l'étape suivante.

[Si Unity est déjà ouvert avec le projet :](#)

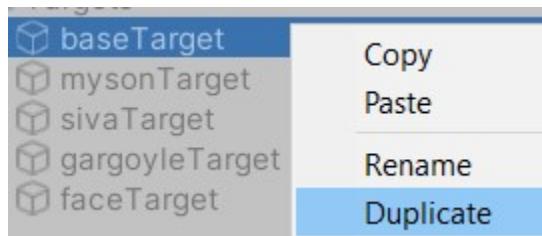
Repérez l'onglet “Project” . Double-Cliquez sur InteractionStatue.unity.



La scène s'ouvre, repérez l'onglet “Hierarchy” , déroulez l'objet “Targets” .



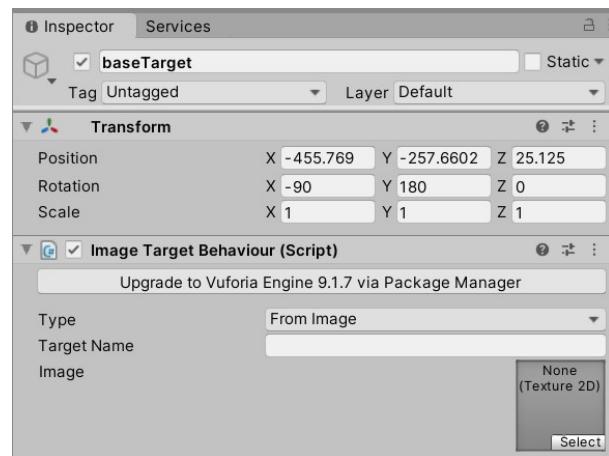
Faites un clic-droit sur “baseTarget” , cliquez sur “Duplicate” .



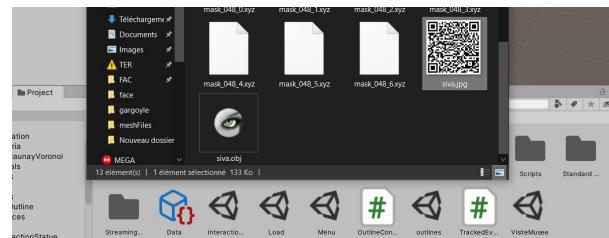
Cliquez sur l’objet “baseTarget (1)” créé. Repérez l’onglet “Inspector” . Renommez l’objet en NomObjet3DTarget. Par exemple sivaTarget pour siva.obj.



Repérez le script “Image Target Behaviour”



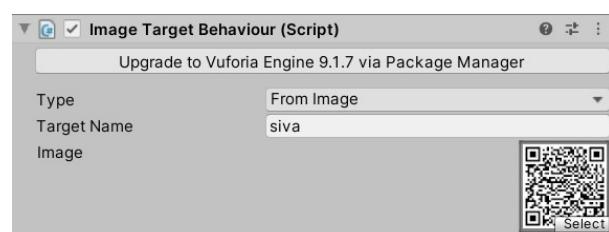
Laissez Unity ouvert. Placez votre explorateur de fichier par dessus. Retrouvez l’image du qrcode créé précédemment.



Glissez-déposez l’image dans la zone de l’onglet “Project” .



(Au besoin retournez dans “Hierarchy” pour re-sélectionner l’ImageTarget en cours de création)



La propriété Image du script est actuellement à

“None (Texture 2D)” .

Glissez-déposez l'image du qrCode des Assets jusqu'au carré gris à droite de la propriété “Image” .

C'est tout pour Unity. Dans le XML de configuration, n'oubliez pas de mettre le bon nom de l'objet target que vous venez de créer :

```
<oeuvre nom="siva" obj="/siva/siva.obj"
qrc="sivaTarget" ...>
```



3.1.3. Critique

Notre application est très légère, aussi bien niveau poids sur le disque dur qu'au niveau des performances requises pour le faire tourner, malheureusement, les musées eux sont plutôt lourds.

De plus, lorsque le nombre de pièces devient conséquent, les temps de chargements pour générer le musée deviennent plutôt conséquents.

Les salles pourraient aussi être plus belles, il n'y a actuellement pas la possibilité de mettre les œuvres sur différents supports (tables, piédestaux), ni d'éléments décoratifs (comme des fenêtres par exemple).

Enfin, le plus gros point noir est sûrement le manque d'éditeurs de niveaux, notamment à cause de la complexité actuelle des fichiers de configuration.

Cependant il nous semble important de noter que malgré ces défauts tout est fonctionnel. Nous réalisons, certes sans grand panache, tous les objectifs qui nous avaient été assignés.

3.2. Tests

Pour vérifier l'optimisation de notre création de salles en 3d, nous en avons générés 10 000 : <https://streamable.com/faugwi> Il n'y a pratiquement aucun impact niveau performances, c'est donc un test concluant.

4. Conclusion

Nous sommes assez content d'avoir pu rendre un projet fonctionnel dans les temps, cependant, il est vrai que de part notre amateurisme initial dans unity de nombreux points seraient améliorables.

Nous souhaitons implémenter un éditeur de niveau, dernier élément sur lequel on travaillait, mais dont nous n'avons pas réussi à créer une version finale à temps. De plus belles textures, ainsi que de plus beaux menus auraient aussi été appréciable. Et si l'on souhaitait aller encore plus loin, on pourrait même imaginer une plateforme pour télécharger en ligne et installer automatiquement différents musées fait par d'autres utilisateurs, améliorant ainsi la facilité d'utilisation, mais aussi, et surtout, retirerait la barrière actuelle du poids de l'application (plus il y a de modèles, plus elle est grosse, notamment de part le fait qu'il n'existe pas de technologie de compression dans unity), on pourrait ainsi aisément imaginer en chercher plus de 500 sans problème dans le futur.

Au final, ce projet a donc été une bonne introduction aux travaux 3d que l'on souhaiterait réaliser plus tard, et qui sait, peut-être sera-t-il ressorti une fois que nos compétences auront grandi dans ce domaine, nous permettant d'enfin concrétiser tous nos objectifs initiaux.