h e p i a
département I T I
ingénierie des technologies
de l'information

Hes·SO GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

SUIVI INTERACTIF AVEC L'HOLOLENS

(*mise-a-jour-hololens-ameliorations-660x330.jpg (660×330)* [sans date])

Mémoire de projet de semestre présenté par

Naores ALWAN

Ingénierie des technologies de l'information avec orientation en Logiciel

Mars 2019

Professeur-e HES responsable

Yassin REKIK

SOMMAIRE DÉTAILLÉ

REMERCIEMENTS

Nous remercions Yassin Rekik et Raed Abdennadher qui nous ont beaucoup aidé durant ce projet.

RÉSUMÉ

La réalité augmentée est une technologie qui devient de plus en plus repandue. La réalité augmentée est l'ajout d'élément tel que des sons, des vidéos, des Object 3D ou 2D à notre réalité. Les domaines de la réalité augmentée affectent l'industrie, l'éducation, la médecine et les jeux vidéo. L'éducation s'intéresse à cette technologie car elle permet d'apprendre dans des situations réelles sans avoir besoin d'acheter des équipements exorbitants. Par exemple, des apprentis mécaniciens pourraient apprendre à démonter un moteur sans devoir le partager avec leur camarade. Le but de ce projet de semestre est d'implémenter un scénario interactif à l'aide du casque de réalité augmentée qui est L'HoloLens. Ce dispositif a été commercialisé par Microsoft en 2015. Le projet que nous avons développé consiste à découvrir les technologies Unity et celle de la réalité augmentée de Microsoft. Cela nous servira à développer une application permettant de générer des représentations 3D de données en temps-réel et d'augmenter la réalité par ces représentations contextualisées. Puis, l'application se connectera à un serveur REST pour récupérer des données au format JSON. Ces données seront traitées et seront automatiquement mises à jour sur l'histogramme. Cette application pourrait être utilisée dans le domaine médical afin d'afficher les données d'un patient en temps réel. Par exemple, il serait possible d'afficher son rythme cardiaque, ses résultats sanguins actuelles ou anciens. Bien évidemment, ce ne sont que des exemples, et il est possible d'appliquer cette technologie à n'importe quel scénario impliquant des graphiques. Les aspects essentiels du projet de semestre sont la transmission de données en temps réel et la réalité augmentée.

(*Illustration* [sans date])

Candidat :

ALWAN NAORES

Filière d'études : ITI

Professeur responsable :

REKIK YASSIN

En collaboration avec : Hepia

Travail soumis à une convention de stage en entreprise : non

Travail soumis à un contrat de confidentialité : non

TABLE DES ANNEXES

TABLE DES ILLUSTRATIONS

LISTE DES ACRONYMES

REST: Representational state transfer

POC: Proof of concept

INTRODUCTION

Ce projet s'inscrit dans la troisième année d'étude de la filière Ingénierie des Technologies de l'Information où chaque étudiant réalise un projet de semestre. Ce dernier a pour but de traiter un problème d'ingénieur avec une difficulté moyenne, permettant de montrer les connaissances professionnelles acquises par l'étudiant. Il permet aussi de développer des réponses techniques et construites provenant d'une approche méthodologique et scientifique. Dans le cadre de mon projet, l'objectif était de concevoir et prototyper une application en réalité augmentée. Cette dernière a pour but de générer des représentations 3D de données en temps-réel et d'augmenter la réalité par ces représentations contextualisées. Cette application pourrait être utile dans le domaine médical.

Trois raisons principales ont motivé notre choix sur un projet qui a une longévité de plus de quatre mois :

- Découverte d'Unity
- Découverte de la réalité augmentée
- Découverte de l'HoloLens

1. DESCRIPTIF DU PROJET

L'HoloLens est un casque de réalité augmentée. Il a été conçu par Microsoft et permet de développer des applications en réalité mixte. L'HoloLens permet de superposer des Object 3D dans le monde réel grâce au moteur de jeux Unity.

2. TRAVAIL DEMANDÉ

Le premier but est de découvrir le moteur de jeu Unity. Pour cela, divers tutoriels d'Unity sont étudiés afin d'acquérir des notions de base. A leur tour, ces notions sont utilisées pour comprendre les tutoriels concernant l'HoloLens. L'exploration de ces 2 types de tutoriels rende possible la conception d'une application. Cette dernière a pour fonction l'acquisition de donnée en temps réel et de les afficher dynamiquement sur le HoloLens par le biais d'Histogramme.

Donc, la première partie de ce travail consiste en un travail d'exploration et de renseignement sur le moteur d'Unity. A cet fin, le site du fournisseur (Unity Technologies) https://unity3d.com/fr/learn/tutorials/s/interactive-tutorials, est d'une grande aide pour trouver toutes les informations utiles à la réalisation de la première phase d'autoapprentissage.

La seconde phase de ce travail se concentre sur le fonctionnement de l'HoloLens. A nouveau, le site du fournisseur de l'HoloLens (Microsoft) https://docs.microsoft.com/en-us/windows/mixed-reality/holograms-101e, est une source d'information relativement complète.

Une fois que les documentations préalables ont été rassemblés, comprises et intégrées pour les deux premières phases, il est possible de consacrer le reste du temps à la troisième étape. Cette dernière consiste en la réalisation d'une application « proof of concept (POC) », qui démontre les différentes technologies acquises. L'application déployée sur l'HoloLens traitera les données récupérées et affichera les données mises à jour. Cette application pourra par la suite être utilisée dans le domaine médical pour afficher en temps réel les données d'un patient.

CHAPITRE 1 : ANALYSE

1. STRUCTURE DU PROJET

Dans cette partie, on va s'intéresser à la description de la structure de notre projet.

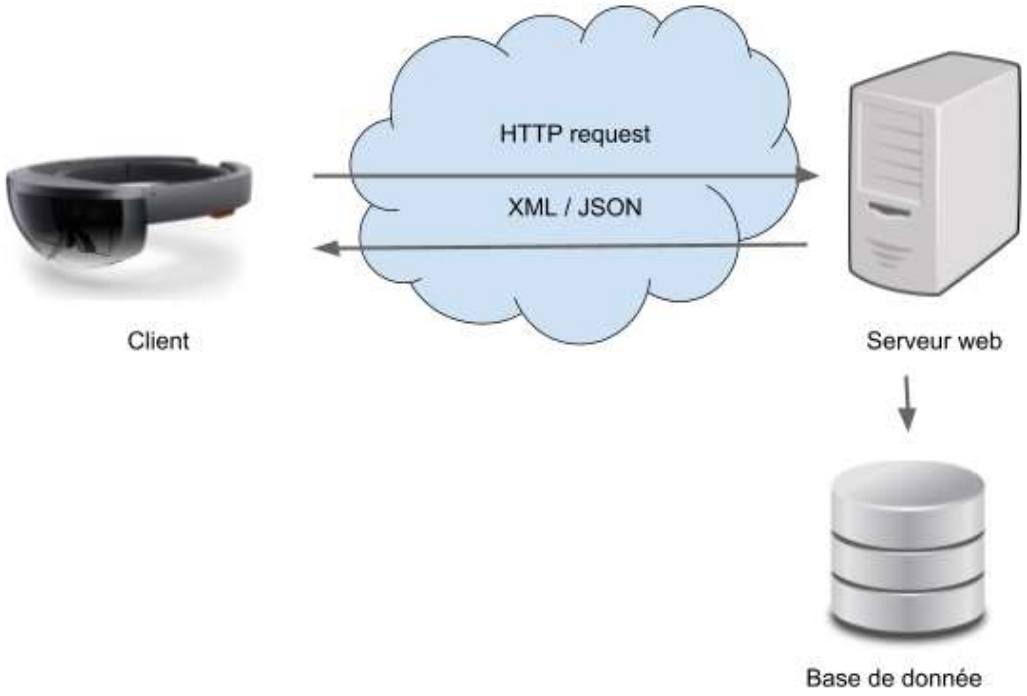

Figure 1 : Structure du projet

(*base-de-données.jpg (214×236)* [sans date]; *screenshot-2016-02-29-at-9-41-48-am.png (1000×563)* [sans date]; *server-567943_960_720.png (512×720)* [sans date])

Le fonctionnement de l'application POC est séparé en plusieurs modules différents. Le premier module est l'application déployée sur l'HoloLens. Elle a pour but d'afficher dynamiquement les histogrammes. Ainsi que de récupérer et de traiter les données envoyées par le serveur. Le second module est le serveur web. Il permet de mettre en relation l'application HoloLens et la base de données. Cette communication se fait par le biais d'un serveur express. Puis, il y a le troisième et dernier module qui est la base de données. Cette dernière est la technologie Rethinkdb. Elle nous permet de centraliser les données afin de les gérer plus facilement.

La première partie de ce projet a été de découvrir la réalité augmentée avec l'HoloLens. Ce chapitre va être séparé en trois parties distinctes. La première est une introduction à certaines notions du moteur Unity par rapport à la réalité augmentée. Puis, une introduction au différents GameObject de Microsoft pour la réalité augmentée. Et enfin, l'explication de Visual Studio.

2. UNITY

Unity est un moteur de jeux vidéo. Il permet de modéliser des objets en 3D ou en 2D. Cette notion est très importante pour la réalité augmentée. Cela sert à concevoir des objets que nous pourrons par la suite déployer sur l'HoloLens. Unity nécessite trois notions importantes qui sont les scènes, les GameObject et les composants.

2.1. Scènes

Les scènes sont l'une des parties les plus importantes d'Unity. Chaque scène correspond à un environnement où les objets sont créés et définis. Le projet est conçu en plusieurs scènes. Une scène équivaut à un scénario spécifique. Dans chaque scène, il y aura nos environnements, décors et obstacles. Cela sert à séparer notre projet en plusieurs parties distinctes.

2.2. GameObject

Un GameObject est une partie de la Scène. C'est à cet endroit que l'on ajoute les différents objets qui vont interagir avec le moteur de jeu. Unity propose une grande variété d'Object tel que les sphères, caméra, lights, et bien d'autres qui peuvent être utilisés en fonction des besoins. Par défaut, une nouvelle scène Unity contient déjà une lumière(lights) et une caméra.

La caméra est la vision de notre scène. Elle permet de voir la scène ou les scènes en question lorsque qu'on désire afficher ces dernières.

Les lights sont les effets de lumières de notre projet. Elles permettent de diriger les lumières d'une scène.

C'est à cet endroit que l'on définira les diffèrent Objects de notre scène. Les Object d'Unity sont basiques. Ces derniers sont des formes géométriques de bases (rectangles, triangles, cubes, sphères, cône, …). Toutefois, les Objects sont des images en 2D ou 3D pouvant être utilisées tel quel par Unity, mais nécessitent des composants pour pouvoir être fonctionnel

avec l'HoloLens. Par exemple, si l'on souhaite crée une table en 3D, il faudra décomposer la table en plusieurs formes de bases : un cube pour faire le haut de la table et ajouter des cylindres pour les pieds. Il y a énormément de GameObjects dans Unity qui peuvent être personnalisée grâce aux composants, expliqué au point suivant.

2.3. Composant

Un GameObject est paramétrable grâce au composant. C'est à cet endroit que nous pourrons définir sa position dans le plan, sa taille et sa rotation. Ces options sont ajustables grâce au composant « Transform ». « Box collider » permet de créer une zone d'interaction pour l'HoloLens. C'est grâce à ce composant que nous pouvons faire des actions sur l'Object. En effet, sans box collider l'HoloLens ne pourra pas sélectionner le GameObject. Nous avons aussi le « Mesh Renderer », ce composant permet d'afficher l'Object.

Le composant le plus important est le « script ». Ce dernier permet de déterminer le comportement de tous les objets, tel que leur déplacement, leur affichage, leur dissimulation et bien d'autres. Dans chaque script, la fonction peut être ajoutée, et elle est appelée au démarrage du script. Puis, il y a Update, cette fonction est appelée à chaque rafraichissement de notre scène. Ce rafraichissement est une valeur que nous définissons. C'est grâce à ce paramètre que nous pouvons déplacer les Objets ou le mettre à jour. Un GameObject n'a pas de restriction pour le nombre de composant qu'il peut posséder.

3. FONCTIONNEMENT HOLOLENS

Le casque de réalité augmenter HoloLens a été lancé par l'entreprise Microsoft. L'entreprise a mis à disposition une plateforme de tutoring pour pouvoir concevoir des applications en réalité augmentée. Les contraintes auxquelles doivent faire face les développeurs sont le système d'exploitation Windows 10, le moteur de jeux Unity, les libraires SDK de Windows et Microsoft Visual studio. Microsoft Visual studio permet de déployer les applications sur le dispositif et d'écrire les scripts.

3.1. Qu'est ce que HoloLens ?

L'HoloLens est un casque de réalité augmentée permettant de simuler des hologrammes. La réalité augmentée est la superposition d'objet 3D dit hologramme dans

l'environnement. La société Microsoft a conçu divers tutoriels permettant de se familiariser avec leurs différentes libraires.

3.2. Que peut-on développer avec l'HoloLens ?

Le développement d'application sur L'HoloLens se fait sur Unity. En effet, Unity a fourni de nombreux outils, packages et ressources permettant de simplifier la conception d'application et de jeux. (*Microsoft Mixed Reality* [sans date])

Le développement de script sur l'HoloLens se fait dans le langage C Sharp. C Sharp(c#) est un langage de programmation orienté Object. Il a été développé par Microsoft et est un dérivé du c++. De plus, la création d'application sur l'HoloLens se fait par Unity. En effet, Microsoft fourni une librairie permettant de simplifier la programmation sur le périphérique. Effectivement, Microsoft a ajouté différents GamObjects qui permettent d'avoir une reconnaissance vocale en anglais, la sélection des hologrammes, le Spatial Mapping et bien d'autres.

3.3. GameObjects de Microsoft

Les GameObjects de Microsoft sont rassemblés en une librairie simplifiant la conception des applications en réalité augmentée. Cette rubrique expliquera certaines notions importantes pour la compréhension du chapitre deux.

Gaze

Le Gaze constitue une forme principale de ciblage pour l'HoloLens. Il nous permet de savoir où l'utilisateur regarde et nous permet de déterminer son intention. Dans le monde réel, vous observerez généralement un objet avec lequel vous souhaitez interagir. C'est le même fonctionnement avec le gaze. **(thetuvix [sans date])**

Cursor

Le cursor est le pointer de l'HoloLens. Il a le même fonctionnement que la souris pour un ordinateur. C'est-à-dire qu'il nous permet de sélectionner les différents hologrammes afficher sur l'HoloLens. Un hologramme est un objet en 3D affiché sur HoloLens.

Spatial Mapping

Le Spatial Mapping ou cartographie spatiale en français est la représentation détaillée de l'environnement réel proche de l'HoloLens. Elle permet aux développeurs de créer une expérience de réalité mixte convaincante, fusionnant le monde réel avec le monde virtuel.

Les applications peuvent également s'aligner plus naturellement sur les attentes des utilisateurs en fournissant des comportements et des interactions proches des sensations du monde réel. (mattzmsft [sans date])

CHAPITRE 2 : RÉALISATION

Dans cette rubrique, nous allons expliquer la réalisation de ce projet, étant donné que nous nous sommes familiarisés avec les notions de base expliquées dans le chapitre un.

Il a été décidé de concevoir une application de réalité augmentée, qui affichera un histogramme déplaçable par l'utilisateur en temps réel. Cette application est connectée à un serveur REST qui récupère les données en temps réel et les mets automatiquement à jour si elles ont été modifiées.

1. CONFIGURATION DE UNITY

Avant de rentrer dans le vif du sujet, il est primordial de configurer l'environnement d'Unity pour qu'il fonctionne correctement avec l'HoloLens.

1.1. Configurer la camera

La position de la caméra doit être en x = 0, y = 0, z = 0, conformément à la documentation de Microsoft.

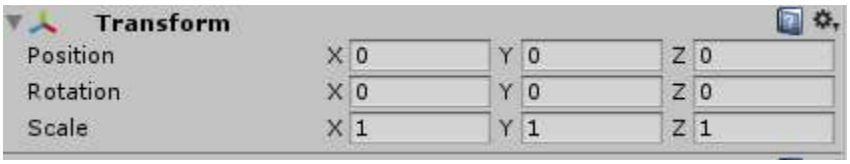

Figure 2 : Configurer caméra

Puis, nous devons modifier l'arrière-plan de notre application pour qu'elle soit transparente, sinon un fond de couleur sera affiché. Il faut donc changer la propriété « Clear Flags » en « Solide Color » et mettre le background en noir. Le noir étant transparent pour l'HoloLens.

Figure 3 : Configruer caméra Couleur

1.2. Compilation et déploiement

Pour que le projet fonctionne sur l'HoloLens nous devons impérativement modifier certains paramètres. Pour ce faire, nous devons aller dans Edit -> Project Setting -> Player

Les paramètres que nous souhaitons modifier se trouvent sur la droite dans l'inspecteur. Il faudra sélectionner le logo Windows affiché sur l'inspecteur. Ensuite, nous devons aller dans « XR Settings » et cocher la case « Virtual Reality Supported ». Cela indique que notre application a besoin des librairie Windows SDK pour son fonctionnement.

Figure 4: Compilation et déploiement player

Puis, nous allons aller dans « Publishing Settings » et cocher la case « SpatialPerception » et « InternetClient » dans « Capabilites ». La « SpatialPerception » est le paramètre pour faire fonctionner le « SpatialMapping » de Microsoft, tandis que l'« InternetClient » sert à activer le client REST pour qu'il accepte les communications réseaux.

Figure 5 : Compilation et déploiement Capabilities

A présent, nous allons aller dans File -> Build Setting et changer notre plateforme à Universal Windows Platform en sélectionner le logo Windows et cliquer sur « Switch Platform », afin que l'application soit compatible avec l'HoloLens.

Figure 6 : Compilation et déploiement Build

Ensuite nous allons faire les manipulations suivantes :

1) Cliquer sur « Add Open Scenes » pour ajouter une scène à l'application.

2) Ensuite, nous allons mettre le « Target device » au paramètre « HoloLens » , le « Build Typ »e à « D3D » et cocher la case « Unity c# projects ».

Figure 7 : Compilation et déploiement

2. DÉVELOPPEMENT

A présent que nous avons un environnement de travail adéquat, nous allons expliquer les différentes étapes que nous avons fait pour développer ce projet.

2.1. Histogramme

En voulant utiliser des histogrammes sur Unity, il a été décidé de chercher une solution déjà existante. Nous avons donc cherché sur l'Unity asset store différente solution et nous avons trouvé celle-ci :

1. https://assetstore.unity.com/packages/3d/props/tools/3d-charts-and-graphs-114377
2. https://assetstore.unity.com/packages/tools/gui/graph-and-chart-78488
3. https://assetstore.unity.com/packages/tools/modeling/quick-chart-20406

4. https://assetstore.unity.com/packages/tools/gui/radar-chart-50074
5. https://assetstore.unity.com/packages/tools/gui/graph-maker-11782
6. https://assetstore.unity.com/packages/tools/gui/prochart-46203

Notre choix s'est porté sur la librairie numéro 6 qui est ProChart car elle répondait à tous nos critères. De plus, le prix demandé pour cette librairie était abordable en comparaison à d'autres plus complètes mais avec un prix plus onéreux.

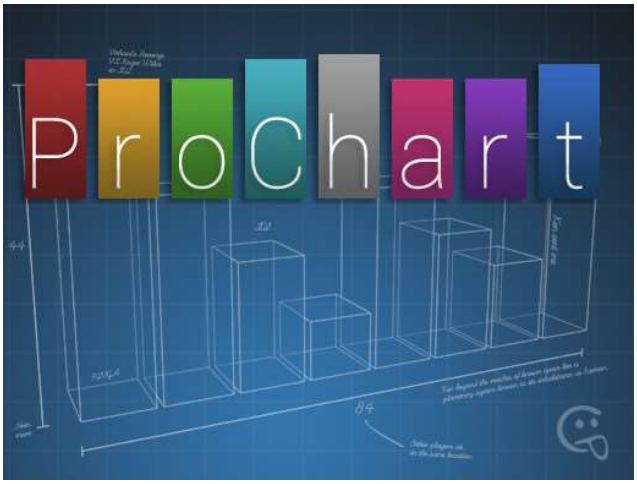

Figure 8 : ProChart

Comment utiliser cette librairie ?

Tout d'abord, nous devons importer la librairie ProChart à notre projet.

Puis, nous devons crée un Game Object dans notre hierarchy en faisant clique droit puis « Create Empty ».

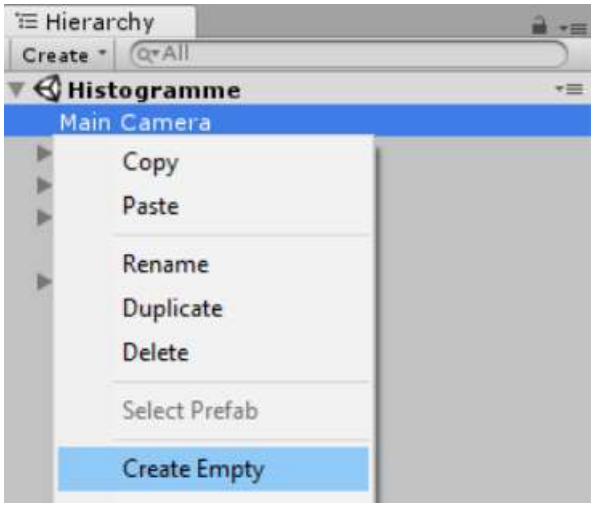

Figure 9 : Create

Puis, le script « Bar Chart mesh » est ajouté. Ce script permet de créer un histogramme en 3d. Il est nommé Bar3D.

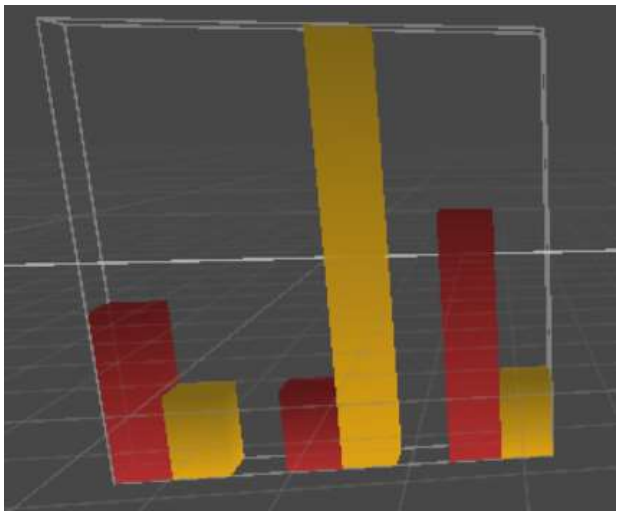

Figure 10 : Bar3D

Ensuite, il faut créer un script qui permet de gérer le graphique, ce script doit être enregistré dans le projet Unity.

Nous devons associer notre script à l'objet crée. Pour cela, il faut sélectionner le gameobject. Ensuite nous devons aller dans l'éditeur et cliquer sur « Add component ». Puis, rechercher dans la barre de recherche le nom du script et le sélectionner.

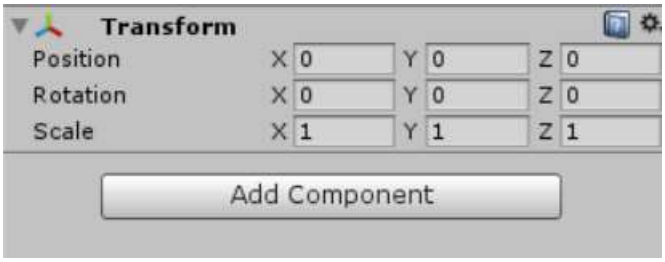

Figure 11 : Add component

Nous avons aussi ajouté le composant « Mesh Renderer » et « Box collider ».

Puis, nous ouvrons le script que nous venons de créer. Il faut lui dire d'utiliser la librairie ProChart.

```
using CP.ProChart;
```

Ce script doit avoir accès au gameobject Bar3d. Pour cela il faut ajouter une propriété publique. Cette propriété publique nous permettra de faire des manipulations sur l'objet.

```
public BarChartMesh bar3d;
```

Puis, nous retournons vers notre script et ajoutons l'objet à notre script.

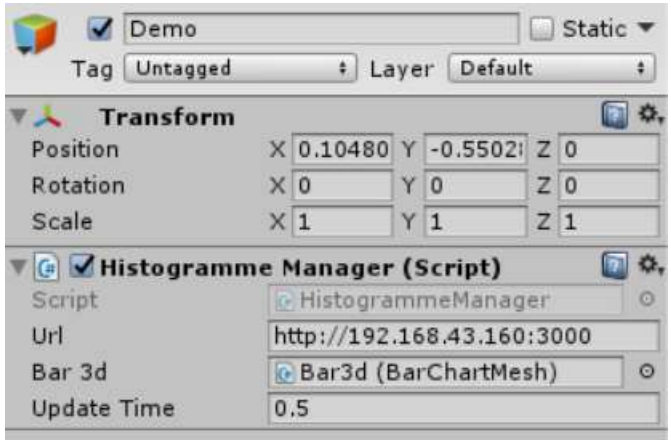

Figure 12: Ajouter script

Dans cet exemple, nous créons des graphiques à barres qui utilisent des données 2D :

Ensuite, il faut déclarer un tableau de la librairie ProChart pour pouvoir y mettre les valeurs que nous souhaitons.

```
ChartData2D dataSet = new ChartData2D();
```

Pour ajouter une donnée à notre tableau, il suffit de faire la commande suivante :

```
dataSet[0, 0] = 15f;
```

Pour mettre à jour l'objet bar3D avec nos données, il nous suffit de faire cette commande :

```
barChart.SetValues(ref dataSet)
```

Nous pouvons associer le tableau de données à plusieurs graphiques différents de façon simultanée.

A présent, qu'un graphique où des données peuvent y être insérées a été conçu, il nous est possible de nous intéresser à l'acquisition de données en temps réel. Pour cela, il nous faut modifier notre scripte pour ajouter une partie client-serveur. De cette façon, le scripte se connectera à un serveur REST pour récupérer les données.

Dans un premier temps, il nous faut déclarer les variables public `url`, `updateTime` et `results`. La variable `url` sert à renseigner l'adresse de notre serveur. `UpdateTime` est la variable qui définit le rafraîchissement par seconde et `results` est la variable où est stocké les données récupérées du serveur.

```
public string url = "http://192.168.43.160:3000";

public float updateTime = 2.0f;

private string results;
```

Puis, la fonction `WaitForRequest` est créée. Cette fonction va nous servir à créer une action dites parallèle pour récupérer les données sans ralentir le reste du programme grâce au coroutine. Une coroutine sers à suspendre l'exécution du code dans cette partie jusqu'à que le yield retourne une réponse. Cela permet d'éviter de mettre en pause Unity. (Technologies [sans date])

```
private IEnumerator WaitForRequest(WWW www, Action
onComplete)
    {
        yield return www;
        if (www.error == null)
        {
            results = www.text;
                //Debug.Log("results: " + results);
            onComplete();
            www.Dispose();
```

```
        }
        else
        {
            Debug.Log(www.error);
        }
    }
```

Puis, nous allons faire une requête au serveur pour récupérer les données à traiter. Le traitement des données se fait par la fonction `CollectResult`. Cette dernière sera expliquée plus tard.

```
private void WebQuery()
{
        //Worked
        irequest++;
        WWW www = new WWW(url);
        StartCoroutine(WaitForRequest(www, CollectResult));
}
```

Dans cette partie, nous créons la fonction `CoUpdate`. Cette fonction va appeler `WebQuery` et dire à Unity de se mettre en pause pendant un temps donné lorsque le travail sera terminé. Dans notre exemple, il s'agira de deux secondes. Le `yield` permet de dire a Unity que le travail est terminé.

```
private IEnumerator CoUpdate()
{
     while (true)
     {
         results = "";
         WebQuery();
         yield return new WaitForSeconds(updateTime);

    }
}
```

Traitement de donnée

Dans cette partie, voici la fonction que nous avons créé pour traiter les données reçues par le serveur. Cette fonction a pour but de parser le résultat et de mettre à jour l'histogramme. Nous avons dû traiter les données car les librairie JSON de c# n'était pas compatible.

Voici la fonction mentionnée :

```
private void CollectResult()
    {
        //Debug.Log(results);
        char[] delimiterChars = { ',', '[', ']', '{', '}',
'\t',':' };


        string[] words = results.Split(delimiterChars);
        int n = 0;
        bool test = true;

        dataSet = new ChartData2D();
        int i = 0;
        foreach (var word in words)
        {
            long number = 0;
            // verification si c'est un nombre
            bool canConvert = long.TryParse(word, out
number);
            if (canConvert == true)
            {
                if (test) {
                    n = Int32.Parse(word);
                    test = false;
                }
                else {
                     dataSet[i, 0]= Int32.Parse(word);
                     i++;
                }
            }
        }
        bar3d.SetValues(ref dataSet);
}
```

Puis, nous allons initialiser le projet en déclarant une coroutine dans la fonction Start. Cette fonction est appelée dès le démarrage du script.

```
private void Start()
{
    /*_coroutine = */
    StartCoroutine(CoUpdate());
```

```
}
```

2.2. Serveur REST

Notre serveur a été conçu sur ExpressJS. Nous allons expliquer les différentes modifications faites au serveur REST.

Tout d'abord, nous avons connecté la base de données à notre serveur en créant un fichier config.js. Ce dernier contient les informations pour se connecter à la base de données. Nous avons créé ce fichier pour centraliser les informations de la base donnée. Cela permettra de simplifier la modification de ces dernières.

```
/**
* Config for the database connection and server listening
port
* @type {{rethinkdb: {host: string, port: number, authKey:
string, db: string}, express: {port: number}}}
*/
module.exports = {
    rethinkdb: {
        host: "localhost",
        port: 28015,
        authKey: "",
        db: "ProjetSem"
    },
    express: {
        port : 8888
    }
}
```

Ces lignes de code permettant de connecter le module de la base de données au serveur.

```
//DATABASE CONNECTION
const async = require('async');
const r = require('rethinkdb');
const config = require(__dirname + '/config.js');
```

Puis, nous avons modifié les informations que la page d'accueil affichait, afin qu'elle affiche les données de la base de données.

```
route index.js
var express = require('express');
var router = express.Router();
     var r = require('rethinkdb');


/* GET home page. */
router.get('/', function (req, res, next) {
   r.table("data").filter(
       {"id": "a3fd0daa-a764-464c-a09d-
5d7e6f80609e"}  //50054ddb-7c30-4dfd-8429-6275ccb4d1e2
   ).run(app._rdbConn, function (err, result) {
       if (err) throw err;
       result.toArray(function (err, val) {
           if (err) return next(err);
           res.json(val);
       });
   });
});


module.exports = router;
```

La structure de données est la taille du tableau et le paramètre « data » qui contient les données du tableau.

Voici la structure de données d'une entrée :

```
{

      "a_size": 10 ,
      "data": [
          o 20 ,
          o 32 ,
          o 18 ,
          o 10 ,
          o 30 ,
          o 13 ,
          o 6 ,
          o 22 ,
          o 9 ,
          o 17
      ] }
```

2.4. Gestion HoloLens

Pour cette partie, nous avons suivi le tutoriel de Microsoft. Cela nous a permis d'avoir un cursor et de pouvoir déplacer nos hologrammes. Voici les étapes que nous avons réalisé pour avoir une application en réalité augmentée.

La sélection des hologrammes : « Gaze »

Tout d'abord, nous avons ajouté le GameObject Cursor à notre hiérarchie. Puis, nous avons créé le script worldCursor. Ensuite, nous l'avons ajouté au GameObject Cursor. Ce script permet d'avoir un cursor qui se déplace.

Voici le script :

```
public class WorldCursor : MonoBehaviour
{
    private MeshRenderer meshRenderer;

    // Use this for initialization
    void Start()
    {
        // Grab the mesh renderer that's on the same object
as this script.
        meshRenderer =
this.gameObject.GetComponentInChildren<MeshRenderer>();
    }

    // Update is called once per frame
    void Update()
    {
        // Do a raycast into the world based on the user's
        // head position and orientation.
        var headPosition = Camera.main.transform.position;
        var gazeDirection = Camera.main.transform.forward;

        RaycastHit hitInfo;

        if (Physics.Raycast(headPosition, gazeDirection, out
hitInfo))
        {
            // If the raycast hit a hologram...
            // Display the cursor mesh.
            meshRenderer.enabled = true;
```

```
            // Move thecursor to the point where the raycast
hit.
            this.transform.position = hitInfo.point;

            // Rotate the cursor to hug the surface of the
hologram.
            this.transform.rotation =
Quaternion.FromToRotation(Vector3.up, hitInfo.normal);
        }
        else
        {
            // If the raycast did not hit a hologram, hide
the cursor mesh.
            meshRenderer.enabled = false;
        }
    }
}
```

Interagir avec les hologrammes : « Gestures »

Pour cette partie, nous avons créé le script GazeGestureManager. Nous avons ajouté ce fichier à l'ensemble d'Object que nous souhaitons gérer. Ce script permet de gérer les gestes de l'utilisateur.

Voici le script :

```
using UnityEngine;
using UnityEngine.XR.WSA.Input;

public class GazeGestureManager : MonoBehaviour
{
    public static GazeGestureManager Instance { get; private
set; }

    // Represents the hologram that is currently being gazed
at.
    public GameObject FocusedObject { get; private set; }

    GestureRecognizer recognizer;

    // Use this for initialization
    void Start()
    {
        Instance = this;
```

```
        // Set up a GestureRecognizer to detect Select
gestures.
        recognizer = new GestureRecognizer();
        recognizer.Tapped += (args) =>
        {
            // Send an OnSelect message to the focused object
and its ancestors.
            if (FocusedObject != null)
            {
                FocusedObject.SendMessageUpwards("OnSelect",
SendMessageOptions.DontRequireReceiver);
            }
        };
        recognizer.StartCapturingGestures();
    }

    // Update is called once per frame
    void Update()
    {
        // Figure out which hologram is focused this frame.
        GameObject oldFocusObject = FocusedObject;

        // Do a raycast into the world based on the user's
        // head position and orientation.
        var headPosition = Camera.main.transform.position;
        var gazeDirection = Camera.main.transform.forward;

        RaycastHit hitInfo;
        if (Physics.Raycast(headPosition, gazeDirection, out
hitInfo))
        {
            // If the raycast hit a hologram, use that as the
focused object.
            FocusedObject = hitInfo.collider.gameObject;
        }
        else
        {
            // If the raycast did not hit a hologram, clear
the focused object.
            FocusedObject = null;
        }

        // If the focused object changed this frame,
        // start detecting fresh gestures again.
        if (FocusedObject != oldFocusObject)
        {
```

```
            recognizer.CancelGestures();
            recognizer.StartCapturingGestures();
        }
    }
}
```

Utiliser la cartographie spatiale : « Spatial Mapping »

Pour pouvoir avoir la cartographie spatiale. Il faut tout d'abord ajouter le GameObject SpatialMapping à notre hiérarchie. Puis, sélectionner l'Object que nous venons d'ajouter et cocher la case « Draw Visual Meshes » dans l'inspecteur.

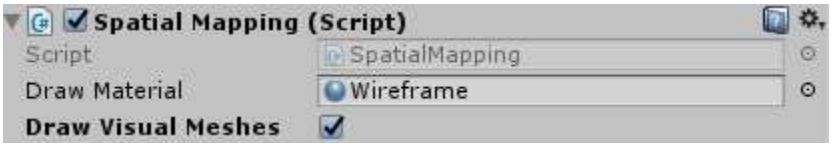

Figure 13 : Script Spatial Mapping

Puis, créer le fichier le script TapeToPlaceParent. Ensuite, nous avons ajouté ce script à notre histogramme. Ce script permet de déplacer notre histogramme dans l'espace grâce à l'Object SpatialMapping.

Voici le script :

```
public class TapToPlaceParent : MonoBehaviour
{
    bool placing = false;

    // Called by GazeGestureManager when the user performs a
Select gesture
    void OnSelect()
    {
        // On each Select gesture, toggle whether the user is
in placing mode.
        placing = !placing;

        // If the user is in placing mode, display the
spatial mapping mesh.
        if (placing)
        {
            SpatialMapping.Instance.DrawVisualMeshes = true;
        }
```

```
        // If the user is not in placing mode, hide the
spatial mapping mesh.
        else
        {
            SpatialMapping.Instance.DrawVisualMeshes = false;
        }
    }

    // Update is called once per frame
    void Update()
    {
        // If the user is in placing mode,
        // update the placement to match the user's gaze.

        if (placing)
        {
            // Do a raycast into the world that will only hit
the Spatial Mapping mesh.
            var headPosition =
Camera.main.transform.position;
            var gazeDirection =
Camera.main.transform.forward;

            RaycastHit hitInfo;
            if (Physics.Raycast(headPosition, gazeDirection,
out hitInfo,
                30.0f, SpatialMapping.PhysicsRaycastMask))
            {
                // Move this object's parent object to
                // where the raycast hit the Spatial Mapping
mesh.
                this.transform.parent.position =
hitInfo.point;

                // Rotate this object's parent object to face
the user.
                Quaternion toQuat =
Camera.main.transform.localRotation;
                toQuat.x = 0;
                toQuat.z = 0;
                this.transform.parent.rotation = toQuat;
            }
        }
    }
}
```

Puis, le bouton pour changer de scène a été créé. Ce bouton est un Object Capsule où un Object 3DText a été ajouté. Afin qu'il fonctionne, nous avons ajouté ce script :

```
public class ChangerSceneCanvas : MonoBehaviour
{
    void OnSelect()
    {
        // On each Select gesture, toggle whether the user is
in placing mode.
        SceneManager.LoadScene("CanvasDemoBar");
    }

}
```

(keveleigh [sans date])

CHAPITRE 3 : RÉSULTAT

Actuellement, l'application POC est capable de communiquer entre l'application HoloLens et le serveur web. Pour l'instant peu de scène sont disponibles, mais le système est implémenté pour que l'on puisse ajouter différentes requêtes au serveur pour afficher différents résultats en fonction de nos désirs. Malheureusement le traitement de donnée n'est pas adapté à la modification de la structure. L'application HoloLens permet de déplacer un Histogramme dans une pièce et de le déposer sur des surfaces (mur, table, ordinateur, etc.). Les informations affichées sur l'histogramme sont récupérées depuis le serveur et mettent à jours automatiquement l'histogramme.

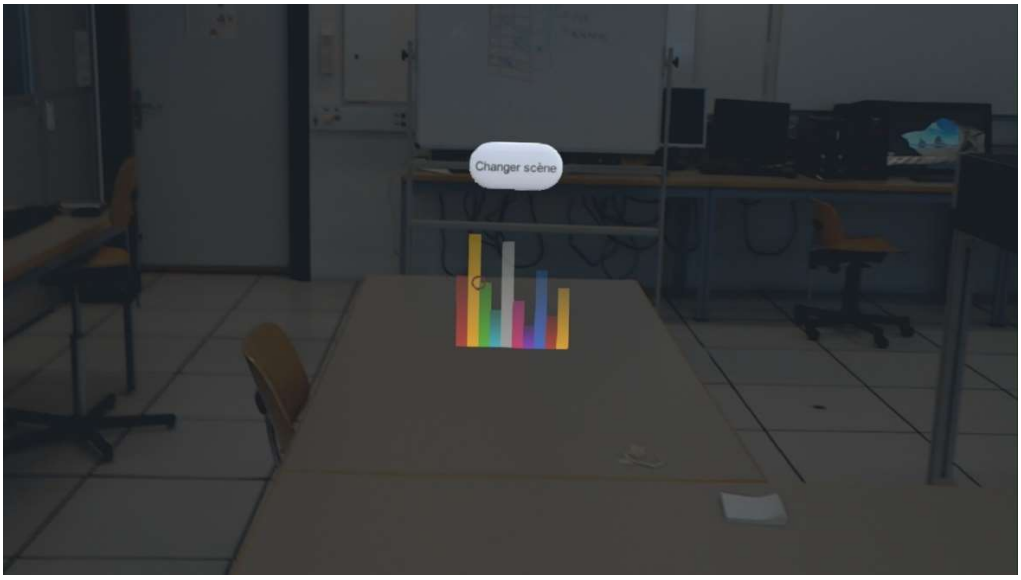

Figure 14 : Résultat

CONCLUSION

Nous avons pu découvrir comment concevoir une application en réalité augmentée. Ceci a été fait en plusieurs étapes : découvert d'Unity, réalisation d'une application sur l'HoloLens et enfin conception d'une application « Proof of concept ». L'objectif du POC était de générer des représentations 3D de données en temps-réel et d'augmenter la réalité par ces représentations contextualisées. Le problème principal qui a été rencontré fut qu'Unity n'était pas compatible d'une version à l'autre. Ceci pose un problème à tout projet développé sous Unity, car il demande de rester sur la même version. De plus, Microsoft devrait fournir une documentation beaucoup plus complète sur les différentes fonctionnalités concernant l'HoloLens. Mon temps a été grandement utilisés dans la recherche d'information sur des forums, ce qui n'est pas optimal et fiable pour tout projet professionnel.

Malheureusement le projet n'a pas entièrement abouti car j'ai eu énormément de difficultés à convertir la librairie ProChart en interface de réalité augmentée. L'erreur était que la version dans laquelle j'étais avait un bug sur les canevas et corrigé dans la version suivante. Par ailleurs, j'ai essayé de mettre à jour Unity, mais cela demandait la réécriture des scripts concernant les Objects de Microsoft. Et ceci ne fut pas envisageable, au vu du temps demandé par cette tâche et du temps restant pour le projet.

Pour les améliorations possibles, je pense qu'il serait intéressant d'ajouter la reconnaissance vocale au projet. De plus, il serait intéressant d'ajouter une partie édition où il est possible de modifier l'histogramme et de l'envoyer au serveur.

ANNEXES

ANNEXE 1

Lien projet : https://github.com/naores/ProjetSemestre.git

RÉFÉRENCES DOCUMENTAIRES

base-de-données.jpg (214×236), [sans date]. [en ligne]. [Consulté le 24 mars 2019]. Disponible à l'adresse : http://matrix-appliances.com/wordpress/wp-content/uploads/2018/05/base-de-donn%C3%A9es.jpg

Illustration, [sans date]. [en ligne]. [Consulté le 19 mars 2019]. Disponible à l'adresse : https://img.bfmtv.com/c/630/420/f7c/cb5b843a40661082f272e503a9315.jpg

KEVELEIGH, [sans date]. MR Basics 101E - Complete project with emulator - Mixed Reality. [en ligne]. [Consulté le 22 mars 2019]. Disponible à l'adresse : https://docs.microsoft.com/en-us/windows/mixed-reality/holograms-101eFollow this coding walkthrough using Unity, Visual Studio and HoloLens Emulator to learn the basics of a holographic application.

MATTZMSFT, [sans date]. Spatial mapping - Mixed Reality. [en ligne]. [Consulté le 22 mars 2019]. Disponible à l'adresse : https://docs.microsoft.com/en-us/windows/mixed-reality/spatial-mappingSpatial mapping provides a detailed representation of real-world surfaces in the environment around the HoloLens.

Microsoft Mixed Reality, [sans date]. *Unity* [en ligne]. [Consulté le 22 mars 2019]. Disponible à l'adresse : https://unity3d.com/fr/partners/microsoft/mixed-realityUnity is the ultimate game development platform. Use Unity to build high-quality 3D and 2D games, deploy them across mobile, desktop, VR/AR, consoles or the Web, and connect with loyal and enthusiastic players and customers.

mise-a-jour-hololens-ameliorations-660x330.jpg (660×330), [sans date]. [en ligne]. [Consulté le 19 mars 2019]. Disponible à l'adresse : https://www.realite-virtuelle.com/wp-content/uploads/2018/03/mise-a-jour-hololens-ameliorations-660x330.jpg

screenshot-2016-02-29-at-9-41-48-am.png (1000×563), [sans date]. [en ligne]. [Consulté le 24 mars 2019]. Disponible à l'adresse : https://pmcvariety.files.wordpress.com/2016/02/screenshot-2016-02-29-at-9-41-48-am.png?w=1000&h=563&crop=1

server-567943_960_720.png (512×720), [sans date]. [en ligne]. [Consulté le 24 mars 2019]. Disponible à l'adresse : https://cdn.pixabay.com/photo/2014/12/14/15/57/server-567943_960_720.png

TECHNOLOGIES, Unity, [sans date]. Unity - Scripting API: MonoBehaviour.StartCoroutine. [en ligne]. [Consulté le 24 mars 2019]. Disponible à l'adresse : https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html

THETUVIX, [sans date]. Gaze - Mixed Reality. [en ligne]. [Consulté le 22 mars 2019]. Disponible à l'adresse : https://docs.microsoft.com/en-us/windows/mixed-reality/gazeGaze is the first form of input and is a primary form of targeting within mixed reality.