

**Manual Técnico**  
Aplicación: ARotar

**Contenido:**

I. Configuración de Vuforia.....	2
II Configuración y uso de Unity-3D.....	7
III. Algunas consideraciones en Unity-3D: Mesh, prefabs y otras particularidades.....	7
IV Códigos y algoritmos utilizados en C#.....	10
V. Conclusiones y descarga de paquete del desarrollo de la app (diseños en Unity-3D y código fuente) .....	15
VI. Enlaces.....	16

## I. Configuración de Vuforia

Cuando se importa el SDK de Vuforia en Unity, los elementos agregados se pueden observar en la vista del proyecto. Vuforia añade varias características y objetos que serán utilizados para desplegar Realidad Aumentada: texturas, materiales, fuentes, botones y prefabs, por mencionar algunos.

Dentro de los prefabs, se encuentra una cámara propia de Realidad Aumentada. Esta sustituye la que usa por defecto Unity-3D, pues se requiere de una propia de RA que sea capaz de rastrear e identificar los marcadores.

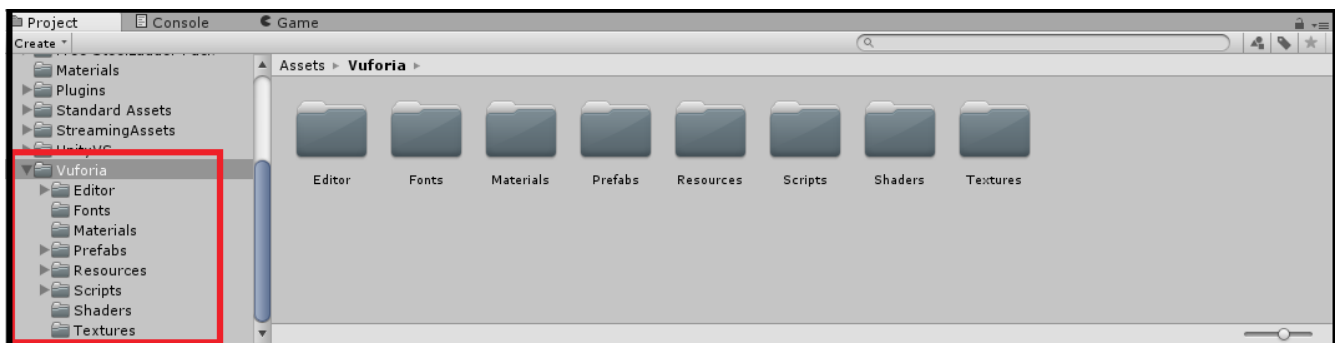


Figura 1. Vuforia en Unity-3D

Vuforia codifica un marcador (target) para que pueda ser utilizado por Unity-3D. Cuando se hace este procedimiento, la herramienta web que lo realiza, produce un paquete que se importa también a Unity. Esto es paralelo a la importación del SDK, un proceso complementario y de la misma naturaleza. Entonces se tienen dos diferentes paquetes de Vuforia que deben importarse al proyecto de Realidad Aumentada que se esté desarrollando en Unity-3D.

Una vez importado el SDK de Vuforia y el paquete del marcador, Unity-3D ya se encuentra en posibilidades de desarrollar un juego o una aplicación de Realidad Aumentada.

El SDK de vuforia, es en realidad, muy fácil de importar. Basta descargarlo de la página oficial y en la opción de importar paquetes dentro de la interfaz de Unity, elegir el paquete descargado.

La página oficial de vuforia es la siguiente: <https://developer.vuforia.com>.

Las acciones que se deben llevar a cabo para poder utilizar correctamente Vuforia en Unity, son las siguientes:

- Crear una base de datos para el image target. Es decir, el paquete que contendrá la información codificada que requiere el marcador de Realidad Aumentada. Aumentada.

-Adquirir una licencia de Vuforia (gratuita).

Estas dos opciones se encuentran dentro de la sección *Develop* de la página de Vuforia.

Para codificar o crear la base de datos del *image target*, hay que seleccionar dicha opción en el *target manager* que ofrece Vuforia vía web.

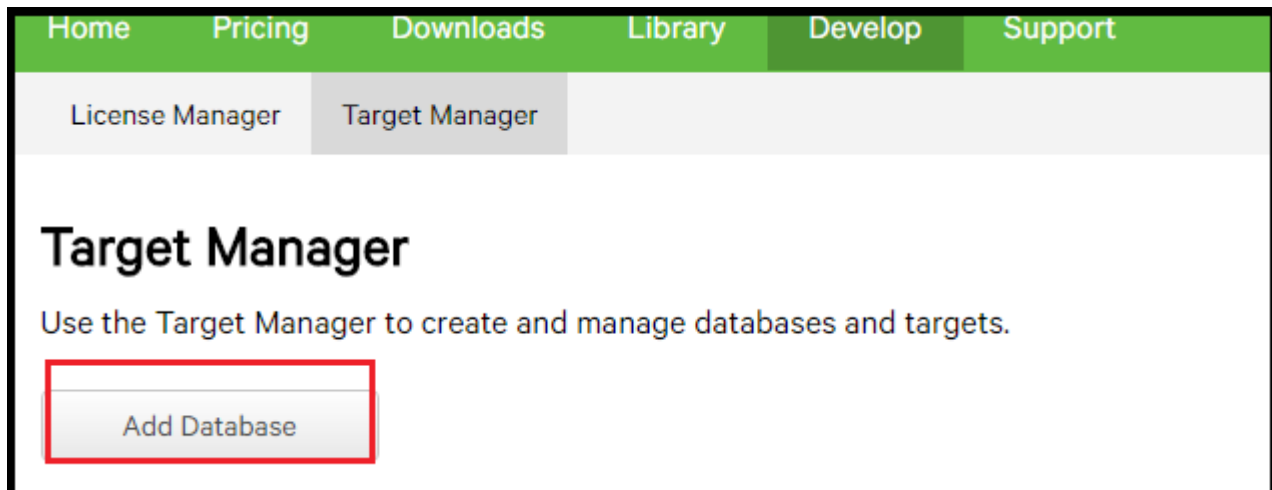


Figura 2-Página principal del Target Manager de Vuforia

Después, hay que elegir el elemento que será el marcador. Por razones de portabilidad, para esta app se eligió una imagen bidimensional.

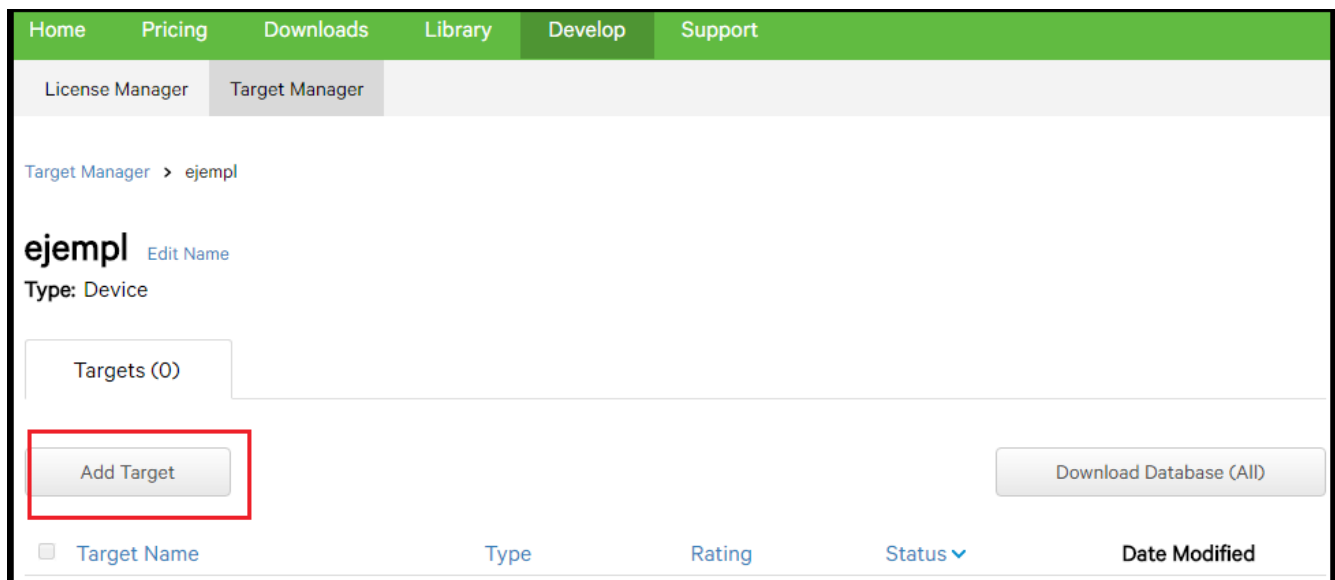





Figura 3. Añadir Target para Vuforia


### Add Target

Type:

  
Single Image

  
Cuboid

  
Cylinder

  
3D Object

File:

.jpg or .png (max file size 2mb).

Width:

Enter the width of your target in scene units. The size of the target should be on the same scale as your augmented virtual content. Vuforia uses meters as the default unit scale. The target's height will be calculated when you upload your image.

Name:

Name must be unique to a database. When a target is detected in your application, this will be reported in the API.

Figura 4 Configuración para añadir Target

El *target manager* de Vuforia, permite ver la calidad del marcador cargado. Mientras mejor sea ésta, mejor estabilidad mostrará la Realidad Aumentada generada y el rastreo por parte de la cámara también será superior.

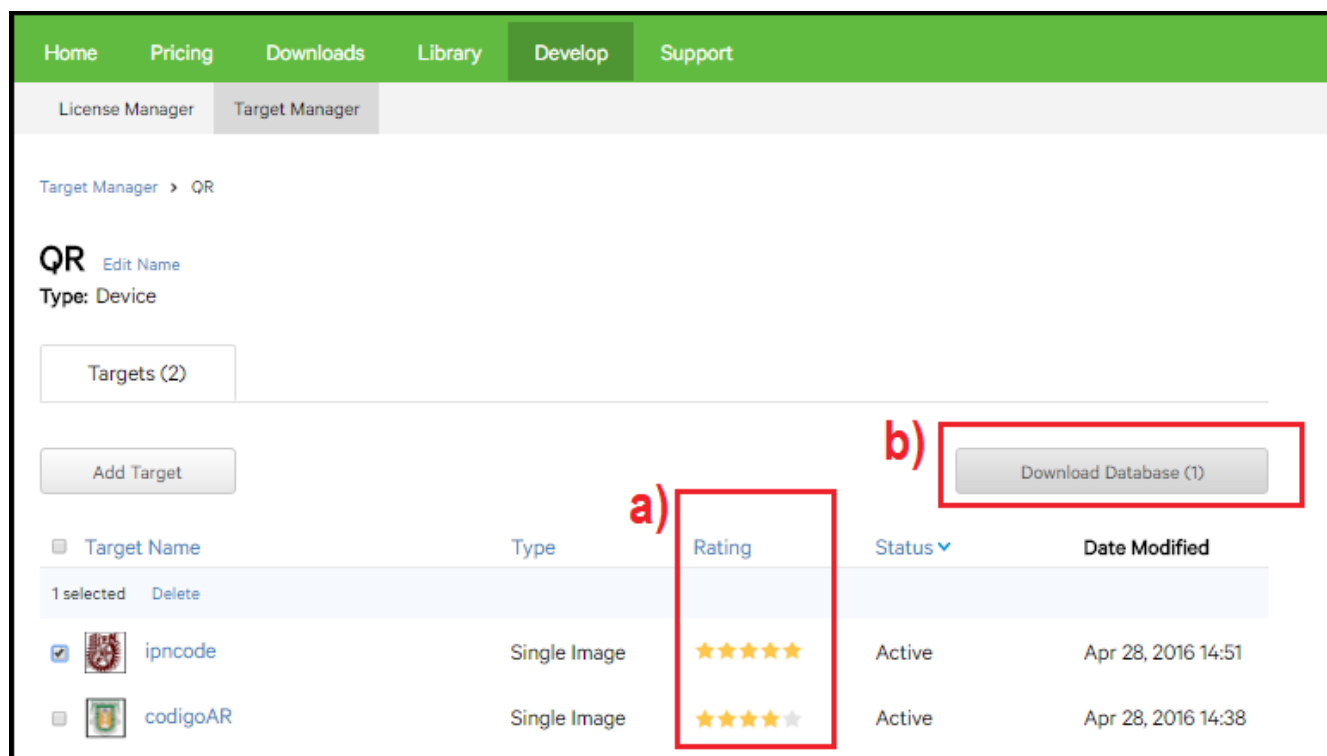


Figura 5. Calidad del Target: a) Calificación. b) Descargar sdk del target

Una vez cargada la base de datos del marcador junto con la imagen deseada, ya se puede proceder a descargar el paquete que se importará en Unity-3D.

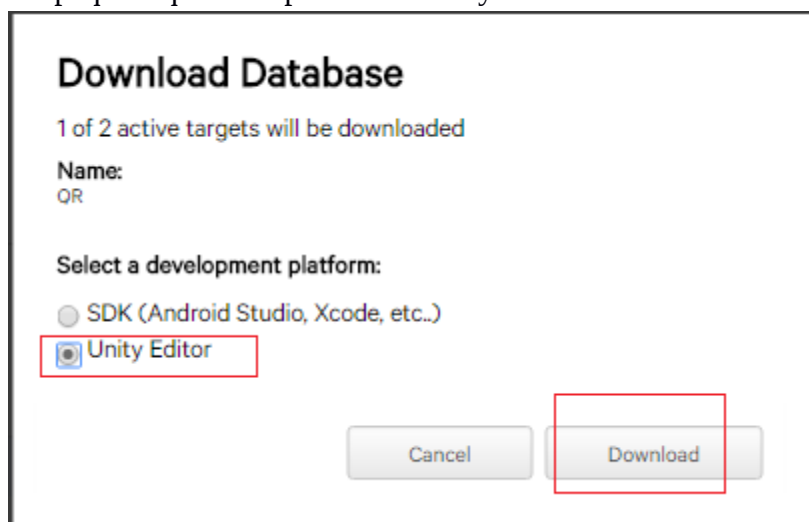
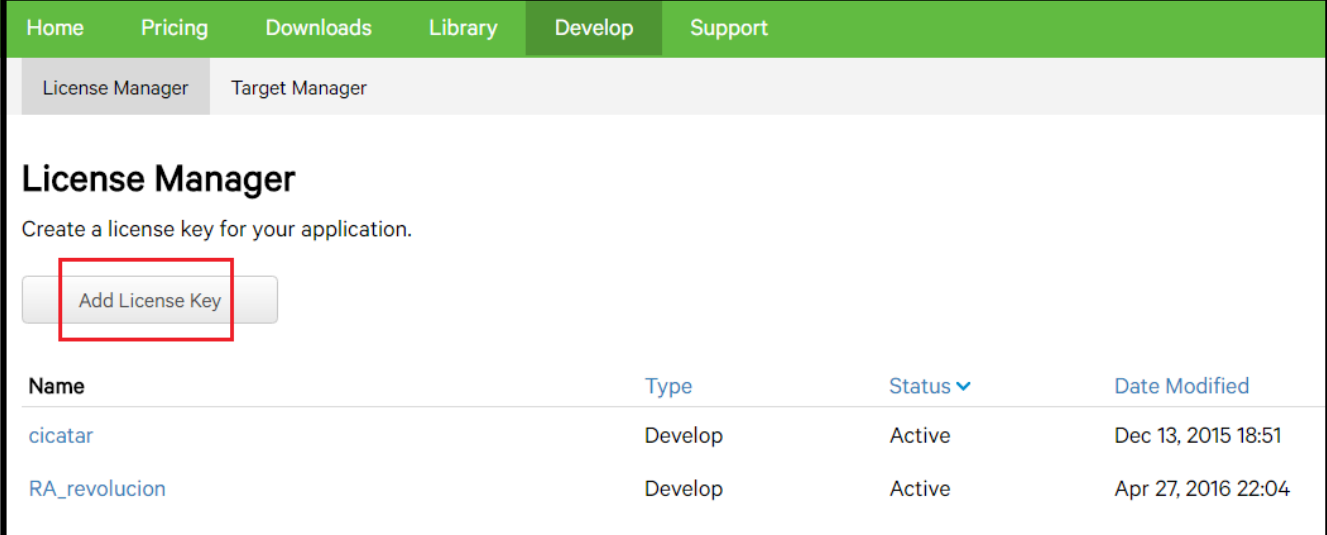


Figura 6 Descargar sdk del target para Unity-3D

El procedimiento para importar este nuevo paquete generado, es idéntico al que se emplea para importar cualquier otro paquete en Unity. De hecho, es igual a la manera en que se importa el SDK de Vuforia.

Ahora bien, para obtener la licencia de Vuforia, solo hay que indicar el motivo por el cuál se está desarrollando la aplicación en la sección *License* de la página de Vuforia.



**License Manager**

Create a license key for your application.

[Add License Key](#)

Name	Type	Status ▾	Date Modified
<a href="#">cicatar</a>	Develop	Active	Dec 13, 2015 18:51
<a href="#">RA_revolucion</a>	Develop	Active	Apr 27, 2016 22:04

*Figura 7 Creación de licencia libre de Vuforia*

Una vez llenados los campos que indica la página de Vuforia, se generará una cadena de texto que será la licencia. Ésta habrá que copiarla y después pegarla en la sección del inspector de Unity-3D, tal como se muestra en la imagen de la Figura 7.

La licencia es necesaria para que Unity reconozca y pueda utilizar los elementos importados de Vuforia, su omisión conlleva a una serie de errores en apariencia inexplicables dentro de la compilación en Unity-3D.

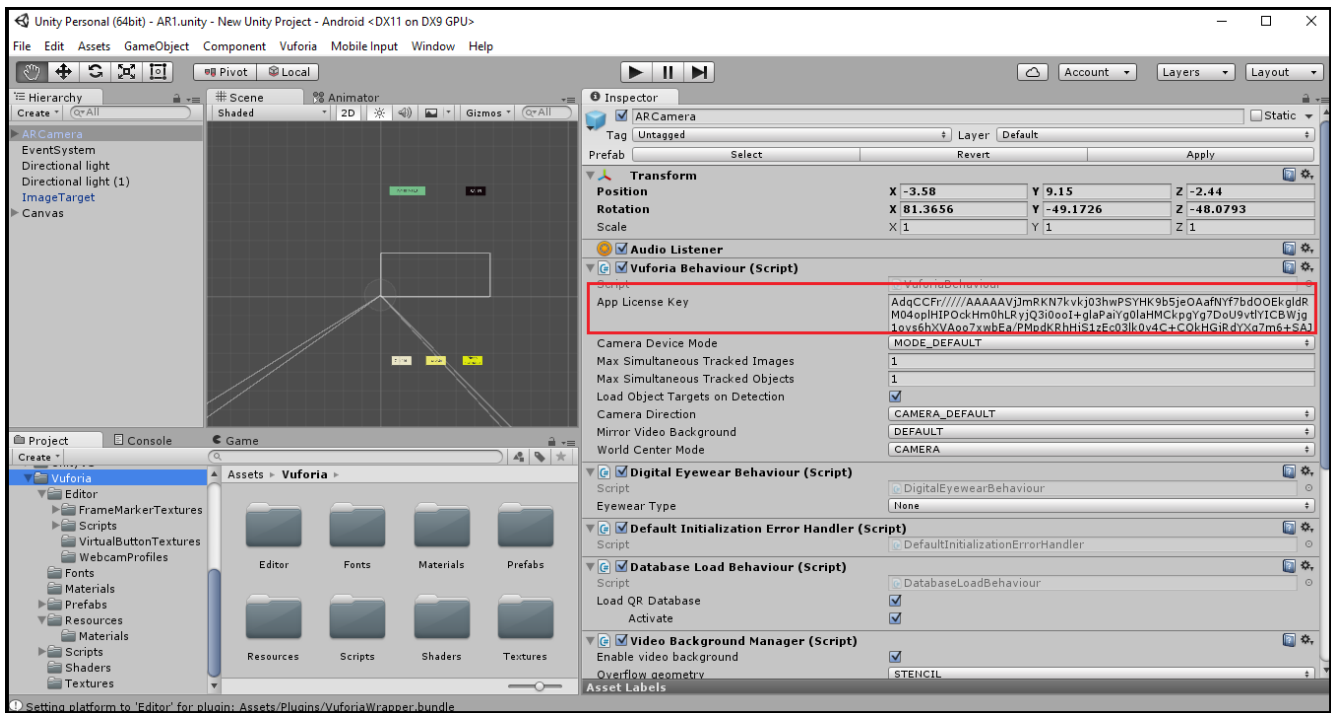


Figura 8 Añadir licencia de Vuforia en editor de Unity-3D

## II Configuración y uso de Unity-3D

Antes de empezar a desarrollar un proyecto en Unity-3D, hay que configurarlo correctamente junto con Vuforia:

- Abrir un nuevo proyecto 3D en Unity.
- Importar SDK de Vuforia
- Importar el paquete del marcador generado por la web de Vuforia.
- Introducir la licencia de Vuforia proporcionada en la web de Vuforia.
- Eliminar de la escena principal la cámara que Unity usa de manera predeterminada, y poner en su lugar la cámara AR que suministra Vuforia (assets < prefabs < camera AR).

Después de lo anterior, hay que agregar objetos a la escena principal y asignarles scripts que determinen sus acciones. Asimismo, determinar el resto de las escenas que serán parte de la aplicación.

## III. Algunas consideraciones en Unity-3D: Mesh, prefabs y otras particularidades.

Unity-3D permite la importación de modelos 3D con extensiones .3DS, .dxf y .obj, entre otras. De manera que si se ha obtenido algún modelo tridimensional de un tercero, no habrá inconveniente en utilizarlo. En el caso de este trabajo, no se importó ningún modelo 3D, todo se construyó con las

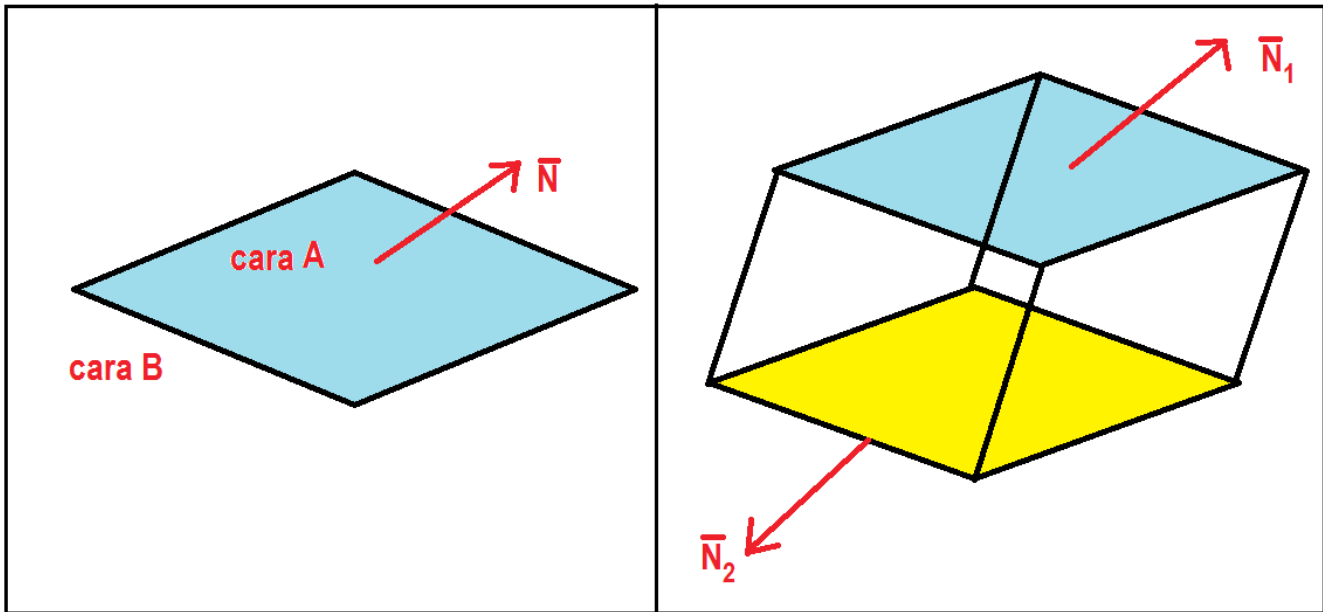
herramientas que Unity ofrece. Se eligió este camino debido a la naturaleza de las animaciones que se querían mostrar, las principales razones son las siguientes:

- Los modelos diseñados con software de diseño ajenos a Unity, como podrían serlo Blender o SketchUp, no importan a Unity las propiedades lógicas se requieren para poder manipularlos.
- Características como los vértices, son necesarios para que, por medio de un script, Unity sea capaz de deformarlos.
- En el ejemplo concreto de deformar o rotar una superficie para construir un sólido de revolución, una superficie diseñada en uno de estos programas, complicaría la tarea.

Por lo expuesto anteriormente, la superficies iniciales para generar los sólidos de revolución de la app, fueron basadas en objetos prediseñados en Unity. En realidad, se emplearon cuerpos tridimensionales con un grosor muy pequeño, de modo que parecieran superficies. La razón por la cual no se utilizaron superficies u objetos en dos dimensiones como punto de partida, fue debido a un atributo que los objetos en el diseño de videojuegos y de gráficos 3D poseen, que es conocido como mesh.

El mesh es la envoltura lógica de un cuerpo tridimensional, en la cual se establecen buena parte de las propiedades visuales de un objeto. En el caso concreto de Unity-3D, una superficie de un cuerpo tridimensional es visible si el vector normal a ella es positivo. Es relevante comentar esto porque al rotar una superficie 360 grados, se tienen que poder ver ambas caras de ella, así como cualquiera de las secciones transversales del sólido en revolución en construcción. Sin embargo, el vector normal solamente es positivo para cada lado de una superficie, de manera que un corte transversal de un cuerpo o la otra cara de un plano, sería invisible al intentar ser mostrado en la pantalla de la aplicación.





*Figura 9 Normales de la cara de un cuerpo*

Con el uso de cuerpos tridimensionales en lugar de superficies, se soluciona el problema del mesh, pues ambas caras tienen su propia normal y ya no hay conflicto con la visibilidad al rotarlas. De igual manera, se soluciona el problema de las secciones transversales, pues cada sección transversal tiene su propio mesh debido a que estaría conformado por varios cuerpos anidados. Basta con hacer a los cuerpos que se toman como punto de partida lo suficiente delgados, para que parezcan superficies. Así que para la app, las figuras que son rotadas: el rectángulo, el plano y la circunferencia, en realidad son prismas rectangulares y un cilindro, respectivamente.

Estas figuras se crearon en Unity-3D modificando en el inspector algunas de sus propiedades tales como el grosor o inclinación. Sin embargo, como al correr la aplicación no se debe ver ninguna de ellas hasta que el usuario presione un botón, se dejan como objetos creados en el proyecto pero se eliminan de la vista de la escena. Solo se mandan a llamar por medio de un script.

Ahora bien, para rotar a estos objetos alrededor de un eje imaginario, a cada uno de ellos se les asignó un objeto vacío. Unity-3D permite emparentar de manera muy fácil a dos objetos, basta arrastrar al hijo sobre el padre. En este caso, los tres cuerpos previamente diseñados y que serán las figuras iniciales de cada sólido de revolución, se emparentaron a un objeto vacío. Esto con la finalidad de que este objeto no visible permaneciera en el centro y girando sobre su propio eje. Al hacer esto, debido al emparentamiento los otros objetos también girarán de igual manera, pero a cierta distancia del centro. Describiendo una circunferencia.

#### IV Códigos y algoritmos utilizados en C#

En la escena principal es donde se desplegarán las animaciones generando los sólidos de revolución en Realidad Aumentada. Lo único que siempre se debe visualizar es el marcador, que es el que activará la posibilidad de desplegar la RA. El marcador es el objeto principal de toda la aplicación, de manera que el script de la generación de las animaciones se le asignó a este objeto. A continuación se describirá este código, que para fines prácticos, se ha dividido en cuatro partes.

```
public class ControlAndroid : MonoBehaviour
{
    float angulo = 0;
    public GameObject prefab;
    public GameObject prefab2;
    public GameObject prefab3;
    public int eleccion = 1;
    int toactivate = 0;
    public Transform toparent;

    GameObject[] figura = new GameObject[360];
}
```

Figura 10 Variables públicas

En la primera parte del código, se muestra que la clase del script se nombró *ControlAndroid*, y también se instancian las variables públicas, que más adelante se explicarán cada una. Unity ofrece la comodidad de poder cambiar el valor de las variables públicas desde el inspector, sin necesidad de editar el script.

A los primeros tres objetos de clase *GameObject* llamados *prefab*, *prefab2* y *prefab3*, se les asignaron las tres figuras iniciales que se rotan para generar los sólidos de revolución, estas figuras previamente diseñadas en Unity se asignan al script por medio del inspector. El *GameObject[ ] figura* es un arreglo que almacenará cada instancia que se genere en cada grado que se rote el ángulo, las guarda y al final, con un ciclo *for* desplegará con cada tap sobre la pantalla las secciones del sólido de revolución que acumuló hasta desplegarlo por completo.

La variable ángulo de tipo *float*, evidentemente, es el ángulo que irá cambiando hasta completar 360 grados.

*Transform* es una clase en la que se pueden manipular características de los objetos, tales como posición, rotación y escala; de modo que *toparent* será un elemento que facilite esto en la generación de los sólidos de revolución.

```

public void setEleccion(int sel)
{
    if (sel < 1 || sel > 3)
        return;
    if (eleccion != sel)
    {
        borrarfigura();
        eleccion = sel;
        rotar = true;
        angulo = 0;
        Vector3 p = this.transform.position;
        for (int n = 0; n < 360; n++)
        {
            switch (eleccion)
            {
                case 1:
                    figura[n] = (GameObject)Object.Instantiate(prefab, p, Quaternion.Euler(0.0f, angulo, 0.0f));
                    break;
                case 2:
                    figura[n] = (GameObject)Object.Instantiate(prefab2, p, Quaternion.Euler(0.0f, angulo, 0.0f));
                    break;
                case 3:
                    figura[n] = (GameObject)Object.Instantiate(prefab3, p, Quaternion.Euler(0.0f, angulo, 0.0f));
                    break;
            }

            figura[n].transform.parent = toparent;
            figura[n].SetActive(false);
            angulo = angulo + 1;
        }
        toactivate = 1;
        figura[0].SetActive(true);
    }
}

```

Figura 11 Método elección

El método principal fue llamado *setEleccion*. Es el encargado de rotar las objetos 360 grados y determinar cuál superficie prefab es la que tiene que rotar (Ver código en Figura 11). Evalúa la selección del usuario (int sel), dado que solo hay tres opciones posibles, si no es ninguna de ellas, simplemente retorna como se puede apreciar en el primer *if*. Si la opción es válida, en caso de que se haya elegido anteriormente otra opción, entonces se limpia la pantalla con el método *borrarfigura()* (Ver Figura 12).

El objeto p es del tipo Vector3, una estructura que representa a un vector en tres dimensiones, mismo al que se le pueden manipular su dimensión y orientación. Además, es un parámetro necesario para crear instancias dentro del mismo script. p tomará el valor de las coordenadas x,y,z, del GameObject oportuno y servirá como referencia para iniciar la rotación y ubicar el primer punto de partida en la generación de los sólidos de revolución.

Es fácil ver que la función que desempeña el *switch-case* en el código de la Figura 11, es para reaccionar según la opción de sólido de revolución que el usuario quiera desplegar. Ahora bien, en cada caso se está generando una instancia que tendrá diferente ángulo en cada paso por el ciclo. La instancia

se *parsea* para que sea congruente con el tipo `GameObject`. El método *instantiate* retorna un objeto similar al original pero rotado sobre el eje y, esto se puede observar en el último parámetro, donde las rotaciones en x y z se dejaron en cero. Los otros dos parámetros de *instantiate* son, en primer lugar, el objeto original y después la posición donde se situará la nueva instancia, indicada por p.

El *switch-case* está dentro de un ciclo *for* que va creando instancias a 360 ángulos distintos y consecutivos, hasta completar la revolución. El sumador de esta variable se encuentra al final del ciclo *for*. Antes de incrementarse, hay dos líneas de código importantes. La primera emparenta la respectiva instancia guardada en el elemento i del arreglo *figura* a *toparent*, éste último tiene asignado (desde el inspector) al objeto `ImageTarget`, de manera que todas las nuevas instancias de prefabs están emparentadas con éste.

Por otro lado, la línea de *figura[n].SetActive(false)*; hace que las instancias que se van generando en cada cambio de ángulo y guardando en el arreglo dentro del ciclo, no sean visibles para el usuario, si no se hiciera esto, se desplegaría en su totalidad cada sólido de revolución desde el primer tap en pantalla y no se apreciarían diferentes facetas de su generación. Por esta razón, ya fuera del ciclo *for*, se activa (se hace visible) solamente el primer elemento generado que sería la superficie inicial que se va a rotar en la animación.

Cabe destacar, que en el método *setEleccion* solo se genera de manera lógica todo el sólido de revolución dependiendo de la selección del usuario. Lo único que se muestra es la superficie inicial, es decir, la primera instancia creada dentro del script.

```
void borrarfigura()
{
    for (int i = 0; i < toparent.childCount; i++)
    {
        Destroy(toparent.GetChild(i).gameObject);
    }
}
```

Figura 12 Método borrar

El último método del script la clase *controlAndroid* es el método *Update*. Se trata de un método que en casi todos los videojuegos existe, pues es el que se encarga de cambiar el estado actual de la animación según la interacción del usuario.

En este caso, la variable entera *nbtouches* está contando el número de taps sobre la pantalla del celular o tableta y mientras que éste sea menor que 360, seguirá activando los elementos de un objeto

del tipo *GameObject[ ]*, llamado *figura*. Se activan, porque como ya se mencionó antes, cuando se generaron dentro del método *setEleccion* se mantuvieron desactivados. Así que en *Update*, es donde se logra que con cada tap se visualice, con un grado de rotación, cada una de las instancias producidas hasta completar el sólido de revolución.

```
void Update()
{
    int nbTouches = Input.touchCount;

    if (nbTouches > 0)
    {
        if (toactivate < 360)
            figura[toactivate++].SetActive(true);
    }
}
```

Figura 13 Método update

En la imagen de la Figura 14, se puede apreciar a) la vista de la escena de RA en Unity, donde solo se visualiza el marcador. b) La sección del inspector donde se elige el *image target* que se había codificado en la página web de Vuforia. c) También, se puede ver el lugar dentro del inspector donde se pueden asignar los valores de las variables públicas sin necesidad de entrar al código.

Nótese que allí se asignan los valores de *prefab*, *prefab2* y *prefab3* respectivamente. También se designa al *ImageTarget* a la variable *toparent* y 0 como valor inicial de la variable *eleccion*.

Los Game Objects que se asignaron a *prefab*, *prefab2* y *prefab3*, fueron diseñados previamente en Unity y ahí mismo se les dotó del color y la textura que les da el brillo y aspecto de las animaciones. Es un procedimiento muy sencillo a partir de modelos de cuerpos 3D que Unity provee. Para que parecieran superficies planas en lugar de cuerpos, se redujo el valor de una de sus dimensiones.

El código mostrado en las imágenes previas, concentra el funcionamiento de la RA dentro de la app. Sin embargo, dentro de la aplicación móvil, también hay otras opciones que contienen teoría y ejercicios. El código que gestiona este cambio de escenas y frames, es el que se muestra en la Figura 15.

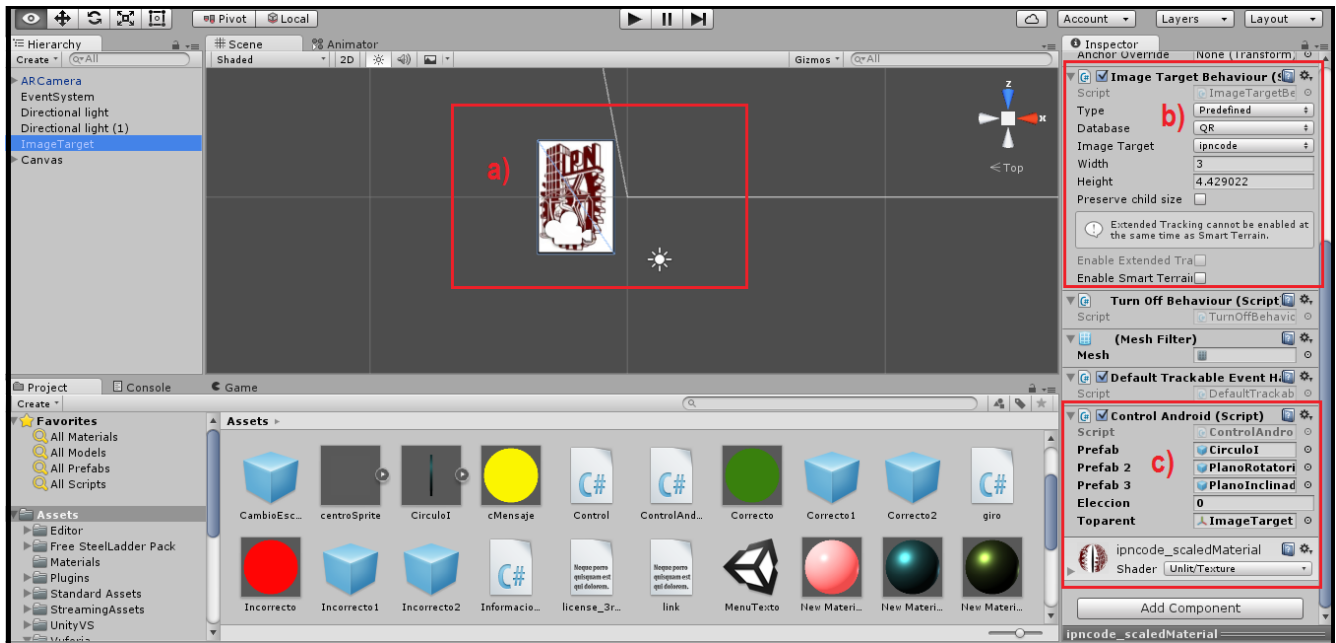


Figura 14 Escena de RA en Unity-3D. a) Target. b) Configuración Target. c) Inclusión del script ControlAndroid al objeto Target

```

public class InformacionTexto : MonoBehaviour
{
    public void TextoCilindro() {
        SceneManager.LoadScene("TextoCilindro");
    }

    public void MenuTexto() {
        SceneManager.LoadScene("MenuTexto");
    }

    public void VistaPrincipal() {
        SceneManager.LoadScene("AR1");
    }

    public void TextoToroide() {
        SceneManager.LoadScene("TextoToroide");
    }

    public void TextoPlano() {
        SceneManager.LoadScene("TextoPlano");
    }

    public void TextoIntroduccion() {
        SceneManager.LoadScene("TextoIntroduccion");
    }

    public void Salir(){
        Application.Quit();
    }
}

```

*Figura 15 Script cambio de escenas*

Cada cambio de pantalla equivale a una escena diferente, que en un videojuego sería un nuevo nivel. En las opciones del menú, se usaron herramientas UI de Unity, tales como Canvas, botones y checkbox.

## **V. Conclusiones y descarga de paquete del desarrollo de la app (diseños en Unity-3D y código fuente)**

Las secciones de la app que corresponden a algunas ideas teóricas de los sólidos de revolución, se crearon con la finalidad de complementar las animaciones 3D desplegadas. Se le podría agregar más teoría y ejercicios, la implementación sería muy sencilla. No obstante, la intención de esta aplicación móvil es que sea un material didáctico que apoye un curso formal de sólidos de revolución y no que lo sustituya. La intuición es lo que pretende desarrollar, más que la memorización de conceptos.

En el siguiente link, se puede descargar el proyecto de Unity-3D. Basta con descargar el proyecto e importarlo en el editor de Unity-3D. Incluye código fuente, prefabs y todo lo que se desarrolló para la app.

Para evitar fallas de compilación, es recomendable que las versiones de Unity-3D y del SDK de Vuforia coincidan con las descritas en este manual. Así mismo, el interesado tendrá que crear su propia licencia de Vuforia en el portal web para su editor de Unity-3D, de lo contrario las funciones de Realidad Aumentada no serán interpretadas por Unity-3D.

## VI. Enlaces

- Link para descargar el paquete de desarrollo en Unity-3D:  
[https://www.dropbox.com/sh/v6kyn1zjdyyhmie/AAB\\_AeMDv6OuEd5eskPG5E5va?dl=0](https://www.dropbox.com/sh/v6kyn1zjdyyhmie/AAB_AeMDv6OuEd5eskPG5E5va?dl=0)
- Github del autor con manuales e información sobre la app:  
<https://github.com/neorelativista/RAsolidsrev>
- Página oficial de Vuforia para generar el sdk del target y obtener la licencia:  
<https://developer.vuforia.com>
- E-mail de contacto: [igomezv0701@alumno.ipn.mx](mailto:igomezv0701@alumno.ipn.mx)