



UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE COMPUTACIÓN

VisualizAR

Por

Christian Oliveros 13-11000
Alexander Romero 13-11274

Realizado con la asesoría de:
Angela Di Serio

EP-4793 Mini Proyecto de Desarrollo de Software

Sartenejas, 18 de diciembre de 2018

ÍNDICE GENERAL

I. Introducción	1
II. Herramientas Utilizadas	2
2.1. Unity 2018.3.0f2	2
2.1.1. Instalación	2
2.1.2. Unity Collaborate	3
2.2. Vuforia SDK v.7.5.26	3
2.2.1. Instalación	3
2.2.2. Vuforia Engine	3
2.2.3. Herramientas	4
2.3. Android SDK y Manager SDK	4
2.3.1. Instalación	4
2.4. Distribución	4
III. Desarrollo	6
3.1. VisualizAR: Vectores	6
3.1.1. Image Render Target de Vuforia	6
3.1.2. Line Renderer	8
3.1.3. Scripts	9
3.1.3.1. Implementación General de Vectores	9
3.1.3.1.1. Representación de Vectores	9
3.1.3.1.2. Tracker	10
3.1.3.1.3. Interfaz Gráfica (Componentes X,Y,Z y Módulo del Vector)	10
3.1.3.2. Operaciones de vectores	10
3.1.3.2.1. Suma	11
3.1.3.2.2. Resta	11
3.1.3.2.3. Producto Punto	12
3.1.3.2.4. Producto Cruz	12
3.1.3.2.5. Limpiar	12
3.1.3.3. Componentes adicionales	13
3.1.3.3.1. Ejes del origen	13
3.1.3.3.2. Proyección del Vector	13
3.1.3.3.3. Planos en el Origen	14

3.1.4. Consideraciones Importantes al Momento de Realizar Operaciones	15
3.2. VisualizAR: Gráficas	16
3.2.1. Image Render Target de Vuforia	17
3.2.2. Shader	18
3.2.3. Scripts	19
3.2.3.1. Manejo de Gráficas	19
3.2.3.2. Calculadora	20
3.2.3.3. Funciones	21
3.2.3.3.1. Seno	21
3.2.3.3.2. Seno 2D	21
3.2.3.3.3. Media del Seno	22
3.2.3.3.4. Multiseno	22
3.2.3.3.5. Multiseno 2D	23
3.2.3.3.6. Onda	23
3.2.3.3.7. Círculo	23
3.2.3.3.8. Remolino	24
3.2.3.3.9. Cilindro	24
3.2.3.3.10. Cilindro Tambaleante	25
3.2.3.3.11. Cilindro de Olla	25
3.2.3.3.12. Cilindro de Torsión	25
3.2.3.3.13. Esfera	26
3.2.3.3.14. Esfera Cardioide	26
3.2.3.3.15. Esfera Pulsante	27
3.2.3.3.16. Toroide Exterior	27
3.2.3.3.17. Toroide de Intersección	27
3.2.3.3.18. Toroide de Cuerno	28
3.2.3.3.19. Toroide Anidado	28
3.2.3.3.20. Toroide Estrella	29
3.2.3.3.21. Banda de Möbius	29
3.2.3.3.22. Banda Especial de Möbius	29
3.2.3.3.23. Botella de Klein Estática	30
3.2.3.3.24. Botella de Klein Dinámica	30

ÍNDICE DE FIGURAS

3.1. Ejemplo de la aplicación de Vectores.	6
3.2. Marca de Centro para controlar el origen de los vectores.	7
3.3. Marcas de Vector 1 y Vector 2 para controlar los destinos de los vectores.	8
3.4. Escena de Unity con las opciones “Suma”, “Ejes Dobles” y “Proyección Vector 2” activadas, la cual en total suma 16 <i>Line Renderers</i> instanciados.	9
3.5. Representación de vectores con sus marcas en la aplicación.	10
3.6. Operación sumar en la aplicación.	11
3.7. Operación restar en la aplicación.	11
3.8. Operación producto punto en la aplicación.	12
3.9. Operación producto cruz en la aplicación.	12
3.10. Opción de ejes positivos y negativos activada en la aplicación.	13
3.11. Opción de proyección del Vector 1 activada.	14
3.12. Opción de plano en el eje x activada.	15
3.13. Opción de planos en los ejes x,y,z activada.	15
3.14. Ejemplo de la aplicación de Gráficas.	16
3.15. Ejemplo de la aplicación de Gráficas.	16
3.16. Ejemplo de la aplicación de Gráficas.	17
3.17. Marca de Botones para controlar la aplicación.	17
3.18. Marca de Gráficas para controlar la posición donde visualizar las funciones.	18
3.19. Ejemplo de la visualización del espacio vectorial mediante colores RGB.	18
3.20. Gráfica del círculo donde se observan los cubos que forman la malla.	19
3.21. Calculadora como se ve en la aplicación.	20
3.22. Gráfica de la función Seno.	21
3.23. Gráfica de la función Seno 2D.	22
3.24. Gráfica de la función Media del Seno.	22
3.25. Gráfica de la función Mutiseno.	22
3.26. Gráfica de la función Mutiseno 2D.	23
3.27. Gráfica de la función Onda.	23

3.28. Gráfica de la función Círculo.	24
3.29. Gráfica de la función Remolino.	24
3.30. Gráfica de la función Cilindro.	24
3.31. Gráfica de la función Cilindro Tambaleante.	25
3.32. Gráfica de la función Cilindro de Olla.	25
3.33. Gráfica de la función Cilindro de Torsión.	26
3.34. Gráfica de la función Esfera.	26
3.35. Gráfica de la función Esfera Cardioide.	26
3.36. Gráfica de la función Esfera Pulsante.	27
3.37. Gráfica de la función Toroide Exterior.	27
3.38. Gráfica de la función Toroide de Intersección.	28
3.39. Gráfica de la función Toroide de Cuerno.	28
3.40. Gráfica de la función Toroide Anidado.	28
3.41. Gráfica de la función Toroide Estrella.	29
3.42. Gráfica de la función Banda de Möbius.	29
3.43. Gráfica de la función Banda Especial de Möbius.	30
3.44. Gráfica de la función Botella de Klein Estática.	30
3.45. Gráfica de la función Botella de Klein Dinámica.	31

LISTA DE ACRÓNIMOS

USB Universidad Simón Bolívar

DEP Decanato de Estudios Profesionales

PDF *Portable Document Format*, Documento en Formato Portable©

3D *Three Dimensions*, Tres Dimensiones

API *Application Programming Interface*, Interfaz de Programación de Aplicaciones

RGB *Red Green Blue*, Rojo Verde Azul

AR *Augmented Reality*, Realidad Aumentada

MB *Mega Byte*

cm Centímetro

GPU *Graphics Processing Unit*, Unidad de Procesamiento Gráfico

SDK *Software Development Kit*, Kit de Desarrollo de Software

AR/RA *Augmented Reality*, Realidad Aumentada

VR *Virtual Reality*, Realidad Virtual

CAPÍTULO I

INTRODUCCIÓN

La tecnología de AR (*Augmented Reality*) está apareciendo cada vez con más frecuencia en el mundo educativo. En una discusión entre miembros de la unidad educativa realizada en línea en julio de este año [3], la gran mayoría estuvo de acuerdo en dos puntos principales. Primero el uso de realidad aumentada es beneficioso tanto para el proceso de aprendizaje de los estudiantes, como para la facilidad de los profesores en el momento de la enseñanza. Segundo, solo una pequeña cantidad de estudiantes ha tenido acceso al uso de las tecnologías de AR y VR (*Virtual Reality*) en el ámbito educativo; este último punto también viene respaldado por un estudio realizado por la asociación EDUCAUSE [1].

Tomando en consideración estos dos primeros puntos, se planteó como objetivo principal realizar una aplicación que utiliza realidad aumentada para el área educativa. Por otra parte, en bachillerato, la enseñanza de las matemáticas son fundamentales para el desarrollo académico del estudiante. Uno de los temas principales, y que son la base para todas las matemáticas tanto en el ciclo educativo secundario como en los estudios superiores, son los vectores y las funciones.

Este proyecto, denominado VisualizAR fue el desarrollo de una aplicación con dos módulos principales:

1. Vectores: se encarga de la representación de vectores en el plano tridimensional, acentuándose en las operaciones gráficas de vectores suma, resta, producto cruz, producto punto y proyección.
2. Gráficas: se basa en el uso de ecuaciones matemáticas (principalmente con senos y cosenos) para representar varias de las funciones más representativas gráficamente.

El proyecto fue realizado utilizando el motor de videojuegos Unity 3D, que además presenta una gran facilidad y accesibilidad para realizar aplicaciones que necesiten cálculos matemáticos.

El trabajo a continuación es la documentación de las herramientas utilizadas para el desarrollo del proyecto, así como la implementación de forma detallada de cada uno de los módulos de la aplicación VisualizAR.

CAPÍTULO II

HERRAMIENTAS UTILIZADAS

En este capítulo se explican todas las herramientas utilizadas para el desarrollo de este proyecto.

2.1. Unity 2018.3.0f2

Unity es un motor de videojuegos multiplataforma, creado por Unity Technologies. Está disponible como plataforma de desarrollo para Microsoft Windows y OS X. El enfoque de la compañía es “democratizar el desarrollo de juegos” hacer el desarrollo de contenidos interactivos en 2D y 3D lo más accesible posible a personas en todo el mundo.

2.1.1. Instalación

Unity se puede descargar desde la página de descarga de Unity. El instalador usa un asistente de descarga muy bien documentado. El Asistente de descargas de Unity es un programa ejecutable de tamaño aproximadamente 1MB, que permite seleccionar los componentes del Editor de Unity que se desean descargar e instalar. El asistente posee una selección predefinida, en caso de que no se sepa cuáles componentes se quieren instalar. Para el desarrollo de VisualizAR, además de la selección predefinida, se debe escoger el componente de Vuforia.

Si se desea instalar Unity sin el asistente de descarga, o multiples versiones simultaneamente, Unity ofrece la opción de descarga por Torrent. Para más información acerca de este método se puede referir a la documentación de Unity acerca del mismo.

Los requerimientos del sistema, al momento del desarrollo de este proyecto, para el correcto funcionamiento de Unity son:

- Sistema Operativo: Windows 7 SP1+, 8, 10, solo versiones de 64-bit; MacOS X 10.9+.
- GPU: Tarjeta gráfica con capacidad DX9 (*shader model 2.0*). Cualquier GPU hecho luego del 2004 debería funcionar.

2.1.2. Unity Collaborate

Unity Collaborate es la manera que ofrece la plataforma para que un equipo salve, comparta y sincronice los proyectos de Unity. Está alojado en la nube y es fácil de usar, permitiendo a todo el equipo contribuir en el proyecto, sin importar donde se encuentre. Mantiene un historial de las versiones del proyecto, permitiendo restaurar archivos individuales o el proyecto entero a una versión anterior. Se pueden agregar miembros del equipo al proyecto. Collaborate revisa de manera continua los cambios hechos por cada miembro del equipo y muestra quien ha editado y publicado archivos. Los cambios se pueden ver, revertir, publicar y manejar conflictos que puedan ocurrir al mezclar versiones.

2.2. Vuforia SDK v.7.5.26

Vuforia es un Kit de Desarrollo de Software (SDK por sus siglas en inglés) para Realidad Aumentada (RA) para dispositivos móviles, que permite la creación de aplicaciones de RA tanto en Android como en iOS. Utiliza tecnología de Vision Computacional para reconocer y rastrear imágenes.

2.2.1. Instalación

A partir de la versión 2017.2, Unity viene integrado con el Motor de Vuforia, facilitando la creación de experiencias de AR para dispositivos de mano y de VR (*Virtual Reality*). Para instalar y correr Vuforia, se debe descargar el asistente de descarga de Unity 2017.2 en adelante, en nuestro caso el asistente de descargas de Unity 2018.3.

2.2.2. Vuforia Engine

El motor de Vuforia es el lado del cliente de la librería que está asociado a la aplicación. Está disponible a través del cliente SDK. Como soporta Android, IOS y UWP, se puede usar Android Studio, Xcode, Visual Studio o Unity para construir la aplicación.

2.2.3. Herramientas

Vuforia posee herramientas para la creación de *targets*(objetivos de AR), manejo de la base de datos de *targets* y asegurar la licencia de la aplicación.

El Vuforia *Model Target Generator* (disponible para Android) permite generar targets a partir de modelos 3D de un objeto físico.

El Vuforia *Object Scanner* (disponible para Android) ayuda a escanear objetos 3D y convertirlos en targets que son compatibles con el motor de Vuforia.

El *Target Manager* es una aplicación web que se encuentra en el portal de desarrolladores que permite crear bases de datos de *targets* para usar en dispositivos y en la nube (para un número grande de *targets*).

2.3. Android SDK y Manager SDK

Se utilizó el Android SDK y el Manager SDK para compilar la aplicación para dispositivos Android.

2.3.1. Instalación

Para usar el SDK de Android se debe descargar el instalador de Android Studio de la página oficial de Android. Correr el instalador e iniciar Android Studio, seleccionar standard como tipo de setup y finalizar la instalación. Para seleccionar los SDK necesarios abrir Android Studio y seleccionar en el *dropdown* de configuraciones la opción de SDK manager.

Luego escoger las distintas SDK para las que se quiere desarrollar la aplicación y hacer click en *Apply*. Para el momento de desarrollo de este proyecto Vuforia era soportado por las versiones de Android de Jellybean hasta Oreo, así que estos fueron los SDK seleccionados.

2.4. Distribución

Para poder distribuir el binario (*apk* para Android) y los archivos fuente, se eligió la plataforma GitHub la cual facilita la distribución de proyectos *Open Source*.

Para acceder a los archivos fuente se tiene que seguir el enlace <https://github.com/maniatic0/VisualizAR> y utilizar el manejador de versiones git.

Para acceder al binario se tiene el enlace <https://github.com/maniatic0/VisualizAR/releases>. Para instalar es necesario tener activado las opciones desarrollador. Se espera que en un futuro la aplicación pueda ser distribuida en la PlayStore de Google.

CAPÍTULO III

DESARROLLO

En este capítulo, se explica todo lo relacionado al desarrollado de la aplicación.

3.1. VisualizAR: Vectores

La primera sección de la aplicación, consiste en la representación gráfica de vectores en el eje de tres dimensiones, junto con las operaciones principales de suma, resta, producto punto y producto cruz entre dos vectores. Esto permite la facilidad de aprendizaje para los estudiantes de bachillerato que están aprendiendo sobre este tema al poder observar gráficamente cómo se resuelven las operaciones de vectores. Un ejemplo de una vista de la aplicación se

Para la fácil representación de vectores, se decidió utilizar tres *Image Render Target* de Vuforia, uno denominado “Centro” que representará el origen de ambos vectores (o las coordenadas $(0, 0, 0)$), y aparte “Vector 1” y “Vector 2”, los cuales representan al destino de cada uno de los vectores correspondientes.

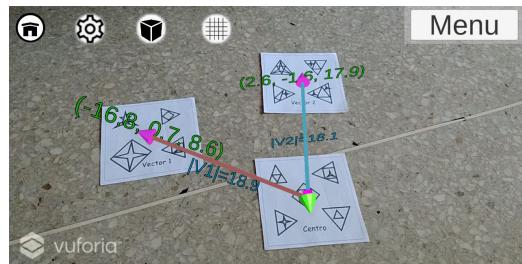


Figura 3.1: Ejemplo de la aplicación de Vectores.

3.1.1. Image Render Target de Vuforia

Para poder utilizar el motor de realidad aumentada de Vuforia, se generan tres *Image Render Target*. Estos fueron utilizados como el centro donde

comenzará cada uno de los vectores y los destinos del primer y segundo vector para controlar sus coordenadas. Para facilitar el reconocimiento por parte de Vuforia y minimizar las veces que los usuarios tengan que volver a apuntar las marcas con la cámara, se utilizaron figuras geométricas sencillas que apuntan en distintas direcciones con la finalidad de que el reconocedor de Vuforia tenga más información de la dirección de la imagen.

En las Figuras 3.2 y 3.3 se puede observar las marcas utilizadas, en estas se observa que se agregaron los nombres bajo ellas para facilitar su uso por los usuarios.

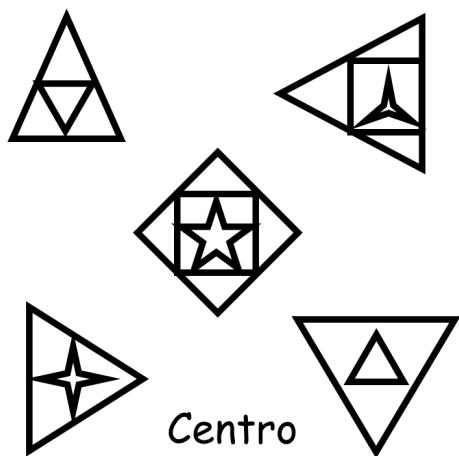


Figura 3.2: Marca de Centro para controlar el origen de los vectores.

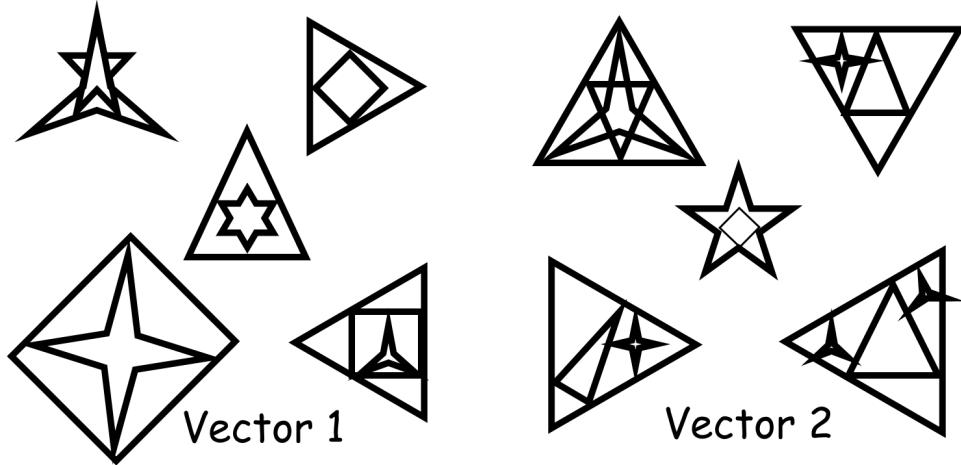


Figura 3.3: Marcas de Vector 1 y Vector 2 para controlar los destinos de los vectores.

3.1.2. Line Renderer

Para representar los vectores gráficamente, se utilizó el componente *Line Renderer* de Unity para el trazado de líneas y la simulación de los vectores a representar. Este componente es fácil de instanciar en la escena de la aplicación y de modificar, lo que permite que tanto los vectores, como los cálculos se realicen instantáneamente tanto al reconocer como al mover alguna de las marcas, lo que permite mayor agilidad y rapidez en el uso de la aplicación sin tener que esperar a que se vuelvan a calcular los datos. El principal problema que tiene este componente es que cada línea debe estar asociado a un *GameObject* asociado, lo cual podría traer problemas de rendimiento si se instancian una gran cantidad de líneas en una escena. A pesar de esto, el proyecto no se vió afectado por esto ya que la mayor cantidad de líneas instanciadas a la vez, es igual a 16 (Figura 3.4), siendo la menor solo 2 al sólo representar los vectores.

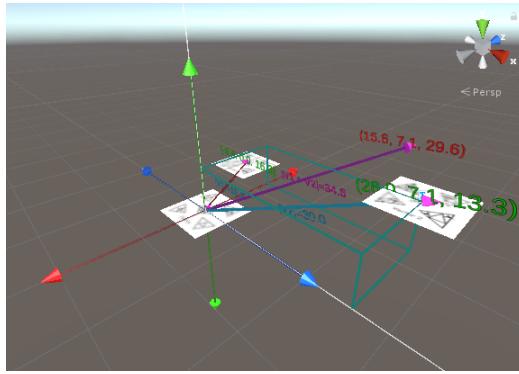


Figura 3.4: Escena de Unity con las opciones “Suma”, “Ejes Dobles” y “Proyección Vector 2” activadas, la cual en total suma 16 *Line Renderers* instanciados.

3.1.3. Scripts

Los *Scripts* se pueden dividir en tres grupos distintos, correspondiente a sus funcionalidades, implementación general de vectores, operaciones de vectores y componentes adicionales.

3.1.3.1. Implementación General de Vectores

3.1.3.1.1. Representación de Vectores Cada vector viene representado en la aplicación por dos *GameObjects*, el primero que contiene un *Line Renderer* para poder renderizar la longitud del vector, y el segundo que contiene el modelo de flecha que está en el destino de cada vector y representa la dirección del mismo. Ambos son controlados por el script *AnimateLine.cs* el cual se encarga de cada vector independientemente de revisar primero si las marcas de Centro y Vector están detectadas, y en ese caso instanciar la línea y flecha dirigidas hacia la marca destino (Figura 3.5).

Una vez ya instanciado el vector en la aplicación, si se detecta algún movimiento en la marca (se revisa en cada frame si se detectó o no un movimiento) se actualizará automáticamente la longitud y dirección del vector, afectando también los cálculos mostrados en la interfaz gráfica.

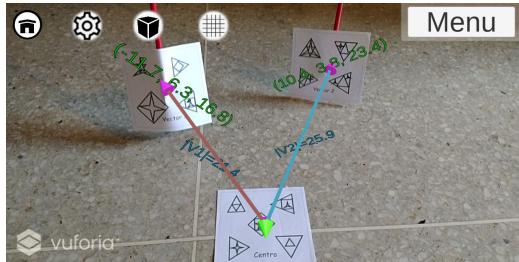


Figura 3.5: Representación de vectores con sus marcas en la aplicación.

3.1.3.1.2. Tracker El *Tracker* (o rastreador) se encarga de conectar las ejecuciones necesarias en la representación de vectores con el reconocedor de marcas de Vuforia. Específicamente, el *script Tracker.cs* llama a las distintas funciones de los otros scripts. Al reconocer Centro con Vector 1, se encarga de llamar a la función de representación de vector, y de igual manera al reconocer Centro con Vector 2. Al reconocer las tres marcas a la vez, llama a las funciones de operaciones que pueden estar activadas para poder representarlas en la aplicación.

3.1.3.1.3. Interfaz Gráfica (Componentes X,Y,Z y Módulo del Vector) El manejo de la interfaz gráfica se divide en cuatro *scripts*, los cuales son: *TextDistance.cs*, el cual está encargado de colocar el texto de la distancia entre los vectores y el centro, además de las posiciones en el espacio de ambos vectores; *PlaneHandler.cs*, el cual se encarga de activar y desactivar los planos de los ejes de forma rotativa; *SettingsHandler.cs*, que se encarga de cambiar el modo de renderizar los ejes de forma rotativa entre no eje, completo con parte negativa, normales y doblemente largos; finalmente está *BoxHandler.cs*, el cual maneja el renderizado una caja construida por las proyecciones del vector contra los ejes y rota entre renderizarlo para uno de los vectores.

3.1.3.2. Operaciones de vectores

Una vez detectada las marcas, el usuario podrá hacer uso del cálculo de las operaciones de vectores al presionar el botón “Menu”, las cuales se dividen en suma, resta, producto cruz, producto punto y limpiar. Todos los cálculos de las operaciones se realizan en el script *Functions.cs* el cual contiene las fórmulas necesarias para calcular cada una de ellas y mostrar gráficamente

el resultado con el vector resultante (compuesto por su longitud y dirección). Por último, solo se puede mostrar una operación a la vez.

3.1.3.2.1. Suma La operación de suma entre los vectores $u = (u_x, u_y, u_z)$ y $v = (v_x, v_y, v_z)$ en un plano de tres dimensiones viene representada por la fórmula $u + v = (u_x + v_x, u_y + v_y, u_z + v_z)$.

En la siguiente figura (Figura 3.6) se muestra un ejemplo de la operación sumar en funcionamiento.

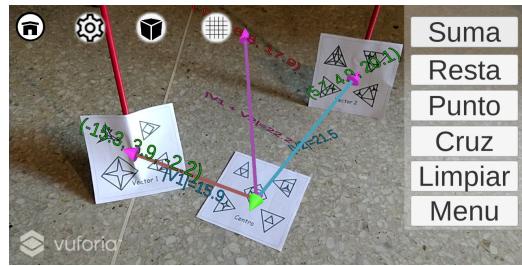


Figura 3.6: Operación sumar en la aplicación.

3.1.3.2.2. Resta La operación de resta entre los vectores $u = (u_x, u_y, u_z)$ y $v = (v_x, v_y, v_z)$ en un plano de tres dimensiones viene representada por la fórmula $u - v = (u_x - v_x, u_y - v_y, u_z - v_z)$.

En la siguiente figura (Figura 3.7) se muestra un ejemplo de la operación restar en funcionamiento.

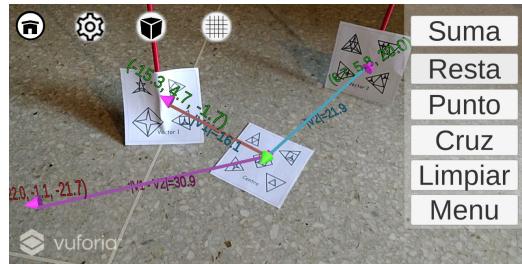


Figura 3.7: Operación restar en la aplicación.

3.1.3.2.3. Producto Punto La operación de producto punto entre los vectores $u = (u_x, u_y, u_z)$ y $v = (v_x, v_y, v_z)$ en un plano de tres dimensiones viene representada por la fórmula $u \cdot v = |u||v|\cos(u, v)$, en donde $|u| = \sqrt{u_x^2 + u_y^2 + u_z^2}$.

En la siguiente figura (Figura 3.8) se muestra un ejemplo de la operación producto punto en funcionamiento.

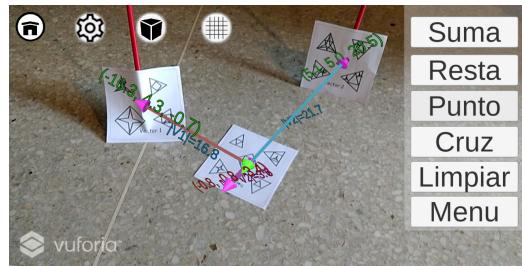


Figura 3.8: Operación producto punto en la aplicación.

3.1.3.2.4. Producto Cruz La operación de producto cruz entre los vectores $u = (u_x, u_y, u_z)$ y $v = (v_x, v_y, v_z)$ en un plano de tres dimensiones viene representada por la fórmula $u \times v = (u_y * v_z - u_z * v_y, u_z * v_x - u_x * v_z, u_x * v_y - u_y * v_x)$.

En la siguiente figura (Figura 3.9) se muestra un ejemplo de la operación producto cruz en funcionamiento.

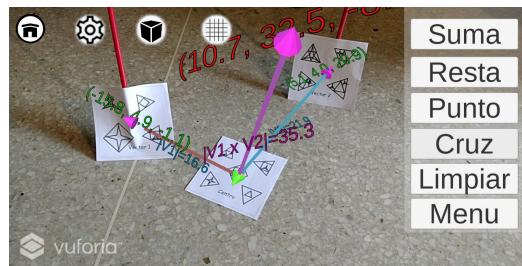


Figura 3.9: Operación producto cruz en la aplicación.

3.1.3.2.5. Limpiar La operación Limpiar se encarga de desactivar la función previamente activada y mostrar únicamente la interfaz original con la representación de los dos vectores.

3.1.3.3. Componentes adicionales

Por último, aparte de las operaciones de vectores en la interfaz gráfica, se encuentran cuatro botones: volver al menú principal, cambiar ejes, proyección de vectores y planos en el origen. Estas opciones se pueden activar independientes a las operaciones, es decir, no tienen restricciones entre ellas y se pueden ver todas activadas al mismo tiempo.

3.1.3.3.1. Ejes del origen El botón de ejes en el origen, permite al usuario colocar distintos tipos de ejes en las coordenadas (x, y, z) para facilitar la visualización del plano cartesiano. Estas operaciones son calculadas en el script *AnimateLineAxis.cs* que se encarga únicamente de instanciar cada uno de los ejes con su dirección determinada. Las distintas opciones que se encuentran son:

1. Desactivado.
2. Ejes positivos.
3. Ejes positivos y negativos (Figura 3.10).
4. Ejes positivos extendidos.

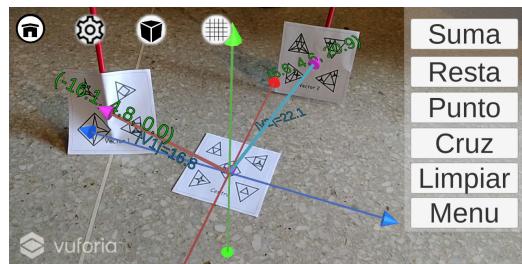


Figura 3.10: Opción de ejes positivos y negativos activada en la aplicación.

3.1.3.3.2. Proyección del Vector El botón de proyección de vectores permite representar la proyección de uno de los dos vectores de acuerdo a su origen y destino. Dependiendo de cuál vector se encuentre activado, además cambia el color del botón (verde para Vector 1 y azul para Vector 2). Estas operaciones se realizan en el script *AnimateAxis.cs* asociado a cada vector,

y se encarga de instanciar cada una de las líneas para finalmente proyectar el vector, representado en el plano de tres dimensiones como un cubo. Las distintas opciones que se encuentran son:

1. Desactivado.
2. Proyección del Vector 1 (Figura 3.11).
3. Proyección del Vector 2.

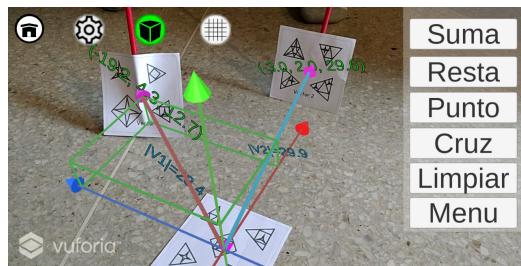


Figura 3.11: Opción de proyección del Vector 1 activada.

3.1.3.3.3. Planos en el Origen El botón de planos en el origen permite al usuario instanciar en la escena un plano cartesiano que facilita las mediciones de los vectores. Cada plano tiene una medida de 30×30 cuadrículas, cada una de aproximadamente $1 \times 1\text{cm}$ reales. Estos planos se instancian en el script *AnimateLineAxis.cs*. Las distintas opciones que se encuentran disponibles son:

1. Desactivado.
2. Plano en el eje x (Figura 3.12).
3. Plano en el eje y.
4. Plano en el eje z.
5. Planos en los ejes x,y,z (Figura 3.13).

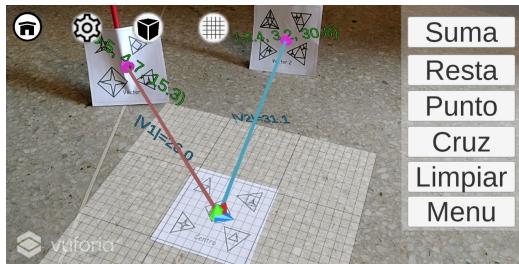


Figura 3.12: Opción de plano en el eje x activada.

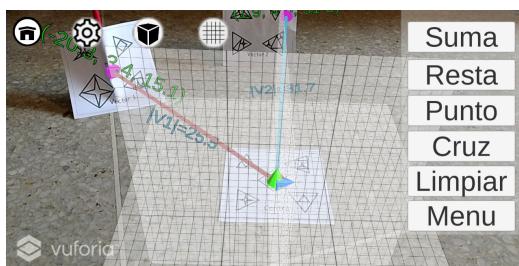


Figura 3.13: Opción de planos en los ejes x,y,z activada.

3.1.4. Consideraciones Importantes al Momento de Realizar Operaciones

En el editor de escenas de Unity, las unidades al momento de ser calculadas están definidas en Unidades de Vuforia, a diferencia de una unidad utilizada en la vida real como centímetros o pulgadas. Uno de los retos del proyecto fue encontrar cual era la relación entre unidades de Vuforia y centímetros para que los vectores tengan las mismas mediciones al ser utilizadas por la aplicación en la vida real. Luego de realizar varias pruebas, se encontró que utilizando marcas de detección de $10 \times 10\text{cm}$, se encuentra que la relación entre unidades de Vuforia y centímetros es de 1 : 1, es decir son totalmente equivalentes. Esto permite que la representación de los vectores tengan a lo sumo un margen de error de 1cm al momento de ser calculados por Vuforia y la aplicación.

3.2. VisualizAR: Gráficas

Esta sección de la aplicación se busca presentar los conceptos de gráficas a estudiantes de bachillerato, mediante la presentación interactiva de distintas funciones parametrizables con dos coordenadas, las cuales son manifestadas en 3D mediante el uso de la realidad aumentada. Además, se utiliza al tiempo como un tercer parámetro, para aumentar el llamativo de las gráficas. En las Figuras 3.14, 3.15 y 3.16 se pueden observar ejemplos de esta aplicación.

Para realizar esto se utilizaron dos *Image Render Target* de Vuforia, uno como base y otro como menú; un *Shader* para la coloración de la gráfica y cuatro *Scripts* de control.

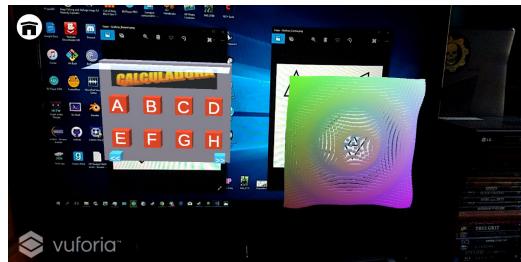


Figura 3.14: Ejemplo de la aplicación de Gráficas.

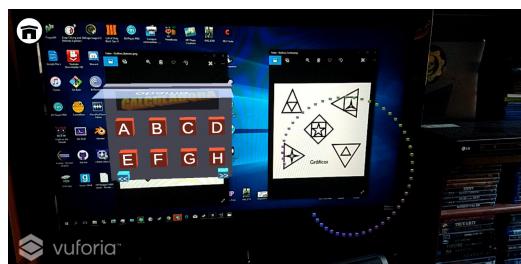


Figura 3.15: Ejemplo de la aplicación de Gráficas.

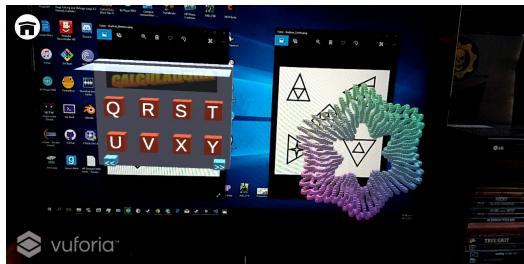


Figura 3.16: Ejemplo de la aplicación de Gráficas.

3.2.1. Image Render Target de Vuforia

Para poder utilizar el motor de realidad aumentada de Vuforia, se generaron dos *Image Render Target*. Estos fueron utilizados como la base donde poder mostrar las gráficas y los botones para controlar la selección de qué función visualizar.

En las Figuras 3.17 y 3.18 se puede observar las marcas utilizadas, en estas se observa que se agregaron los nombres bajo ellas para facilitar su uso por los usuarios.

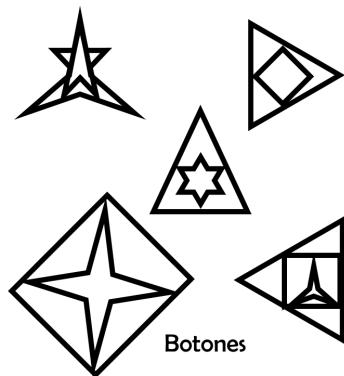


Figura 3.17: Marca de Botones para controlar la aplicación.

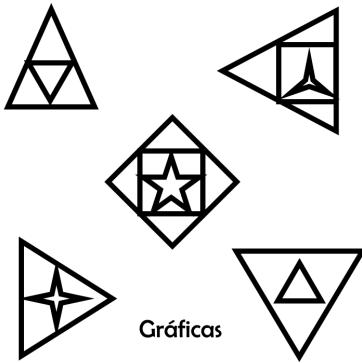


Figura 3.18: Marca de Gráficas para controlar la posición donde visualizar las funciones.

3.2.2. Shader

Para aumentar el llamativo visual de las gráficas, se creó un *Shader* para controlar la forma de colorear. Este se basa en mostrar la representación del espacio vectorial, centrado en base, en el espacio de colores RGB. Esto se realiza mediante la normalización del espacio, utilizando un factor de escalamiento, para pasar al espacio $[-1, 1] \times [-1, 1] \times [-1, 1]$ y luego transformarlo al espacio $[0, 1] \times [0, 1] \times [0, 1]$, el cual es representable como un color RGB. El código se puede encontrar en la carpeta de *Shaders* bajo el nombre *ColoredPoint.shader*. En la Figura 3.19 se observa un ejemplo de la coloración generada por el *Shader*.

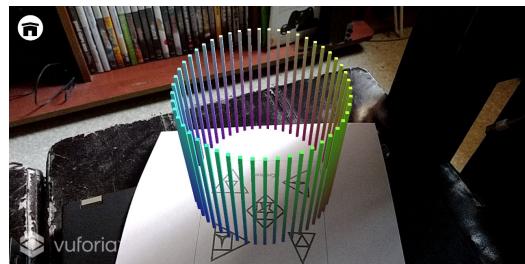


Figura 3.19: Ejemplo de la visualización del espacio vectorial mediante colores RGB.

3.2.3. Scripts

Los *Scripts* se pueden dividir en dos grupos, los que manejan las gráficas y el que maneja la interfaz.

3.2.3.1. Manejo de Gráficas

El proceso de graficar se realiza mediante la parametrización de superficies. En el caso de la aplicación, se parametrizan superficies 3D mediante coordenadas 2D del espacio $[0, 1] \times [0, 1]$, conocido en inglés como U V Mapping, el cual consiste en proyectar un plano 2D a una superficie 3D. Además, se incluye un parámetro de tiempo para darle un aspecto dinámico a las gráficas para atraer la atención del usuario. El desarrollo del manejador de gráficas se basó en las explicaciones de Jasper Flick [2], adaptado a los requerimientos específicos de la aplicación y con más funciones.

Las funciones utilizadas en la aplicación para generar las superficies visualizadas tienen la forma presentada en la Ecuación 3.1, es importante destacar que no siempre tienen que usar los tres parámetros de entrada.

$$\begin{aligned} [0, 1] \times [0, 1] \times \mathbb{R}^+ &\rightarrow \mathbb{R}^3 \\ f(u, v, t) = (x, y, z) \end{aligned} \tag{3.1}$$

Para representar los puntos de la superficie, se utilizó una malla 2D de 100×100 cubos (para darles un tamaño visible por los usuarios), la cual se instancia al comienzo de la aplicación de gráficas y se adapta dinámicamente a la función que se desea graficar. Un ejemplo para visualizar los cubos que forman la superficie, es la gráfica del círculo de la Figura 3.20.

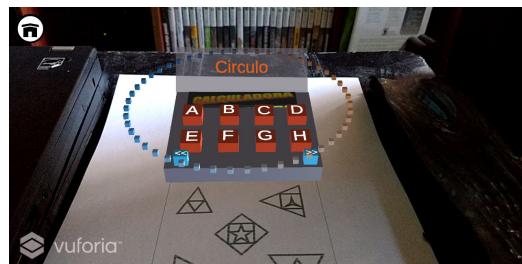


Figura 3.20: Gráfica del círculo donde se observan los cubos que forman la malla.

Todo esto es manejado por el *Script Graph.cs*, el cual, además, controla las funciones para graficar. Para facilitar desarrollo y la futura posible expansión, se agregaron: *GraphFunction.cs* el cual contiene el prototipo de las funciones que se grafican, para poder crear y utilizar más funciones; y *GraphFunctionName.cs* el cual contiene un enumerable de los nombres de las funciones, para poder visualizarlas mejor en los menús dentro del editor de Unity.

3.2.3.2. Calculadora

La calculadora se encarga de darle el control al usuario de decidir cuál gráfica mostrar al presionar cada uno de los botones sin necesidad de tener estas opciones en la interfaz gráfica. Por facilidad, se decidió tener tres páginas diferentes de opciones de ocho botones cada uno (identificados con una letra), las cuales pueden ser presionadas, emitiendo un sonido y animación; e inmediatamente cambiar la gráfica actual. Aparte, la calculadora muestra una pequeña pantalla transparente la cual contiene el nombre de la gráfica actual. En la Figura 3.21 se puede observar la calculadora como se ve en la aplicación.

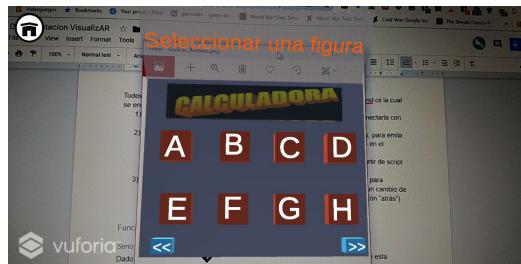


Figura 3.21: Calculadora como se ve en la aplicación.

Todos los procedimientos y funciones de la calculadora están calculados en *ARsound.cs* la cual se encarga de:

1. Inicializar la calculadora con la primera página y primera función, Seno y conectarla con el Manejador de Funciones.
2. Detectar cuando el usuario ha presionado alguno de los botones de gráficas, para emitir una animación de presionado de botón, un sonido y cambiar tanto la gráfica en el Manejador de Funciones como la pantalla

de la calculadora con el nombre correspondiente. Cabe resaltar que la pequeña animación es realizada a partir de script y no con el animador de Unity.

3. Detectar cuando el usuario ha presionado alguno de los cambios de página para actualizar la interfaz de la calculadora. En caso de que el usuario presione un cambio de página incorrecto (por ejemplo, estar en la primera página y presionar el botón “atrás”) se emite un sonido de error diferente al sonido predeterminado.

3.2.3.3. Funciones

La aplicación consta de veinticuatro funciones para graficar, estas son explicadas a continuación.

3.2.3.3.1. Seno Dado por la ecuación $f(u, v, t) = (u, \text{Sen}(\pi * (u + t)), v)$. La Figura 3.22 es la gráfica de esta función, la cual es la opción A del menú.

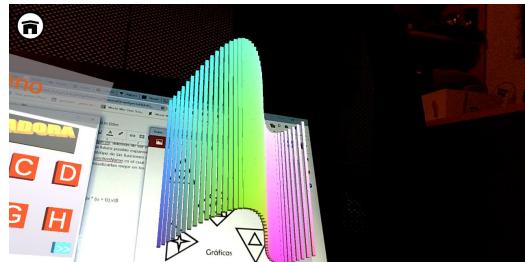


Figura 3.22: Gráfica de la función Seno.

3.2.3.3.2. Seno 2D Dado por la ecuación $f(u, v, t) = (u, \text{Sen}(\pi * (u + v + t)) * 0,5, v)$. La Figura 3.23 es la gráfica de esta función, la cual es la opción B del menú.

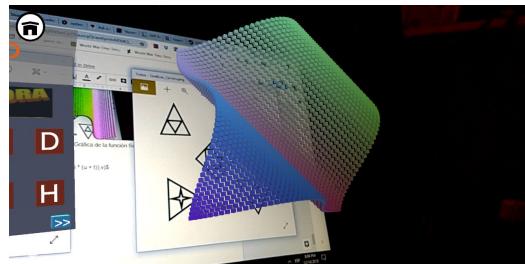


Figura 3.23: Gráfica de la función Seno 2D.

3.2.3.3.3. Media del Seno Dado por la ecuación $f(u, v, t) = (u, \text{Sen}(\pi*(u+t))/2 + \text{Sen}(\pi*(v+t))/2, v)$. La Figura 3.24 es la gráfica de esta función, la cual es la opción C del menú.



Figura 3.24: Gráfica de la función Media del Seno.

3.2.3.3.4. Multiseno Dado por la ecuación $f(u, v, t) = (u, \text{Sen}(\pi*(u+t))*2/3 + \text{Sen}(2*\pi*(v+2*t))/3, v)$. La Figura 3.25 es la gráfica de esta función, la cual es la opción D del menú.

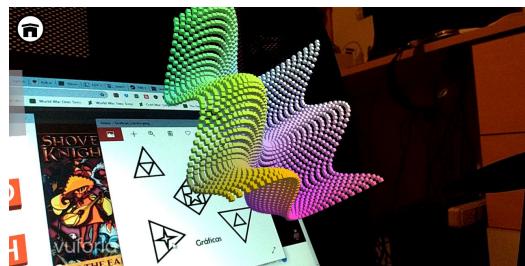


Figura 3.25: Gráfica de la función Mutiseno.

3.2.3.3.5. Multiseno 2D Dado por la ecuación $f(u, v, t) = (u, (4 * \text{Sen}(\pi * (u + v + t/2)) + \text{Sen}(\pi * (u + t)) + \text{Sen}(2 * \pi * (v + 2 * t)) * 0,5) * 1/5,5, v)$. La Figura 3.26 es la gráfica de esta función, la cual es la opción E del menú.

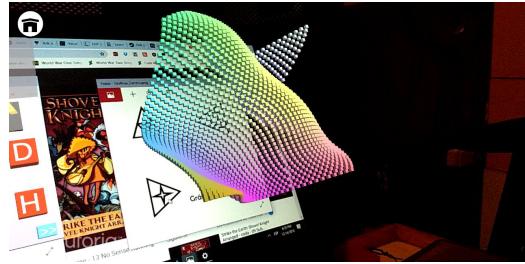


Figura 3.26: Gráfica de la función Mutiseno 2D.

3.2.3.3.6. Onda Dado por la ecuación $f(u, v, t) = (u, \text{Sen}(\pi * (4 * d - t))/(1 + 10 * d), v)$ donde $d = \sqrt{u * u + v * v}$. La Figura 3.27 es la gráfica de esta función, la cual es la opción F del menú.

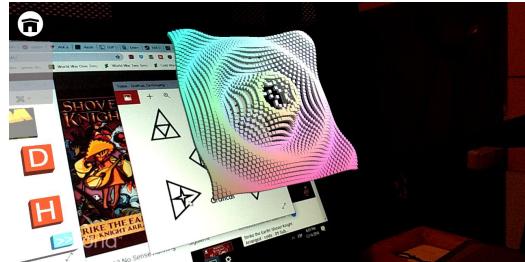


Figura 3.27: Gráfica de la función Onda.

3.2.3.3.7. Círculo Dado por la ecuación $f(u, v, t) = (\text{Sen}(\pi * u), 0, \text{Cos}(\pi * u))$. La Figura 3.28 es la gráfica de esta función, la cual es la opción G del menú.

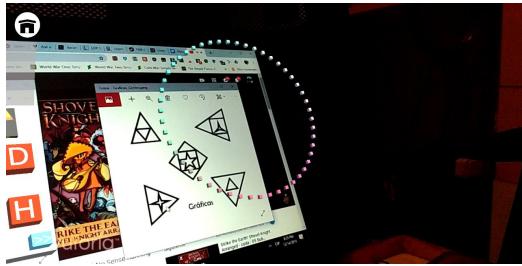


Figura 3.28: Gráfica de la función Círculo.

3.2.3.3.8. Remolino Dado por la ecuación $f(u, v, t) = (\operatorname{Sen}(\pi*w), u, \operatorname{Cos}(\pi*w))$ donde $w = u + t$. La Figura 3.29 es la gráfica de esta función, la cual es la opción H del menú.

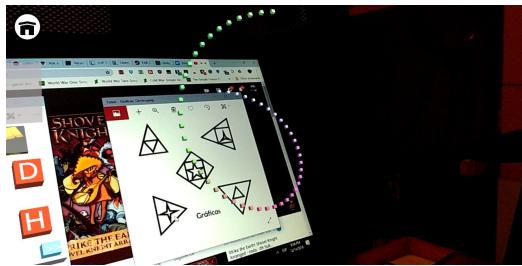


Figura 3.29: Gráfica de la función Remolino.

3.2.3.3.9. Cilindro Dado por la ecuación $f(u, v, t) = (r*\operatorname{Sen}(\pi*u), v, r*\operatorname{Cos}(\pi*u))$ donde $r = 1$. La Figura 3.30 es la gráfica de esta función, la cual es la opción I del menú.

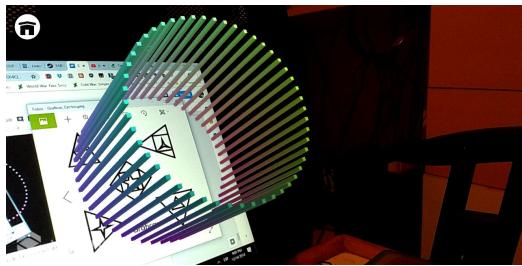


Figura 3.30: Gráfica de la función Cilindro.

3.2.3.3.10. Cilindro Tambaleante Dado por la ecuación $f(u, v, t) = (r * \text{Sen}(\pi * u), v, r * \text{Cos}(\pi * u))$ donde $r = 1 + \text{Sen}(6 * \text{pi} * u) * 0,2$. La Figura 3.31 es la gráfica de esta función, la cual es la opción J del menú.

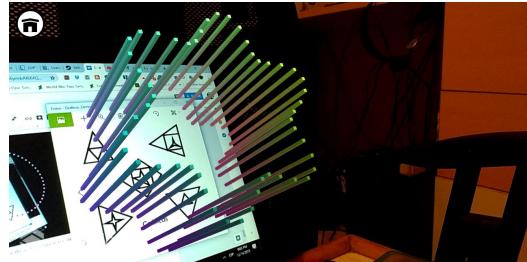


Figura 3.31: Gráfica de la función Cilindro Tambaleante.

3.2.3.3.11. Cilindro de Olla Dado por la ecuación $f(u, v, t) = (r * \text{Sen}(\pi * u), v, r * \text{Cos}(\pi * u))$ donde $r = 1 + \text{Sen}(2 * \text{pi} * v) * 0,2$. La Figura 3.32 es la gráfica de esta función, la cual es la opción K del menú.

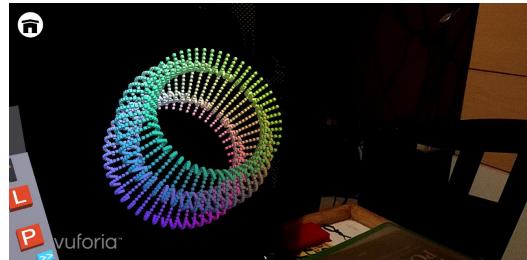


Figura 3.32: Gráfica de la función Cilindro de Olla.

3.2.3.3.12. Cilindro de Torsión Dado por la ecuación $f(u, v, t) = (r * \text{Sen}(\pi * u), v, r * \text{Cos}(\pi * u))$ donde $r = 0,8 + \text{Sen}(\pi * (6 * u + 2 * v + t)) * 0,2$. La Figura 3.33 es la gráfica de esta función, la cual es la opción L del menú.

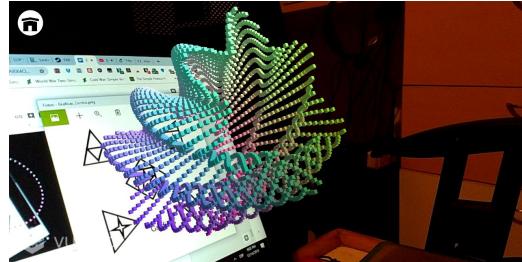


Figura 3.33: Gráfica de la función Cilindro de Torsión.

3.2.3.3.13. Esfera Dado por la ecuación $f(u, v, t) = (r * \text{Sen}(\pi * u), \text{Sen}(\pi * 0,5 * v), r * \text{Cos}(\pi * u))$ donde $r = \text{Cos}(\pi * 0,5 * v)$. La Figura 3.34 es la gráfica de esta función, la cual es la opción M del menú.

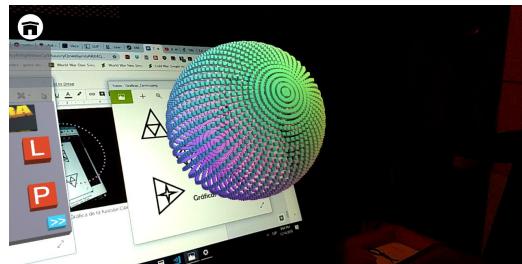


Figura 3.34: Gráfica de la función Esfera.

3.2.3.3.14. Esfera Cardioide Dado por la ecuación $f(u, v, t) = (r * \text{Sen}(\pi * u), v, r * \text{Cos}(\pi * u))$ donde $r = \text{Cos}(\pi * 0,5 * v)$. La Figura 3.35 es la gráfica de esta función, la cual es la opción N del menú.



Figura 3.35: Gráfica de la función Esfera Cardioide.

3.2.3.3.15. Esfera Pulsante Dado por la ecuación $f(u, v, t) = (s * \text{Sen}(\pi * u), r * \text{Sen}(\pi * 0,5 * v), s * \text{Cos}(\pi * u))$ donde $s = r * \text{Cos}(\pi * 0,5 * v)$ y $r = 0,8 + \text{Sen}(\pi * (6 * u + t)) * 0,1 + \text{Sen}(\pi * (4 * v + t)) * 0,1$. La Figura 3.36 es la gráfica de esta función, la cual es la opción O del menú.

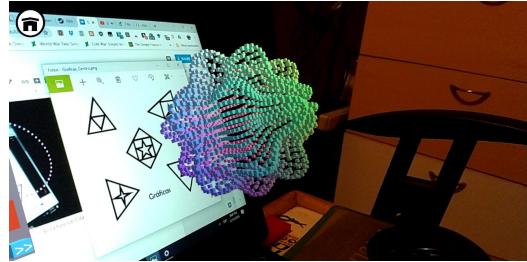


Figura 3.36: Gráfica de la función Esfera Pulsante.

3.2.3.3.16. Toroide Exterior Dado por la ecuación $f(u, v, t) = (s * \text{Sen}(\pi * u), \text{Sen}(\pi * 0,5 * v), s * \text{Cos}(\pi * u))$ donde $s = \text{Cos}(\pi * 0,5 * v) + 0,5$. La Figura 3.37 es la gráfica de esta función, la cual es la opción P del menú.



Figura 3.37: Gráfica de la función Toroide Exterior.

3.2.3.3.17. Toroide de Intersección Dado por la ecuación $f(u, v, t) = (s * \text{Sen}(\pi * u), \text{Sen}(\pi * v), s * \text{Cos}(\pi * u))$ donde $s = \text{Cos}(\pi * 0,5 * v) + 0,5$. La Figura 3.38 es la gráfica de esta función, la cual es la opción Q del menú.

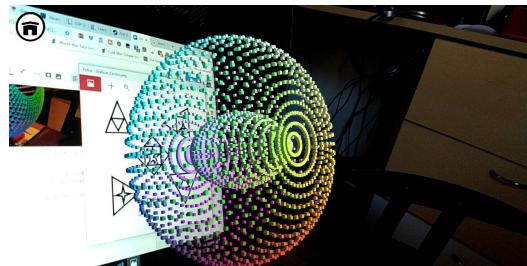


Figura 3.38: Gráfica de la función Toroide de Intersección.

3.2.3.3.18. Toroide de Cuerno Dado por la ecuación $f(u, v, t) = (s * \operatorname{Sen}(\pi * u), \operatorname{Sen}(\pi * v), s * \operatorname{Cos}(\pi * u))$ donde $s = \operatorname{Cos}(\pi * v) + r_1$ y $r_1 = 1$. La Figura 3.39 es la gráfica de esta función, la cual es la opción R del menú.

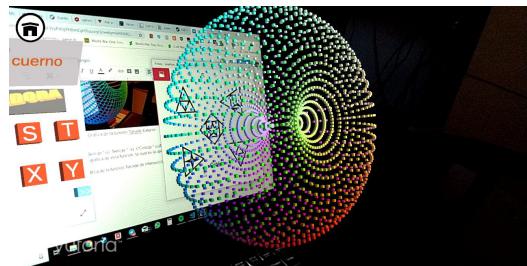


Figura 3.39: Gráfica de la función Toroide de Cuerno.

3.2.3.3.19. Toroide Anidado Dado por la ecuación $f(u, v, t) = (s * \operatorname{Sen}(\pi * u), r_2 * \operatorname{Sen}(\pi * v), s * \operatorname{Cos}(\pi * u))$ donde $s = \operatorname{Cos}(\pi * v)$, $r_1 = 1$ y $r_2 = 0,5$. La Figura 3.40 es la gráfica de esta función, la cual es la opción S del menú.

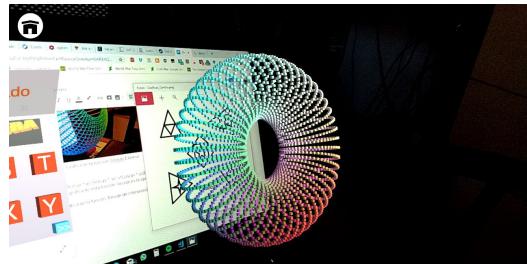


Figura 3.40: Gráfica de la función Toroide Anidado.

3.2.3.3.20. Toroide Estrella Dado por la ecuación $f(u, v, t) = (s * \operatorname{Sen}(\pi * u), r_2 * \operatorname{Sen}(\pi * v), s * \operatorname{Cos}(\pi * u))$ donde $s = \operatorname{Cos}(\pi * v)$, $r_1 = 10,65 + \operatorname{Sen}(\pi * (6 * u + t)) * 0,1$ y $r_2 = 0,2 + \operatorname{Sen}(\pi * (4 * v + t)) * 0,05$. La Figura 3.41 es la gráfica de esta función, la cual es la opción T del menú.

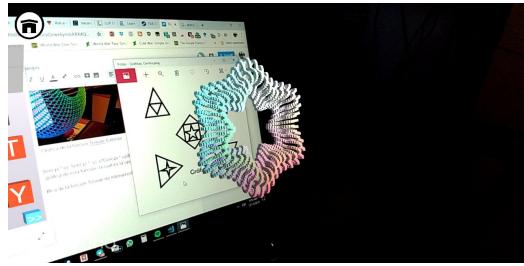


Figura 3.41: Gráfica de la función Toroide Estrella.

3.2.3.3.21. Banda de Möbius Dado por la ecuación $f(u, v, t) = (te * \operatorname{Cos}(nu), te * \operatorname{Sen}(nu), halfnv * \operatorname{Sen}(halfnu))$ donde $te = 1 + halfnv * \operatorname{Cos}(halfnu)$, $nu = 2 * halfnu$, $halfnv = v - 0,5$ y $halfnu = (u + t * 0,05) * \pi$. La Figura 3.42 es la gráfica de esta función, la cual es la opción U del menú.

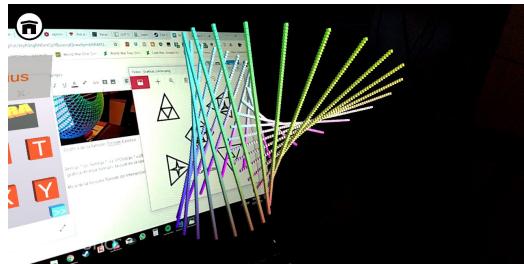


Figura 3.42: Gráfica de la función Banda de Möbius.

3.2.3.3.22. Banda Especial de Möbius Dado por la ecuación $f(u, v, t) = (te * \operatorname{Cos}(nu), te * \operatorname{Sen}(nu), halfnv * \operatorname{Sen}(halfnu))$ donde $te = 1 + halfnv * \operatorname{Cos}(halfnu)$, $nu = 2 * halfnu$, $halfnv = (v - 0,5) * \operatorname{Cos}(\pi * (t * 0,05 + v)) * \operatorname{Cos}(\pi * (t * 0,1f + u)) * \operatorname{Cos}(\pi * (t * 0,2 + u + v))$ y $halfnu = (u + t * 0,05) * \pi$. La Figura 3.43 es la gráfica de esta función, la cual es la opción V del menú.

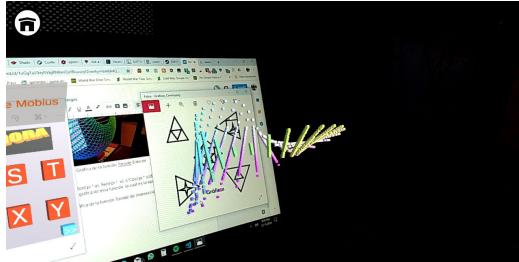


Figura 3.43: Gráfica de la función Banda Especial de Möbius.

3.2.3.3.23. Botella de Klein Estática Dado por la ecuación $f(u, v, t) = (-2/15*cu*(3*cv - 30*su + 90*cu^4*su - 60*cu^6*su + 5*cu*cv*su), -1/15*su*(3*cv - 3*cu^2*cv - 48*cu^4*cv + 48*cu^6*cv - 60*su + 5*cu*cv*su - 5*cu^3*cv*su - 80*cu^5*cv*sv + 80*cu^7*cv*su) - 2, 2/15*(3 + 5*cu*su)*sv)$ donde $sv = \text{Sen}(nv)$, $cv = \text{Cos}(nv)$, $su = \text{Sen}(nu)$, $cu = \text{Cos}(nu)$, $nv = v * \pi * 2$ y $nu = u * \pi$. La Figura 3.44 es la gráfica de esta función, la cual es la opción W del menú.

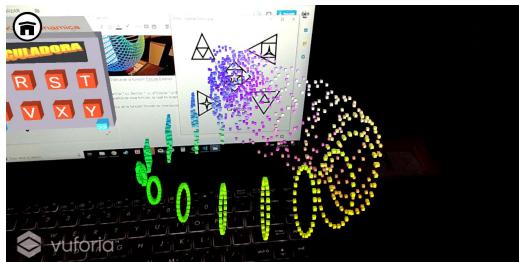


Figura 3.44: Gráfica de la función Botella de Klein Estática.

3.2.3.3.24. Botella de Klein Dinámica Dado por la ecuación $f(u, v, t) = (-2/15*cu*(3*cv - 30*su + 90*cu^4*su - 60*cu^6*su + 5*cu*cv*su), -1/15*su*(3*cv - 3*cu^2*cv - 48*cu^4*cv + 48*cu^6*cv - 60*su + 5*cu*cv*su - 5*cu^3*cv*su - 80*cu^5*cv*sv + 80*cu^7*cv*su) - 2, 2/15*(3 + 5*cu*su)*sv)$ donde $sv = \text{Sen}(nv)$, $cv = \text{Cos}(nv)$, $su = \text{Sen}(nu)$, $cu = \text{Cos}(nu)$, $nv = (v + t * 0,2) * \pi * 2$ y $nu = (u + t * 0,05) * \pi$. La Figura 3.45 es la gráfica de esta función, la cual es la opción X del menú.

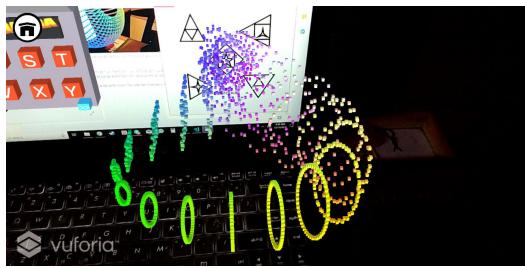


Figura 3.45: Gráfica de la función Botella de Klein Dinámica.

REFERENCIAS

- [1] Emory Craig and Maya Georgieva. From VR and AR to Our XR Future: Transforming Higher Education, Agosto 2018. Último Acceso: 2018-12-12, <https://er.educause.edu/blogs/2018/8/from-vr-and-ar-to-our-xr-future-transforming-higher-education>.
- [2] Jasper Flick. Mathematical Surfaces Sculpting with Numbers, 2017. Último Acceso: 2018-11-12, <https://catlikecoding.com/unity/tutorials/basics/mathematical-surfaces/>.
- [3] Michael Sano. Can AR/VR Improve Learning? Integrating Extended Reality Into Academic Programs #DLNchat, Julio 2018. Último Acceso: 2018-12-12, <https://www.edsurge.com/news/2018-07-19-can-ar-vr-improve-learning-integrating-extended-reality-into-academ>