UNIVERSIDAD POLITÉCNICA DE CHIAPAS

INGENIERÍA EN DESARROLLO DE SOFTWARE

MULTIMEDIA Y DISEÑO DIGITAL
GRUPO: 8A

U3ACT2:
REALIDAD AUMENTADA

ELABORADO POR:
DIEGO CARMONA BERNAL (213498)

PRESENTADO A:
ELIAS BELTRAN NATURI

TUXTLA GUTIÉRREZ, CHIAPAS A: 01 DE JULIO DE 2025

MarvelARApp v2.1 – Reporte de Realidad Aumentada

Introducción

En este proyecto se desarrolló **MarvelARApp**, una aplicación de Realidad Aumentada (RA) que superpone contenido digital temático de Marvel sobre marcadores visuales ArUco. El objetivo es utilizar la cámara web para detectar marcadores impresos y desplegar distintas experiencias AR: un escudo del Capitán América en 2D, una secuencia animada de Iron Man y un modelo 3D del martillo de Thor. Esta versión 2.1 implementa una corrección en la superposición 2D para alinear correctamente las imágenes con la perspectiva del marcador, mejorando la realismo de la experiencia.

Tecnologías y Herramientas Utilizadas

- **OpenCV (cv2)**: Biblioteca principal para procesamiento de imágenes, detección de marcadores ArUco y cálculos de proyección 3D. Se emplea OpenCV-Python ≥ 4.8 junto con el módulo cv2.aruco para la detección de marcadores.

- **Librería NumPy**: Usada para operaciones matemáticas y manejo eficiente de arrays (por ejemplo, cálculo de vectores, manejo de coordenadas y transformaciones).

- **Marcadores ArUco 6x6**: Se utilizaron marcadores del diccionario predefinido **DICT_6X6_250** de ArUco, que contiene 250 patrones únicos de 6x6 bits. Esto garantiza la detección fiable de nuestros marcadores específicos (IDs 4, 5 y 6).

- **Modelo 3D en formato OBJ**: El modelo del martillo de Thor fue obtenido de internet en formato .obj. Debido a su complejidad original (muchos vértices), se utilizó una herramienta en línea para **reducir su tamaño** (simplificar la malla) y así mejorar el rendimiento en tiempo real. En caso de que el modelo no esté disponible o sea muy pesado, la aplicación recurre a un modelo 3D *low-poly* por defecto (de baja complejidad) integrado en el código.

- **Hardware**: Cámara web para captura de video en vivo, donde se mostrarán los marcadores impresos y se visualizarán las superposiciones en pantalla.

Detección de Marcadores ArUco

La detección de marcadores es el primer paso crítico en la aplicación:

- Se crea un **diccionario ArUco** con aruco.getPredefinedDictionary(cv2.aruco.DICT_6X6_250) y parámetros por defecto de detección (aruco.DetectorParameters()).

- En cada cuadro capturado de la cámara, la imagen se convierte a escala de grises y se ejecuta aruco.detectMarkers, el cual devuelve las esquinas (cuatro vértices) de cada marcador detectado y sus IDs correspondientes.

- Si se detectan marcadores, la aplicación dibuja un recuadro alrededor de ellos (aruco.drawDetectedMarkers) para visualización y procede a identificar cada ID para decidir qué contenido superponer.

Para facilitar las pruebas, se incluyó una función de generación de marcadores (generate_aruco_markers) que crea imágenes PNG de los marcadores necesarios:

- **ID 4:** Marca para el escudo del Capitán América.

- **ID 5:** Marca para la animación de Iron Man.

- **ID 6:** Marca para el martillo de Thor.

Estas imágenes (ej. capitan_america_marker.png, iron_man_marker.png, thor_marker.png) se pueden imprimir o mostrar en pantalla para que la cámara las reconozca.

Superposición de Imagen 2D (Escudo del Capitán América)

Para el marcador con **ID 4**, la aplicación superpone la imagen 2D del **escudo del Capitán América** sobre el marcador detectado, logrando que parezca un escudo físico adherido al marcador. La implementación considera la perspectiva de la cámara:

- **Carga de imagen:** Al iniciar, se carga capitan_america_shield.png como imagen con canal alfa (transparencia). Si la imagen no se encuentra, el programa crea dinámicamente un **marcador de posición** con la palabra "ESCUDO" para no interrumpir la ejecución.

- **Transformación perspectiva:** Al detectar el marcador ID 4, se invoca la función overlay_image_2d(). Esta función toma las cuatro esquinas detectadas del marcador en la imagen de la cámara y calcula una **Transformación por Perspectiva** (cv2.getPerspectiveTransform). Dicha transformación mapea la imagen del escudo (cuadrada) exactamente a la forma cuadrilátera que el marcador ocupa en la escena, respetando la

orientación y distancia.

- **Overlay con alfa blending:** La imagen del escudo se transforma usando cv2.warpPerspective para que coincida con el área del marcador. Luego se combina con el cuadro de video usando el canal alfa para mezclar los bordes de forma suave. Esto asegura que partes transparentes (o los bordes) del PNG del escudo no cubran innecesariamente el fondo, integrándose de manera realista.

Superposición de Sprite Animado (Transformación de Iron Man)

El marcador con **ID 5** desencadena una animación inspirada en **Iron Man**, mostrando un *sprite* animado que simula la transformación o aparición de la armadura:

- **Sprite sheet:** El recurso iron_man_transformation.png contiene una hoja de sprites (sprite sheet) con 8 cuadros de animación organizados en una cuadrícula de 4 columnas x 2 filas. Al cargar esta imagen, se asegura también que tenga canal alfa (añadiéndolo si es necesario).

- **División de cuadros:** Mediante la función split_sprite(), la hoja de sprites se divide en 8 imágenes separadas (frames). Estas se almacenan en una lista iron_man_frames. En caso de que la imagen no esté disponible, la aplicación crea un conjunto de **imágenes de marcador de posición** con texto ("IRON MAN 1", "IRON MAN 2", etc.) en distintos colores para visualizar la secuencia de manera genérica.

- **Animación cíclica:** La clase mantiene un índice de frame (frame_index) y un contador de animación para controlar la velocidad (animation_speed = 3, lo que significa que se avanza de frame cada 3 iteraciones de cuadro aproximadamente). Cada vez que se procesa un cuadro de video con el marcador ID 5 presente, se elige el frame correspondiente de iron_man_frames según frame_index y se superpone usando la función overlay_animated_sprite(). Esta función interna simplemente selecciona el frame actual y llama a overlay_image_2d para realizar la superposición con perspectiva correcta, igual que con el escudo.

- **Reinicio y control:** Si el usuario pulsa la tecla **R** durante la ejecución, el índice de animación se resetea a 0, reiniciando la animación de Iron Man desde el primer frame. Esto es útil para volver a ver la secuencia desde el inicio sin reiniciar la aplicación.

Renderizado de Modelo 3D (Martillo de Thor)

El marcador con **ID 6** activa la superposición de un **modelo 3D**: el mítico martillo de Thor (*Mjölnir*) que aparece renderizado como si estuviera posado sobre el marcador. Esta parte combina técnicas de visión por computadora para estimar la pose 3D del marcador y gráficos 3D básicos para dibujar el objeto:

- **Carga del modelo 3D:** Al iniciar, se intenta leer el archivo thor_hammer.obj que contiene la malla 3D del martillo. El código parsea vértices (v) y caras (f) del OBJ. Para optimizar:

    - Se recentra la nube de vértices al origen (restando la media) y se escala uniformemente para que el modelo quepa en un tamaño estándar.

    - Se aplica una rotación correctiva (Rx) a los vértices para alinear el modelo con el sistema de coordenadas de cámara de OpenCV (ajustando ejes debido a diferencias de convención).

    - Adicionalmente, cada cara poligonal se triangula (si la cara tenía más de 3 vértices, se divide en triángulos) porque es más sencillo dibujar triángulos.

    - Si el archivo no existe o no contiene datos válidos (por ejemplo, si la compresión u obtención falló), se recurre a create_default_mjolnir_model(), que define un martillo simplificado mediante unos cuantos vértices y caras predefinidas (forma rectangular para la cabeza del martillo y un eje para el mango). Esto garantiza que siempre haya algo que dibujar.

- **Estimación de pose con solvePnP:** Cuando se detecta el marcador ID 6 en un cuadro, el programa calcula la posición y orientación (pose) del marcador relativo a la cámara.

    - Se definen cuatro puntos 3D correspondientes a los bordes del marcador en unidades reales (por ejemplo, un cuadrado de 2 unidades de lado en el plano *z=0*).

    - Usando las esquinas 2D detectadas del marcador en la imagen y los puntos 3D conocidos, se resuelve la pose con cv2.solvePnP. El resultado es un vector de rotación (rvec) y de traslación (tvec) que transforman del sistema de coordenadas del marcador al de la cámara.

    - Con rvec y tvec, OpenCV puede proyectar cualquier punto 3D del modelo a coordenadas 2D de la imagen (usando cv2.projectPoints), dado también una matriz de cámara. En esta aplicación, la **matriz de cámara** se calibró de manera simplificada: se asume una focal aproximada igual al ancho de la imagen y punto principal al centro (derivado de la resolución de la cámara). Aunque no es una calibración real, funciona suficientemente para la demostración.

- **Dibujado del modelo:** Con los puntos proyectados en 2D:

    - Se calculan las **normales de las caras** del modelo 3D (ya precomputadas al cargar el modelo).

    - Se implementa un *pintor simple 3D*: se estima la profundidad promedio de cada triángulo de la malla en la cámara y se ordenan los triángulos de más lejano a más cercano. Luego se dibujan rellenando polígonos (cv2.fillConvexPoly) en ese orden para que las caras frontales tapen a las traseras, simulando oclusión correcta.

    - Se aplica un **sombreado plano (flat shading)** básico: para cada triángulo, se calcula la intensidad de luz según el ángulo de la normal con una fuente de luz fija. En este caso, la luz se define proveniente de la cámara (vector [0,0,1]). Se obtiene la intensidad como el coseno del ángulo entre la normal y la luz (producto punto), limitando a valores ≥ 0. Los triángulos se colorean con una base gris claro, y se oscurecen o aclaran ligeramente con un tinte azulado dependiendo de esa intensidad, dando una impresión de tridimensionalidad.

    - **Textura adicional:** Si existe el archivo runes.png (una imagen de runas o adornos), el programa la superpone a escala sobre un lado del martillo (por ejemplo, grabados en la cabeza de Mjölnir) para mayor efecto visual. Esto se realiza encontrando el centro de una de las caras principales proyectadas y dibujando la pequeña imagen de runas con transparencia sobre esa zona.

Flujo de Ejecución de la Aplicación

A continuación se resume el flujo principal (run()) de MarvelARApp v2.1:

1. **Inicialización de video:** Se activa la captura de video de la cámara web (cv2.VideoCapture(0)). Si no se puede acceder a la cámara, el programa lanza un error. Se obtienen la resolución (w x h) de la imagen para configurar

la matriz intrínseca de la cámara.

2. **Parámetros iniciales:** Se imprimen en consola datos de referencia (resolución detectada y la asignación de IDs a contenidos: “Marcadores: 4=Escudo, 5=IronMan, 6=Mjölnir”). También se muestra la ventana de visualización de la cámara con título "Marvel AR".

3. **Bucle de procesamiento de cuadros:** Por cada cuadro capturado:

    - Se detectan marcadores ArUco presentes.

    - Por cada marcador detectado, según su ID, se llama a la función correspondiente de superposición:

        - **ID 4:** overlay_image_2d para dibujar el escudo del Capitán América en la posición del marcador.

        - **ID 5:** overlay_animated_sprite para dibujar el frame actual de la animación de Iron Man.

        - **ID 6:** draw_3d_object para renderizar el martillo de Thor en 3D sobre el marcador.

    - Si no hay marcadores conocidos en el cuadro, la imagen se muestra sin alteraciones (o solo con recuadros si detectó otros marcadores).

    - Se actualiza el contador de animación para Iron Man: cada iteración incrementa un contador, y cuando alcanza el valor animation_speed, avanza al siguiente frame (esto hace que la animación sea más lenta que la tasa de cuadros para apreciarla bien).

    - Se presenta el cuadro resultante en la ventana.

4. **Interacción usuario:** El programa escucha teclas:

    - **ESC:** Cierra la aplicación y sale del bucle.

    - **R:** Reinicia la animación de Iron Man (pone frame_index = 0). Útil para volver a ver la animación desde el inicio sin reiniciar todo.

    - **S:** Toma una captura de pantalla guardando la imagen actual con un nombre de archivo timestamp (por ejemplo "shot_20250804_000123_00.png"). Esto permite documentar resultados fácilmente (como las figuras presentadas en este reporte).

Cada pulsación de S incrementa un contador para no sobrescribir archivos.

5. **Finalización:** Al salir del bucle (por ESC o error), se libera la cámara (cap.release()) y se destruyen las ventanas de OpenCV (cv2.destroyAllWindows()).

Este flujo asegura que la aplicación funcione en tiempo real, detectando múltiples marcadores simultáneamente si aparecen, y superponiendo cada contenido en el lugar correcto. El diseño modular con funciones separadas para cada tipo de overlay facilita la legibilidad y mantenimiento del código.

Desafíos Encontrados y Soluciones Implementadas

Durante el desarrollo de MarvelARApp v2.1 se presentaron varios desafíos técnicos, junto con las soluciones adoptadas para resolverlos:

- **Alineación correcta de imágenes 2D:** En versiones anteriores, las imágenes 2D (como el escudo) simplemente se dibujaban sobre el centro del marcador, sin considerar la perspectiva, lo que resultaba poco realista cuando la cámara o el marcador se inclinaban. **Solución:** Implementar la transformación por perspectiva en overlay_image_2d. Ahora la imagen se deforma adecuadamente para coincidir con la orientación del marcador, corrigiendo el problema.

- **Manejo de transparencia:** Al combinar imágenes con canal alfa sobre el video, hubo que asegurarse de aplicar una composición alfa correcta. Se utilizó una fórmula de *alpha blending* por píxel: output = background * (1 - alpha) + foreground * alpha. Esto evita bordes duros y permite que solo el escudo (o sprite) visible tape el video, manteniendo transparente el resto.

- **Rendimiento con modelo 3D pesado:** El modelo original del martillo de Thor tenía demasiados polígonos, lo cual hacía lento el dibujado en cada cuadro. **Solución:** Se redujo la complejidad del modelo usando herramientas en línea de simplificación de mallas (reducción de vértices y caras). Además, se implementó una alternativa de respaldo *low-poly* integrada en el código para asegurar rendimiento en cualquier caso.

- **Calibración de cámara:** Idealmente, la proyección 3D requiere parámetros intrínsecos de la cámara calibrados (focales, distorsión, etc.). Aquí no se realizó una calibración física; en su lugar se estimó la matriz de cámara asumiendo una focal aproximada según el ancho de la imagen y sin distorsión (dist_coeffs se dejó en cero). Aunque no es perfecto, resultó

suficiente para mantener el modelo 3D estable sobre el marcador en pruebas casuales. En entornos profesionales, se recomendaría una calibración adecuada para mayor precisión.

- **Uso de recursos externos y asistencia:** Algunas partes del desarrollo, como la corrección de la superposición 2D y optimizaciones, fueron realizadas con apoyo de recursos externos. Por ejemplo, se consultó documentación de OpenCV y buenas prácticas de implementación.

Resultados y Observaciones

El resultado final es una aplicación de RA funcional que reconoce los tres marcadores y despliega correctamente cada elemento multimedia de Marvel:

- Al mostrar el **marcador 4**, en la pantalla aparece el escudo del Capitán América perfectamente superpuesto, siguiendo la posición y rotación del marcador. La imagen permanece estable y del tamaño adecuado incluso si el marcador se mueve o rota, demostrando la efectividad de la transformación por perspectiva.

- Con el **marcador 5**, la aplicación muestra a Iron Man “apareciendo” sobre el marcador a través de una secuencia de imágenes. La animación se reproduce en bucle mientras el marcador esté visible, y se reinicia según se requiera. Los cuadros de la animación también respetan la orientación del marcador, creando el efecto de que Iron Man emerge desde ese patrón físico.

- En presencia del **marcador 6**, se renderiza el martillo de Thor en 3D. A simple vista, se puede observar cómo el martillo parece un objeto tridimensional real: las caras cambian de iluminación conforme se mueve la cámara (debido al sombreado plano aplicado), y el objeto mantiene su lugar sobre el marcador, incluso cuando se inclina la tarjeta. Esto añade un gran impacto visual, pues da la impresión de tener el martillo realmente frente a la cámara.

- La aplicación corrió en tiempo real a una velocidad razonable (dependiendo del hardware de la cámara y PC). Tras la simplificación del modelo 3D, no se notaron caídas significativas de cuadros por segundo. Todos los overlays (2D, animación y 3D) se mostraron simultáneamente sin problemas si múltiples marcadores estaban en la vista al mismo tiempo.

- Se grabó un **video demostrativo** (adjunto por separado) que evidencia el funcionamiento en vivo: en él se muestran los tres marcadores uno por uno frente a la cámara y cómo la aplicación responde instantáneamente

mostrando el contenido RA correspondiente. Este video sirve para validar el logro de los objetivos del proyecto.

Conclusiones

MarvelARApp v2.1 cumplió con éxito el objetivo de combinar el seguimiento de marcadores ArUco con la superposición de contenido 2D, animado y 3D de temática Marvel. A través del proyecto, se integraron conocimientos de visión por computador (detección de patrones, pose 3D) con técnicas de gráficos (manipulación de imágenes, animación cuadro a cuadro, renderizado de modelos con sombreado básico).

Los principales logros incluyen:

- **Realismo mejorado** en la superposición 2D gracias a la corrección de perspectiva.

- **Animación interactiva** implementada de forma sencilla y reiniciable, añadiendo dinamismo a la experiencia RA.

- **Integración 3D** efectiva, mostrando que incluso con métodos sencillos es posible visualizar objetos tridimensionales alineados con el mundo real.

En cuanto a aprendizaje, el proyecto permitió experimentar con la biblioteca OpenCV en profundidad, entender la importancia de calibración y optimización de recursos, así como resolver problemas prácticos (como la disminución de complejidad de modelos) para mantener la aplicación en tiempo real.

En resumen, la aplicación MarvelARApp v2.1 sirve como una demostración sólida de Realidad Aumentada basada en marcadores, proporcionando una experiencia lúdica y educativa. A futuro, se podría expandir agregando más marcadores con distintos objetos/personajes, incorporando interacción del usuario (por ejemplo, que al hacer clic o alguna señal se reproduzca un sonido o efecto), o migrando a bibliotecas/motores más avanzados si se busca mayor realismo en los modelos 3D. Las bases sentadas en este proyecto facilitan esas extensiones y muestran un camino claro para seguir desarrollando aplicaciones de RA.