# Visualización de Radar: Implementación en Software y Aceleración por Hardware

Jordan D. Carranza Benitez[4] Cristian A. Silva[4] Juan R. U. Machado[4] Diego M. Martínez[2] Christian L. Galasso[1,2,3] Nicolás A. Fernández Pavesi[2] Luis E. Gomed[2]

[1] Universidad de la Defensa Nacional – FADARA – ESOA, Punta Alta, Argentina
[2] Servicio de Análisis Operativos, Armas y Guerra Electrónica de la Armada, Punta Alta, Argentina
[3] Universidad Tecnológica Nacional – FRBB, Bahía Blanca, Argentina
[4] Universidad Nacional del Sur, Bahía Blanca, Argentina
jordancarranza045@gmail.com, sivacristian29@gmail.com, jrumachado@gmail.com, dmmartinez7@gmail.com, clgalasso@frbb.utn.edu.ar,nicopavesi@hotmail.com, gomedluis@hotmail.com

**Abstract.** Este trabajo presenta un sistema integrado para la visualización de información radar codificada bajo el protocolo ASTERIX CAT240 de Eurocontrol[1], que utiliza un convertidor analógico-digital de 14 bits para la captura y procesamiento de la señal de Video Crudo Radar. El sistema implementado proporciona una solución completa cuyo alcance abarca desde la captura de datos radar hasta su representación visual en tiempo real, utilizando tecnologías modernas como C++ y OpenGL [2].

La arquitectura del software, desarrollada bajo el paradigma de programación orientada a objetos, permite la traducción eficiente de tramas ASTERIX CAT240 a través de un módulo decodificador dedicado en un hilo de CPU. El decodificador extrae información relevante de las tramas de video radar, procesando parámetros de azimut, rango y niveles de intensidad. Adicionalmente, se implementa un algoritmo especializado para la detección y manejo de datos históricos, abordando la problemática del corrimiento inherente al funcionamiento del radar que no cubre todos los grados en cada giro.

El componente central de visualización implementa diversos algoritmos de procesamiento de imágenes que operan directamente en la GPU, permitiendo ajustes en tiempo real de brillo, contraste y umbrales. La implementación utiliza shaders GLSL para aplicar transformaciones avanzadas a los datos de intensidad, incluyendo corrección gamma, mapeo logarítmico, funciones sigmoidales y técnicas de histograma adaptativo para la mejora de contraste. El renderizador desarrollado optimiza la representación de grandes volúmenes de datos radar mediante estructuras eficientes de OpenGL como Vertex Buffer Objects.

**Keywords:** ASTERIX CAT240, Radar, OpenGL, Procesamiento de imágenes, Conversor analógico-digital, GPU, Shaders, Tiempo real.

# 1 Introducción

A 26 km de Bahía Blanca se encuentra el puerto militar más importante de la república Argentina, la Base Naval Puerto Belgrano. Dicho puerto es el lugar de asiento de la Flota de Mar, que cuenta con una gran cantidad y variedad de barcos destinados a la custodia del mar argentino y a la salvaguarda de la vida en el mar. Los mismos están dotados de diversos sistemas electrónicos, eléctricos, informáticos, y mecánicos; para la navegación y vigilancia. Un grupo de estas unidades tiene radares de vigilancia de tecnología de los años 70, y si bien tienen más de 40 años de servicio, los mismos siguen en capacidad de mantenerse operativos varios años más. Ahora bien, dado el avance de la tecnología de redes digitales de datos, y de representación de la información, es deseable desarrollar una nueva visualización digital de la información provista por dichos radares. Además que la digitalización puede permitir la distribución de las imágenes radar en tiempo real a muchos lugares dentro del barco y a una gran variedad de dispositivos (PC, TV, Tablets, otros).

Para esto, se realizó un desarrollo en software utilizando el lenguaje de C++ con el framework Qt y OpenGl para la representación visual del radar, permitiendo un mejor manejo de los recursos y un funcionamiento optimizado.
La representación visual de radar en una computadora significaba resolver 4 problemas:

1. Decodificar la trama ASTERIX CAT 240.
2. Guardar esa información en memoria para actualizar la imagen.
3. Renderizar la información con OpenGL.
4. Procesar las celdas de video para obtener información útil.

Se buscó resolver la visualización de tres tipos de radares de diferentes alcances y velocidades de giro de antena. La unidad de medida utilizada para el rango de los radares de DM (Milla de Datos). Una milla de datos es igual a 6000 pies, o 0.987 millas náuticas [6].

Los radares seleccionables se conocen como:

- RAD1: De corto alcance, giro rápido
- RAD2: Alcance mediano.
- RAD3: Alcance largo, giro lento.

La consola posee un rango máximo de 256 DM. Esto no significa que los radares instalados en la unidad lleguen al rango máximo. Antes de entrar al sistema, los datos del radar son digitalizados y codificados bajo el protocolo ASTERIX CAT 240 versión 1.3 de EuroControl con una profundidad de 14 bits [4]. Una vez codificados llegan al sistema a través de una red designada para compartir los datos de radar.

# 2 Protocolo ASTERIX

El protocolo ASTERIX (All Purpose Structured EUROCONTROL Surveillance Information Exchange) es un estándar desarrollado por EUROCONTROL para el

intercambio de información de vigilancia aérea. Dentro de este estándar, la Categoría 240 (CAT 240) se utiliza para la transmisión de video de radar rotativo hacia displays de mantenimiento locales o remotos. Esta categoría permite la codificación y transmisión eficiente de datos de video de radar, facilitando su integración y visualización en sistemas de control de tráfico aéreo [1].

La transmisión de información de video radar se basa en la transmisión de dos tipos de mensajes: Resumen de video y Celdas de video. El primer tipo de mensaje es metadata, y provee información sobre los mensajes a recibir: su longitud, resolución, compresión y volumen de datos. El segundo tipo de mensajes son los datos de intensidad de las celdas que transmite el radar.

El sistema posee un módulo que es capaz de recibir las tramas ASTERIX en UDP de la red y decodificarlas en tiempo real para llenar una estructura de datos mantenida en memoria.

## 2.1 Interpretación del Protocolo

ASTERIX Categoría 240 se utiliza para la transmisión de datos de video de radar en sistemas de vigilancia aérea. Cada mensaje incluye uno o más Data Items, estructurados según el User Application Profile (UAP). Los campos de interés para el sistema se detallan a continuación [5].

### 2.1.1 I240/000 – Message Type

- **Propósito**: Identifica el tipo de mensaje.
- **Detalle**: Indica si el mensaje es un *Video Summary* (tipo 001) o un *Video Message* (tipo 002). Facilita el procesamiento del mensaje por parte del receptor.
- **Formato**: 1 octeto.

### 2.1.2 I240/020 – Video Record Header

- **Propósito**: Proporciona un identificador de secuencia.
- **Detalle**: Contiene un contador cíclico de 32 bits que permite detectar mensajes perdidos o fuera de orden.
- **Formato**: 4 octetos.
- **Presente solo en mensajes de video**.

### 2.1.3 I240/041 – Video Header Femto

- **Propósito**: Igual que el anterior, pero con mayor resolución temporal.
- **Detalle**: Usa femtosegundos como unidad para la duración de la celda.
- **Formato**: 12 octetos.
- **Se usa en lugar de I240/040 si se requiere precisión temporal más alta.**

2.1.4 I240/048 – Video Cells Resolution & Data Compression Indicator

- **Propósito**: Indica la resolución de cuantificación del video y si se aplicó compresión.
- **Detalle**: Define cuántos bits representan la amplitud de cada celda (de 1 a 32 bits), y si se usó un algoritmo de compresión.
- **Formato**: 2 octetos.
- **Crucial para la decodificación de los datos de video.**

2.1.5 I240/051 – Video Block Medium Data Volume

- **Propósito**: Transmite bloques de celdas de tamaño intermedio.
- **Detalle**: Cada bloque tiene 64 octetos. Permite mayor volumen que I240/050, con eficiencia razonable.
- **Formato**: Variable.

# 3 Representación lógica en memoria

El sistema de software que permite visualizar el radar debe guardar en memoria principal los datos de video contenidos en la trama UDP codificada. Para esto se decidió utilizar una matriz de tamaño AZIMUT × RANGE (filas y columnas, respectivamente) implementada como un vector de vectores alojado de manera secuencial en la memoria al inicio del programa. Cada celda de la matriz alberga un número entero no signado de 16 bits, debido a la resolución especificada en el protocolo (*Very High Resolution*).

El protocolo de transmisión de video establece un valor máximo de 32 bits para el RANGE. Por otro lado, los radares utilizados poseen 8192 muestras para *azimuth*. El tamaño total para la supuesta matriz es de:

$$\text{RANGE\_MAX} * \text{AZIMUTH\_MAX} * \text{BYTES\_PER\_CELL} = \text{MAX\_SIZE} \quad (1)$$

$$2^{32} * 2^{13} * 2 = 70{,}3 \text{ [TB]} \quad (2)$$

Este tamaño es absurdo, por lo que debemos ajustar el tamaño de memoria a alojar según los radares utilizados y las restricciones que posee el sistema. Es posible calcular la distancia máxima de cada radar utilizando la cantidad de muestras de START RG recibidas antes de volver a 0 y el valor de CELL DUR. Se plantean algunos valores típicos para poder presentar un ejemplo de cálculo:

- CELL DUR = $6{,}25 * 10^{-8}$
- MAX RANGE = 4336

Luego, utilizando la fórmula provista por el protocolo:

$$d_{max} = \text{CELL\_DUR} * \text{MAX\_RANGE} * c / 2 \quad (3)$$

$$d_{max} = 6{,}25 * 10^{-8} * 4336 * 2{,}99792458 * 10^{8} / 2 \quad (4)$$

Y la distancia máxima para el radar del ejemplo

$$d_{max} = 40.621,87 \text{ [Metros]}$$

Este dato nos permite mapear muestras de rango a distancias absolutas. Para este caso, sabemos que una muestra de rango de 4336 equivale a 40.621,87 metros, o 22,209 millas de datos (DM), que es la unidad de medida utilizada por el sistema. Ahora, sabiendo que el sistema posee un rango de visualización máximo de 256 DM, obtenemos una cota superior a la cantidad de muestras que debemos representar en la matriz.

$$256 \text{ DM} = 468.224 \text{ metros} = 32.688 \text{ [muestras]} \quad (5)$$

Vistos los valores obtenidos se proyecta entonces que utilizando una matriz de 8.192 ×15.580, que convenientemente entre dentro de los límites de OpenGL para la creación de texturas [2] se podrán obtener representaciones de distancias razonables.

## 4 Algoritmo para la detección y manejo de datos históricos

El radar genera nueva información a través del movimiento rotatorio (barrido) que realiza junto con rápidas sucesiones de transmisión y recepción. Posee señales de control para disparar pulsos electromagnéticos y luego detectar los rebotes sobre los que calcula posibles puntos de interés en el terreno. Esto presenta una particularidad técnica importante: al completar una rotación completa, el siguiente giro no necesariamente comienza desde el mismo ángulo, fenómeno que llamaremos corrimiento angular.

En los sistemas radar tradicionales, que utilizaban tubos de rayos catódicos (CRT) como medio de visualización, este corrimiento no presentaba mayores inconvenientes. En este tipo de visualizadores la persistencia fosfórica de la imagen es corta, lo que produce una renovación constante de la imagen. De esta manera, se garantiza visualmente una imagen coherente y actualizada en cada instante.

En los sistemas modernos, el comportamiento es distinto. La visualización de la información radar se realiza mediante renderizado digital, línea por línea, a medida que se recibe la información proveniente de las tramas ASTERIX. Esto implica que los datos visualizados permanecen en pantalla de forma persistente, salvo que se sobrescriban con nueva información correspondiente al mismo ángulo o sector. Dado el corrimiento angular, no se puede garantizar que cada nueva vuelta del radar emita exactamente en los mismos ángulos que la anterior. Como consecuencia, puede quedar información antigua en pantalla que no representa el estado actual del espacio supervisado.

Para abordar este problema, se desarrolló un algoritmo específico para la detección y gestión de datos históricos. El objetivo principal del algoritmo es identificar y eliminar aquellas trazas o líneas antiguas que ya no reflejan información útil del entorno.

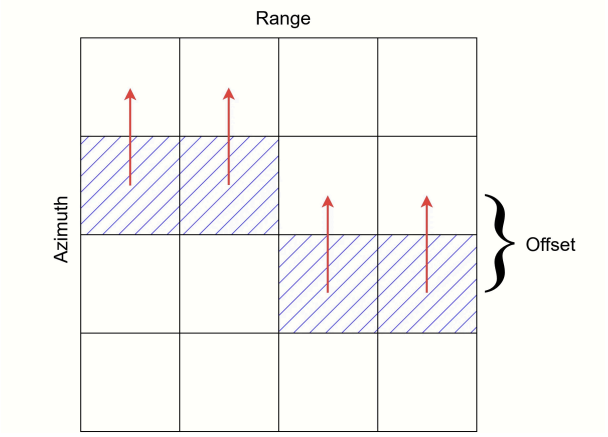


**Fig. 1.** Ilustración del algoritmo de detección y manejo de datos históricos

La solución propuesta utiliza parte de la metadata recibida junto con las intensidades de las celdas para sobreescribir la matriz de manera inteligente. Cada vez que se escriba sobre una celda particular, se agrega como información adicional a la intensidad un número índice (timestamp) que puede ser extraído de la trama ASTERIX. Este índice será útil para realizar un chequeo en un barrido posterior y verificar si existen celdas sin actualizar que pertenecen a barridos previos.

Al momento de escribir una celda y luego de incluir el índice actual se realizará un chequeo sobre las celdas superiores dentro de la misma columna. Si el timestamp de las celdas superiores difieren en un número que coincida con barridos previos, se procede a repetir el valor de intensidad de la celda objetivo en las celdas revisadas, eliminando intensidades que no pertenezcan al barrido actual. La cantidad de celdas superiores a verificar será controlada por un offset que se podrá ajustar y dependerá de las frecuencias de las señales de control, la cantidad de muestras asignadas a definir los azimuth y la velocidad de giro del radar.

## 5 Interfaz HW-SW

El conversor analógico digital utilizado posee una profundidad de 14 bits para las intensidades de video. La estructura de datos establece 16 bits de datos para cada celda, por lo que tendremos 2 bits MSB de padding [3]. En la investigación sobre las intensidades se encontró que la información útil correspondiente a video radar se encontraba en el rango [0,256] del total, lo que implicaba usar únicamente 8 de los 16 bits asignados a cada celda. La información restante se atribuyó a ruido y se descartó.

El overhead que generó este descubrimiento a mitad de la implementación no cambió las especificaciones de la estructura de datos utilizada. Se utilizarán los bits restantes de cada celda en la matriz para incluir metadata que sea útil para aplicar el algoritmo de detección de datos históricos.

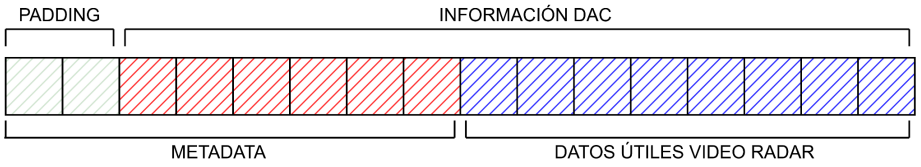


**Fig. 2.** Celda de 16 bits de la matriz en memoria

# 6 Representación visual con OpenGL

## 6.1 Recepción y decodificación de tramas ASTERIX CAT240

El sistema hace uso de una arquitectura orientada a eventos para la recepción de datos radar bajo el protocolo ASTERIX CAT240, transmitido mediante datagramas UDP[4]. Esta funcionalidad está centralizada en un componente **Decodificador**, el cual instancia un socket que permanece a la escucha en un puerto predeterminado. Al momento de recibir un datagrama, se extrae la información contenida en el paquete y la reenvía a la función **decodificar().** El procedimiento de decodificación identifica primero la categoría del paquete, y si corresponde a CAT 240, se analiza la sección FSPEC (Field Specification) para determinar qué campos adicionales están presentes. A partir de esta información, se extraen datos tales como:

- Cabecera de video (azimut inicial/final, rango inicial, duración de celda)
- Resolución y datos de celdas de video

La información de video se interpreta como una secuencia de intensidades (usualmente codificadas en 8 o 16 bits) que representan la reflectividad de cada celda radar para un determinado azimut y rango. Estos datos se encapsulan en una estructura **Paquete**, la cual es enviada a la estructura de datos para su almacenamiento y posterior visualización.

## 6.2 Construcción y actualización de la matriz de intensidades

La matriz almacena los datos de video radar, implementando una estructura matricial bidimensional organizada en Azimut × Distancia. Cada fila representa una línea de barrido a un determinado ángulo, y cada columna una celda a una distancia específica.

Al recibir un paquete, el sistema genera índices dentro de la matriz a partir de sus valores de azimut y distancia, y guarda dentro de ella en el lugar indicado el valor de intensidad correspondiente. Adicionalmente, se mantiene una lista de áreas modificadas que permite al sistema saber qué fragmentos del espacio de radar han sido actualizados. Esta información se utiliza en el renderizador para minimizar la transferencia de datos hacia la GPU, optimizando el rendimiento en tiempo real.

### 6.3 Renderización en tiempo real mediante OpenGL

El módulo **Renderizador**, es responsable de transformar la matriz de intensidades en una representación visual utilizando OpenGL. Esta representación se realiza mediante la generación de una textura 2D (**GL_TEXTURE_2**D), cuyo contenido refleja los valores de intensidad almacenados en la matriz.

Durante la inicialización, se cargan y compilan shaders personalizados que permiten aplicar operaciones como brillo, contraste, umbrales y zoom, directamente en la GPU. Se define un área de visualización en forma de "quad" que ocupa toda la pantalla, sobre la cual se mapea la textura.

La textura se inicializa, asignando memoria y configurando sus parámetros de muestreo. Posteriormente, cada vez que la matriz se actualiza, la función **actualizarIntensidades()** es invocada para subir únicamente los fragmentos modificados de la textura mediante **glTexSubImage2D()**, utilizando los datos de la lista de áreas modificadas. Finalmente, el proceso de renderizado ocurre la función de pintado de OpenGL, donde se:

1. Actualiza la textura en GPU con los nuevos datos.
2. Enlaza los shaders y les pasa los parámetros visuales.
3. Renderiza el quad con la textura que representa el radar.

Una parte crucial para mantener una visualización clara, informativa y ajustada a las condiciones del entorno es el conjunto de controles visuales disponibles: brillo, contraste, umbral mínimo y umbral máximo. Estos parámetros permiten al operador adaptar la representación visual de los ecos radar en función del contenido y la necesidad de visualización.

En la implementación, estos controles se enlazan como *uniforms* al shader de fragmentos. El umbral mínimo define el valor por debajo del cual las intensidades son descartadas (visualizadas como valores nulos), mientras que el umbral máximo actúa como límite superior. La función de contraste ajusta la pendiente de la transformación entre niveles bajos y altos de intensidad, y el brillo desplaza el rango de intensidades hacia valores más claros u oscuros. Todo esto se realiza en tiempo real, sin necesidad de reescalar la matriz ni alterar los datos originales.

```
if (distancia_cruda <= RANGO_MAXIMO) {
color_tex = textura(matrixTexture, coordenadas_polares);
intensidad = color_tex.r;
if (intensidad < umbral_min || intensidad > umbral_max) {
    intensidad = 0.0;
} else {
    // Normalización del rango
    intensidad = (intensidad - umbral_min) / (umbral_max
- umbral_min);
    // Realce logarítmico de detalles bajos
    intensidad = log(1 + intensidad * FACTOR_ESCALA) /
log(1 + FACTOR_ESCALA);
    // Corrección gamma
```

```
    intensidad = pow(intensidad, 1.0 / 2.0);
    // Contraste centrado
    intensidad = 0.5 + (intensidad - 0.5) * contraste;
    // Brillo
    intensidad += brillo;
}
FragColor = vec4(intensidad, intensidad, intensidad,
1.0);
} else {
    FragColor = vec4(0.0, 0.0, 0.0, 1.0);
}
```

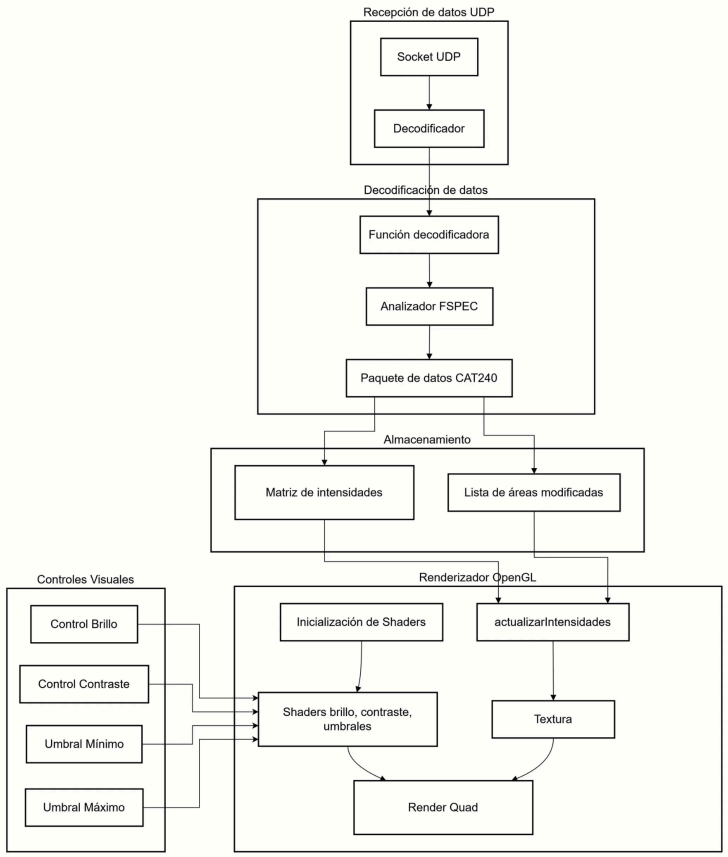


**Fig. 3.** Diagrama de componentes del visualizador.

Este enfoque permite al operador maximizar el detalle visual relevante, minimizando el ruido y adaptando la escala de grises a cada situación. Es, por tanto, un mecanismo esencial para garantizar que la representación radar sea tanto precisa como legible en contextos operativos exigentes. Este orden de operaciones es crítico

para preservar tanto la fidelidad como la legibilidad visual. La normalización debe preceder al mapeo logarítmico para que los valores sean consistentes; la corrección gamma suaviza el comportamiento visual en intensidades bajas; el contraste se aplica simétricamente respecto del punto medio; y finalmente, el brillo desplaza el resultado global. Esta secuencia garantiza una representación adaptativa, útil para distintas condiciones ambientales.

# 7 Hardware

La solución propuesta se implementó en tres computadoras de capacidades diferentes:

1. Intel Core I7-7500U, 2.7GHz - 2 núcleos - 8 Gb RAM - Intel HD Graphics 620 - Windows
2. Intel Core I5-11400H, 2.7GHz - 6 núcleos - 8 Gb RAM - NVIDIA GeForce 1060 - Windows
3. Intel Core I5-10400, 2.0GHz - 6 núcleos - 8 Gb RAM - Intel UHC Graphics 630 - Linux

# 8 Conclusiones

## 8.1 Resultados alcanzados

Los resultados demuestran que el sistema propuesto proporciona una visualización radar de alta calidad con tiempos de respuesta reducidos, beneficiándose de la aceleración por hardware y el procesamiento paralelo. La arquitectura modular facilita la integración con otros sistemas y la adaptación a diferentes configuraciones de hardware. Este trabajo contribuye al campo de los sistemas de visualización radar al ofrecer una implementación optimizada que aprovecha las capacidades de procesamiento gráfico moderno junto con la alta resolución proporcionada por el convertidor.

En cuanto al rendimiento en los sistemas donde se implementó la solución, se encontró un uso moderado de la GPU en torno al 40%, excepto la (2) que aproximó apenas un 20%, medido en los sistemas Windows, acompañado de un uso aproximado del 40% total de la capacidad de la CPU. Específicamente, el sistema utiliza intensivamente (77% aproximadamente) el núcleo donde se aloja el hilo dedicado a la decodificación de la trama ASTERIX, lo cual demuestra el alto ancho de banda utilizado y la necesidad de un módulo de procesamiento I/O lo suficientemente eficiente. En el sistema con Linux, cabe resaltar un leve incremento del **iowait** del sistema. Esto responde a la necesidad del sistema de aprovechar los recursos de disco y utilizar hardware competente, como SSD y memorias principales de alto rendimiento.

Por último, la computadora con 2 núcleos mantiene el mismo uso aproximado de las otras dos, pero los FPS percibidos al utilizar la aplicación bajan considerablemente. No se recomienda por lo tanto instalar en sistemas con menos de 4 núcleos debido a la necesidad de núcleos dedicados a la decodificación

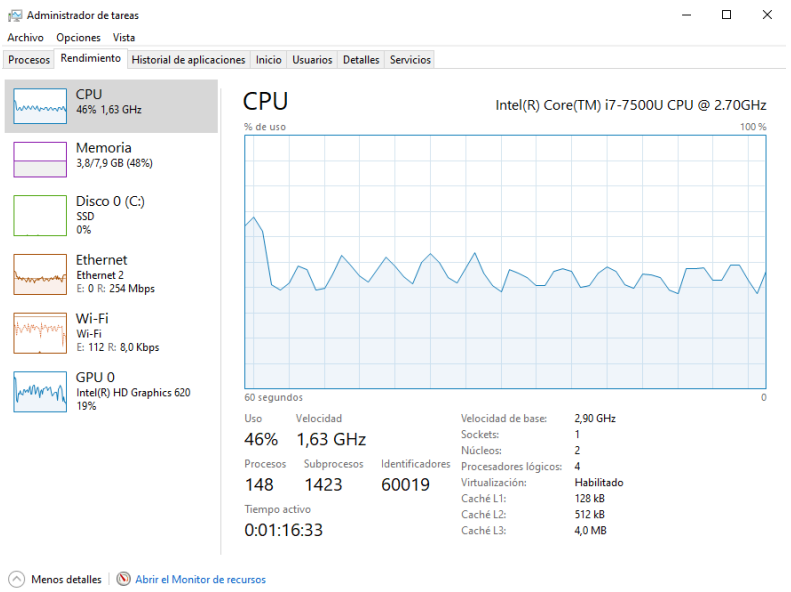


**Fig 4.** Rendimiento computadora 1

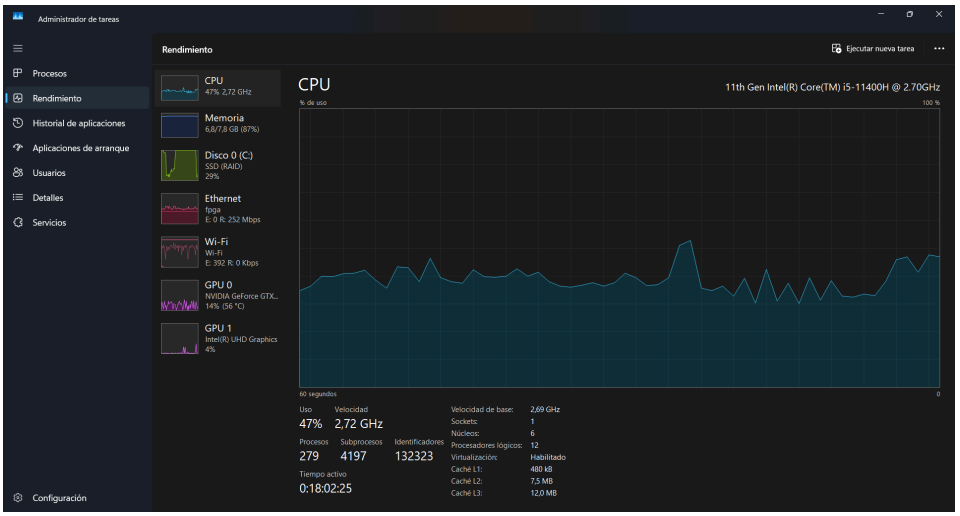


**Fig 5.** Rendimiento computadora 2

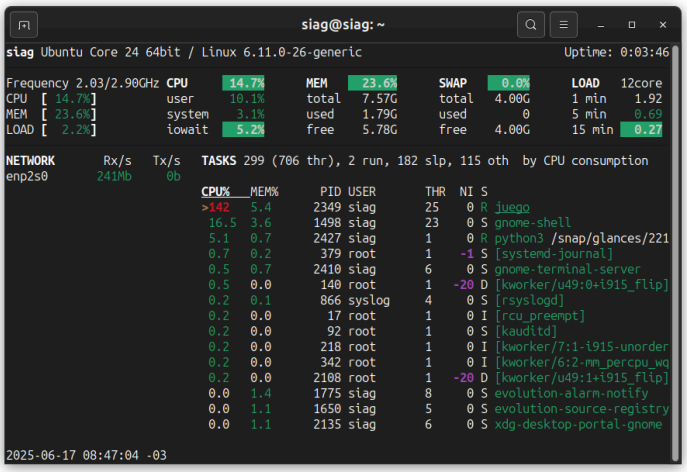


**Fig 5.** Rendimiento computadora 3

### 8.2 Trabajo futuro

Se prevé realizar estudios sobre OpenGL o tecnologías similares para renderizar en pantalla radares de mayor alcance, poniendo énfasis en la performance y el uso de la memoria disponible en el sistema. Los ensayos realizados sobre el procesamiento de la imagen son sólo un acercamiento con gran margen de mejora, siendo posible un trabajo de investigación posterior en conjunto con operarios del sistema que definen requerimientos específicos de visualización.

## Referencias

1. Página oficial de EUROCONTROL: https://www.eurocontrol.int/asterix.
2. Documentación OpenGL : https://www.opengl.org/Documentation/Documentation.html
3. Diego M. Martínez et al. Sistema embebido para la distribución de video crudo radar. Memorias de las 53 JAIIO - SAIC, Simposio Argentino de Ingeniería en Computación. Pp 93 – 102. 12 al 16 de agosto de 2024. ISSN 2451-7496.
4. Allende, Agustin Emannuel & Miguel Aguirre, Juan José. (2023). “Desarrollo de una red de distribución de vídeo RADAR en protocolo ASTERIX CAT240 sobre Ethernet Gigabit”
5. Gálvez, Nelida; Valdez Mariano; Cayssials, Ricardo.(2024). Generación de datos ASTERIX CAT 240 mediante FPGA. Memorias de las 53 JAIIO - SAIC. 12 al 16 de agosto de 2024. ISSN 2451-7496.
6. Explicación del concepto de milla de datos: https://scienceworld.wolfram.com/physics/DataMile.html