

PONTIFICIA UNIVERSIDAD JAVERIANA

ESTRUCTURA DE DATOS

ENTREGA#1 PROYECTO

PRESENTA:

Diego Fernando Zabala,
Brandon Garcia Rodriguez
Santiago Camargo Trujillo
David Villareal

PROFESOR

John Jairo Corredor Franco

BOGOTA D.C

2025

- **Resumen**

El procesamiento de volúmenes 3D a partir de imágenes PGM enfrenta el desafío de garantizar una carga estructurada y homogénea de los datos. Se propone una implementación en C++ que organiza las imágenes en una estructura tridimensional, verificando su formato y tamaño. Los resultados muestran una carga eficiente del volumen, permitiendo generar proyecciones en distintas direcciones.

- **Introducción**

En el presente trabajo se aborda el problema de la carga de imágenes en formato pgm y también se aborda la carga y proyección de volúmenes tridimensionales a partir de una serie de imágenes en formato PPM. La dificultad principal radica en garantizar la correcta lectura y estructuración de los datos, asegurando que todas las imágenes posean dimensiones homogéneas y que el volumen resultante sea adecuado para posteriores análisis y proyecciones.

Para solucionar este problema, se propone una implementación en C++ que permite la lectura secuencial de múltiples archivos PPM y de una imagen de tipo PGM donde se usan struct para almacenar imágenes haciendo uso de vectores para datos de tipo entero y para varias imágenes ppm se usa un struct de volumen en el cual se usa un vector de varias imágenes.

- **Diseño**

Proyección de Imágenes

El TAD Proyección de Imágenes permite manejar información de los píxeles de imágenes y también volúmenes 3D de imágenes en escala de grises y generar una proyección 2D a partir de criterios de colapso en una dirección específica.

- **Datos**

El TAD maneja la siguiente información:

volumen: Una serie ordenada de imágenes 2D en escala de grises representadas como una lista de matrices de enteros (valores de intensidad de píxeles en el rango [0, 255]).

ancho (W): Cantidad de columnas en cada imagen del volumen.

alto (H): Cantidad de filas en cada imagen del volumen.

num_imagenes (D): Cantidad de imágenes en la serie (profundidad del volumen).

proyección: Matriz resultante después de aplicar la proyección en una dirección dada.

- Operaciones

El TAD debe implementar las siguientes operaciones:

1. Cargar una imagen individual en memoria

Entrada: nombre_imagen.pgm (ruta del archivo).

Salida: Mensaje indicando éxito o error.

Descripción: Carga una imagen PGM en memoria, sobrescribiendo cualquier imagen cargada previamente.

2. Cargar un volumen de imágenes en memoria

Entrada: nombre_base (nombre común de los archivos) y n_im (cantidad de imágenes en el volumen).

Salida: Mensaje de éxito o error.

Descripción: Carga una secuencia de imágenes PGM en memoria, formando el volumen 3D.

3. Obtener información de la imagen cargada

Salida: Mensaje con el nombre, ancho y alto de la imagen cargada.

Descripción: Muestra en pantalla la información de la imagen almacenada en memoria.

4. Obtener información del volumen cargado

Salida: Mensaje con el nombre base, tamaño (cantidad de imágenes), ancho y alto del volumen.

Descripción: Muestra en pantalla la información del volumen almacenado en memoria.

5. Generar una proyección 2D del volumen

Entrada: dirección (x, y, z), criterio (mínimo, máximo, promedio, mediana), nombre_archivo.pgm.

Salida: Imagen PGM con la proyección generada y mensaje de éxito o error.

Descripción: Recorre el volumen en la dirección dada, colapsando los valores según el criterio especificado para generar la imagen 2D resultante.

- Condiciones

El volumen debe estar cargado antes de ejecutar la proyección.

Los archivos PGM deben ser válidos (deben existir y contener datos en el formato correcto).

El número de imágenes en el volumen no puede superar 99, según la especificación.

Los criterios de proyección deben ser válidos, limitados a mínimo, máximo, promedio y mediana.

La dirección de proyección debe ser válida, siendo x, y o z.

- **Análisis**

El problema central abordado en este proyecto es la carga y estructuración de imágenes en formato PGM y PPM para la generación de volúmenes tridimensionales. La principal dificultad radica en garantizar la coherencia en dimensiones y formato de las imágenes, permitiendo una correcta representación del volumen y su posterior análisis mediante proyecciones.

Para resolver este desafío, se implementó una solución en C++ basada en estructuras de datos eficientes. Se utilizó una representación mediante `struct` para modelar las imágenes y un `vector<vector<vector<int>>>` para gestionar el volumen tridimensional. Esta estructura permite acceder de manera eficiente a los valores de píxeles en cualquier posición del espacio 3D y facilita la manipulación de los datos.

Durante el desarrollo, se diseñaron y probaron varias funciones esenciales, tales como:

1. **Carga de imágenes individuales:** Se valida la existencia del archivo, su formato y sus dimensiones antes de almacenarlo en memoria.
2. **Carga de volúmenes tridimensionales:** Se verifica la consistencia de las imágenes para formar un volumen homogéneo, evitando errores de tamaño o formato.
3. **Extracción de información:** Se implementaron funciones para recuperar detalles sobre imágenes individuales y volúmenes completos, facilitando la inspección de los datos.
4. **Generación de proyecciones 2D:** Se desarrolló un método para proyectar el volumen en los ejes X, Y y Z, utilizando diferentes criterios de colapso (mínimo, máximo, promedio y mediana).

El diseño adoptado permite una gestión flexible y eficiente de los datos. Sin embargo, el uso de vectores anidados implica un costo en memoria que podría optimizarse con estructuras más avanzadas, como el uso de arreglos dinámicos o técnicas de compresión de datos en volúmenes grandes.

- **Plan de pruebas**

Vamos a probar uno a uno los comandos

Antes compilamos el programa de la siguiente manera:

```
brandon_garcia@localhost:~/Documentos/ESTRUCTURA DE DATOS/PROYECTO
[brandon_garcia@localhost PROYECTO]$ g++ -std=c++20 -o proyecto proyecto.cpp
[brandon_garcia@localhost PROYECTO]$ ./proyecto
```

Esto con el fin de que funcionen las librerías que implementamos en el código, ya que solo funcionan desde `-std=c++17` en adelante.

Ahora si ejecutamos el programa y empezamos a probar uno a uno los comandos

Comando: `cargar_imagen`:

```
brandon_garcia@localhost:~/Documentos/ESTRUCTURA DE DATOS/PROYECTO
[brandon_garcia@localhost PROYECTO]$ g++ -std=c++20 -o proyecto proyecto.cpp
[brandon_garcia@localhost PROYECTO]$ ./proyecto
$ cargar_imagen img_04.pgm
La imagen img_04.pgm ha sido cargada.
$
```

Ya acá tenemos cargada la imagen con nombre `img_04.pgm` ahora probaremos con una imagen inexistente:

```
brandon_garcia@localhost:~/Documentos/ESTRUCTURA DE DATOS/PROYECTO
[brandon_garcia@localhost PROYECTO]$ g++ -std=c++20 -o proyecto proyecto.cpp
[brandon_garcia@localhost PROYECTO]$ ./proyecto
$ cargar_imagen img_04.pgm
La imagen img_04.pgm ha sido cargada.
$ cargar_imagen imagen-01.pgm
Error: La imagen imagen-01.pgm no ha podido ser cargada.
$
```

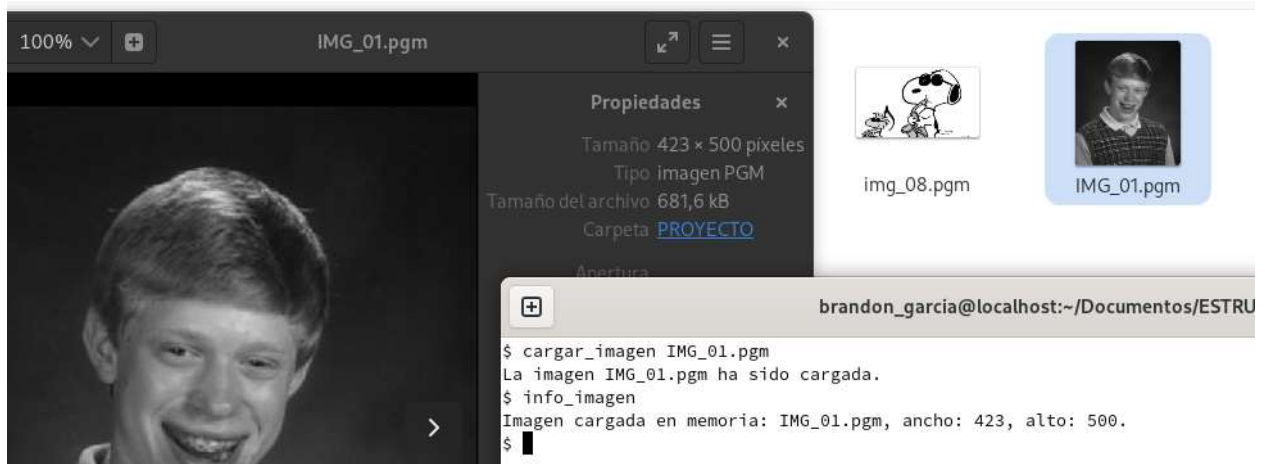
Y tal cual como debería ser la imagen no ha podido ser cargada con ello probamos ambos mensajes tanto el de éxito como el de error.

Comando: info_imagen

Ahora probamos la información que debió ser cargada a partir de la función de cargar_imagen la cual debe ser ancho y altura en pixeles de la imagen.



Acá comprobamos que la función cargar_imagen y la función info_imagen funcionan correctamente tanto guardando como mostrando los datos que se piden, igual comprobemos con otra imagen



Con ello ya tenemos que los comandos funcionan perfectamente.

Comando: cargar_volumen



```

+ brandon_garcia@localhost:~/Documentos/ESTRUCTURA DE DATOS
$ cargar_volumen IM-126-0002-epiT2 36
Cargando volumen desde carpeta: IM-126-0002-epiT2-ppm con 36 imágenes...
Leyendo IM-126-0002-epiT2-ppm/IM-126-0002-epiT201.ppm - Ancho: 256, Alto: 256
  Cargando IM-126-0002-epiT2-ppm/IM-126-0002-epiT201.ppm (Ancho: 256, Alto: 256)...
Leyendo IM-126-0002-epiT2-ppm/IM-126-0002-epiT202.ppm - Ancho: 256, Alto: 256
  Cargando IM-126-0002-epiT2-ppm/IM-126-0002-epiT202.ppm (Ancho: 256, Alto: 256)...
Leyendo IM-126-0002-epiT2-ppm/IM-126-0002-epiT203.ppm - Ancho: 256, Alto: 256
  Cargando IM-126-0002-epiT2-ppm/IM-126-0002-epiT203.ppm (Ancho: 256, Alto: 256)...
Leyendo IM-126-0002-epiT2-ppm/IM-126-0002-epiT204.ppm - Ancho: 256, Alto: 256
  Cargando IM-126-0002-epiT2-ppm/IM-126-0002-epiT204.ppm (Ancho: 256, Alto: 256)...
Leyendo IM-126-0002-epiT2-ppm/IM-126-0002-epiT205.ppm - Ancho: 256, Alto: 256
  Cargando IM-126-0002-epiT2-ppm/IM-126-0002-epiT205.ppm (Ancho: 256, Alto: 256)...
Leyendo IM-126-0002-epiT2-ppm/IM-126-0002-epiT206.ppm - Ancho: 256, Alto: 256
  Cargando IM-126-0002-epiT2-ppm/IM-126-0002-epiT206.ppm (Ancho: 256, Alto: 256)...
Leyendo IM-126-0002-epiT2-ppm/IM-126-0002-epiT207.ppm - Ancho: 256, Alto: 256

```

La función busca la carpeta que debe tener el mismo nombre base de las imágenes de las cuales se desea obtener la información.

Y se guarda la información de ancho y alto de pixeles que es un promedio de todas las imágenes.

```

  Cargando IM-126-0002-epiT2-ppm/IM-126-0002-epiT234.ppm (Ancho: 256, Alto: 256)...
Leyendo IM-126-0002-epiT2-ppm/IM-126-0002-epiT235.ppm - Ancho: 256, Alto: 256
  Cargando IM-126-0002-epiT2-ppm/IM-126-0002-epiT235.ppm (Ancho: 256, Alto: 256)...
Leyendo IM-126-0002-epiT2-ppm/IM-126-0002-epiT236.ppm - Ancho: 256, Alto: 256
  Cargando IM-126-0002-epiT2-ppm/IM-126-0002-epiT236.ppm (Ancho: 256, Alto: 256)...
☒ Volumen cargado correctamente desde 'IM-126-0002-epiT2-ppm'.
$

```

También la función permite el uso de que las carpetas la ppm acabe en _

```

$ cargar_volumen t1_icbm_5mm 36
Cargando volumen desde carpeta: t1_icbm_5mm_ppm con 36 imágenes...
Leyendo t1_icbm_5mm_ppm/t1_icbm_5mm_01.ppm - Ancho: 180, Alto: 216
  Cargando t1_icbm_5mm_ppm/t1_icbm_5mm_01.ppm (Ancho: 180, Alto: 216)...
Leyendo t1_icbm_5mm_ppm/t1_icbm_5mm_02.ppm - Ancho: 180, Alto: 216
  Cargando t1_icbm_5mm_ppm/t1_icbm_5mm_02.ppm (Ancho: 180, Alto: 216)...
Leyendo t1_icbm_5mm_ppm/t1_icbm_5mm_03.ppm - Ancho: 180, Alto: 216
  Cargando t1_icbm_5mm_ppm/t1_icbm_5mm_03.ppm (Ancho: 180, Alto: 216)...

```

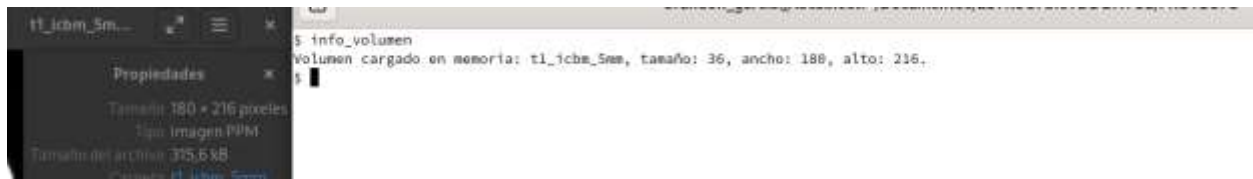
Comando: Info_volumen

```

  Cargando IM-126-0002-epiT2-ppm/IM-126-0002-epiT235.ppm (Ancho: 256, Alto: 256)...
Leyendo IM-126-0002-epiT2-ppm/IM-126-0002-epiT236.ppm - Ancho: 256, Alto: 256
  Cargando IM-126-0002-epiT2-ppm/IM-126-0002-epiT236.ppm (Ancho: 256, Alto: 256)...
☒ Volumen cargado correctamente desde 'IM-126-0002-epiT2-ppm'.
$ info_volumen
Volumen cargado en memoria: IM-126-0002-epiT2, tamaño: 36, ancho: 256, alto: 256.
$

```

Como lo dice el mismo nombre del comando muestra la información que es proporcionada por el uso de la función cargar_volumen

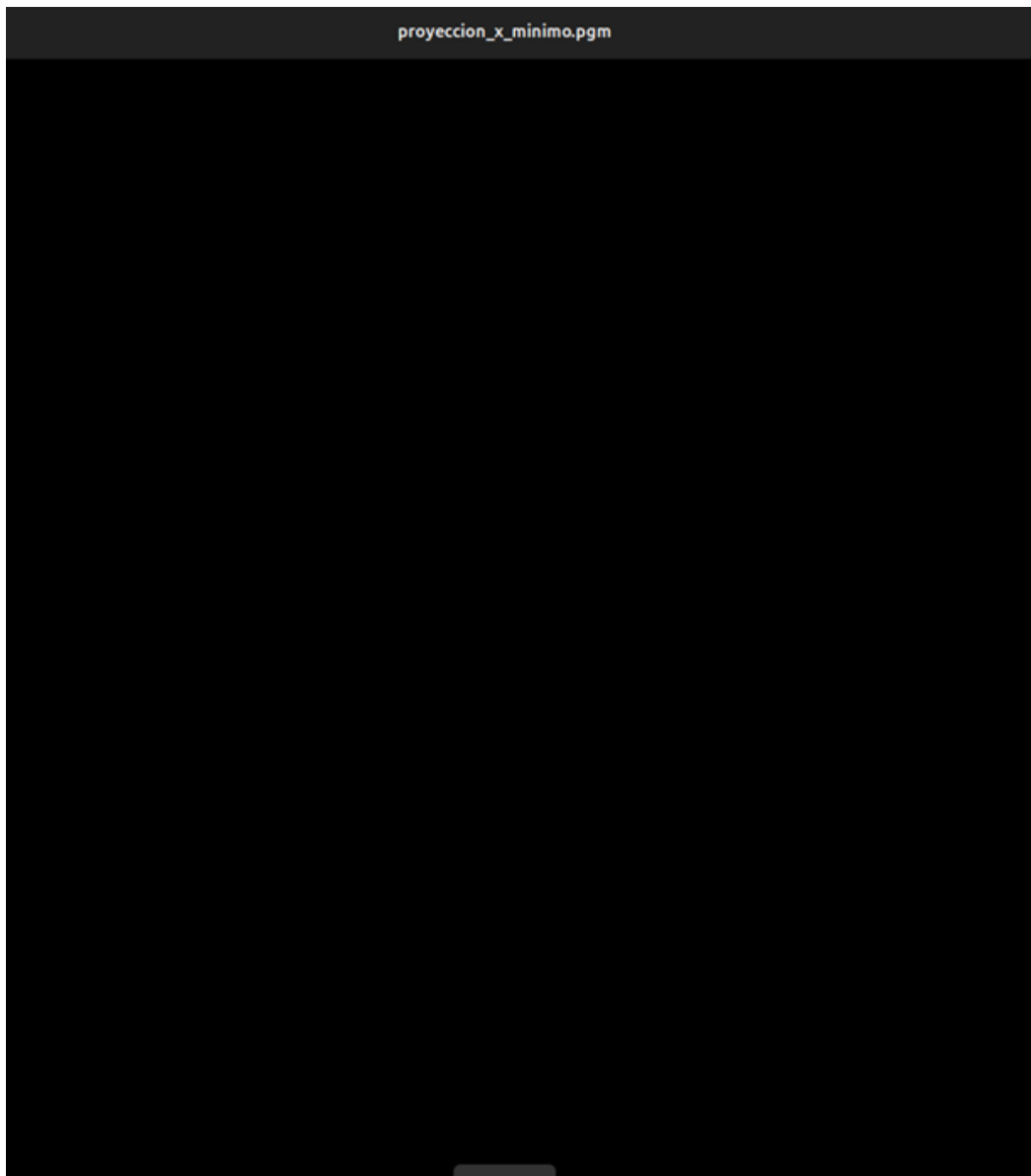


Proyecciones en el eje X:

1. Criterio mínimo:

Copiar

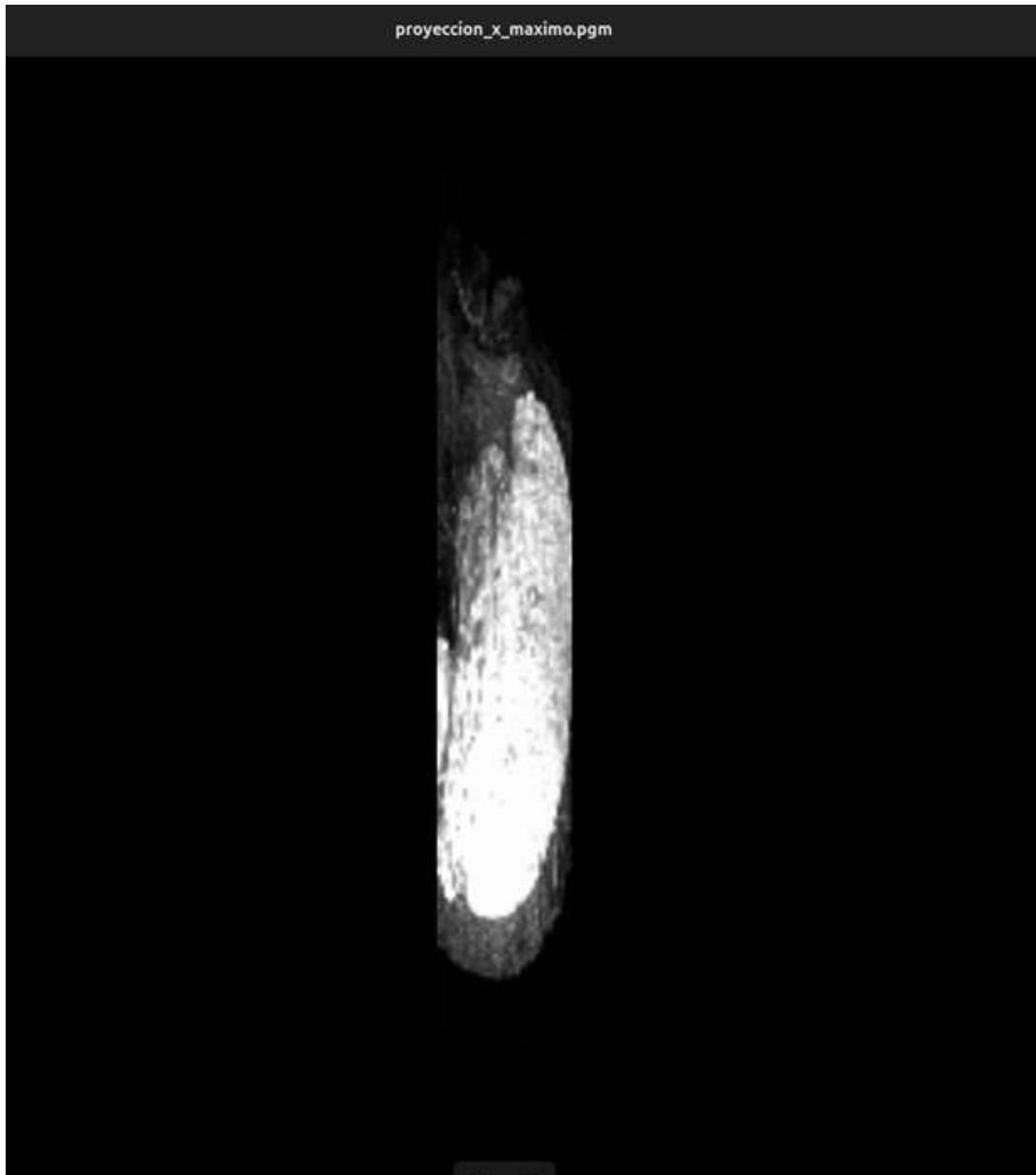
proyeccion2D x minimo proyeccion_x_minimo.pgm



2. Criterio máximo:

Copiar

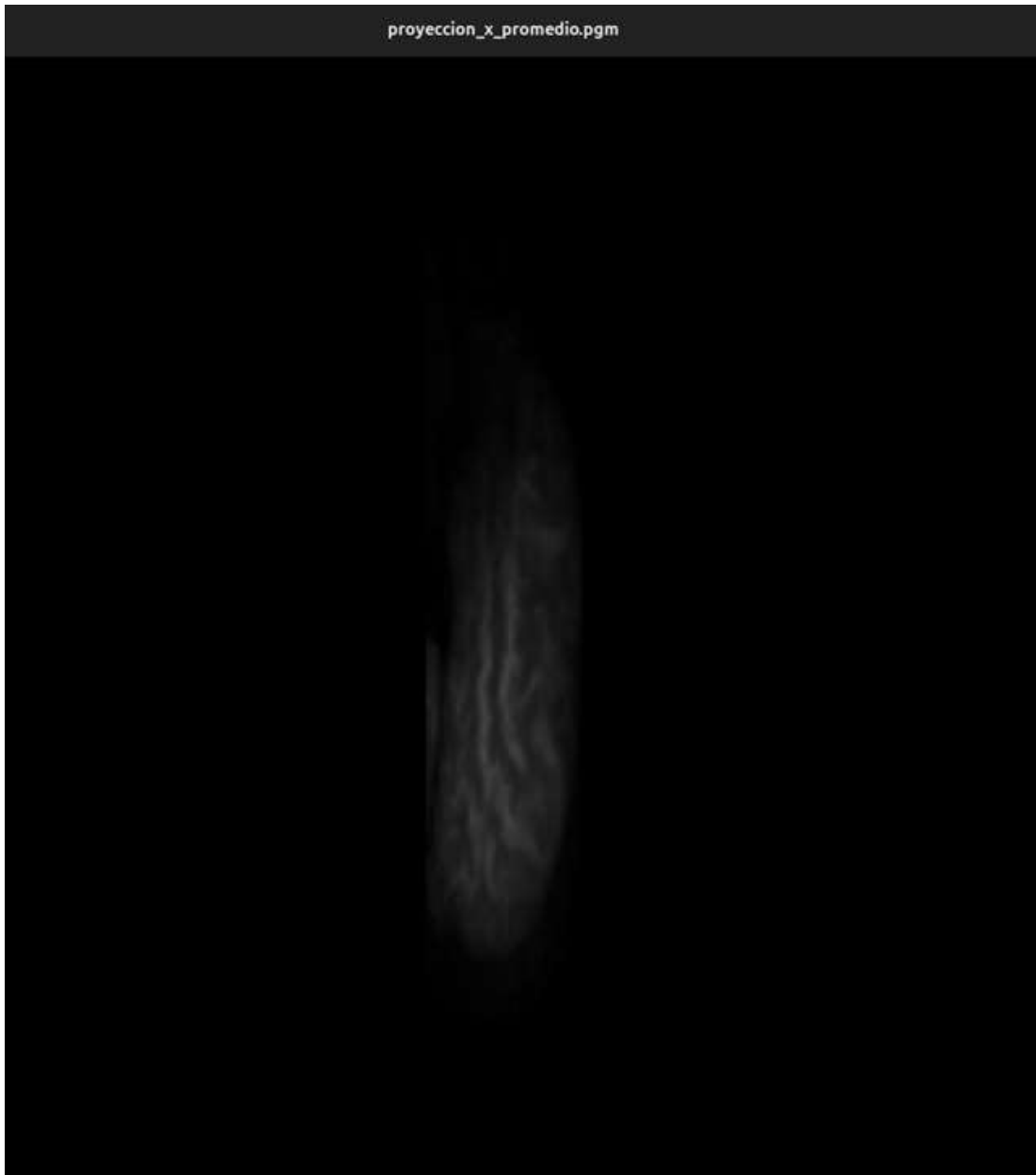
proyeccion2D x maximo proyeccion_x_maximo.pgm



3. Criterio promedio:

Copiar

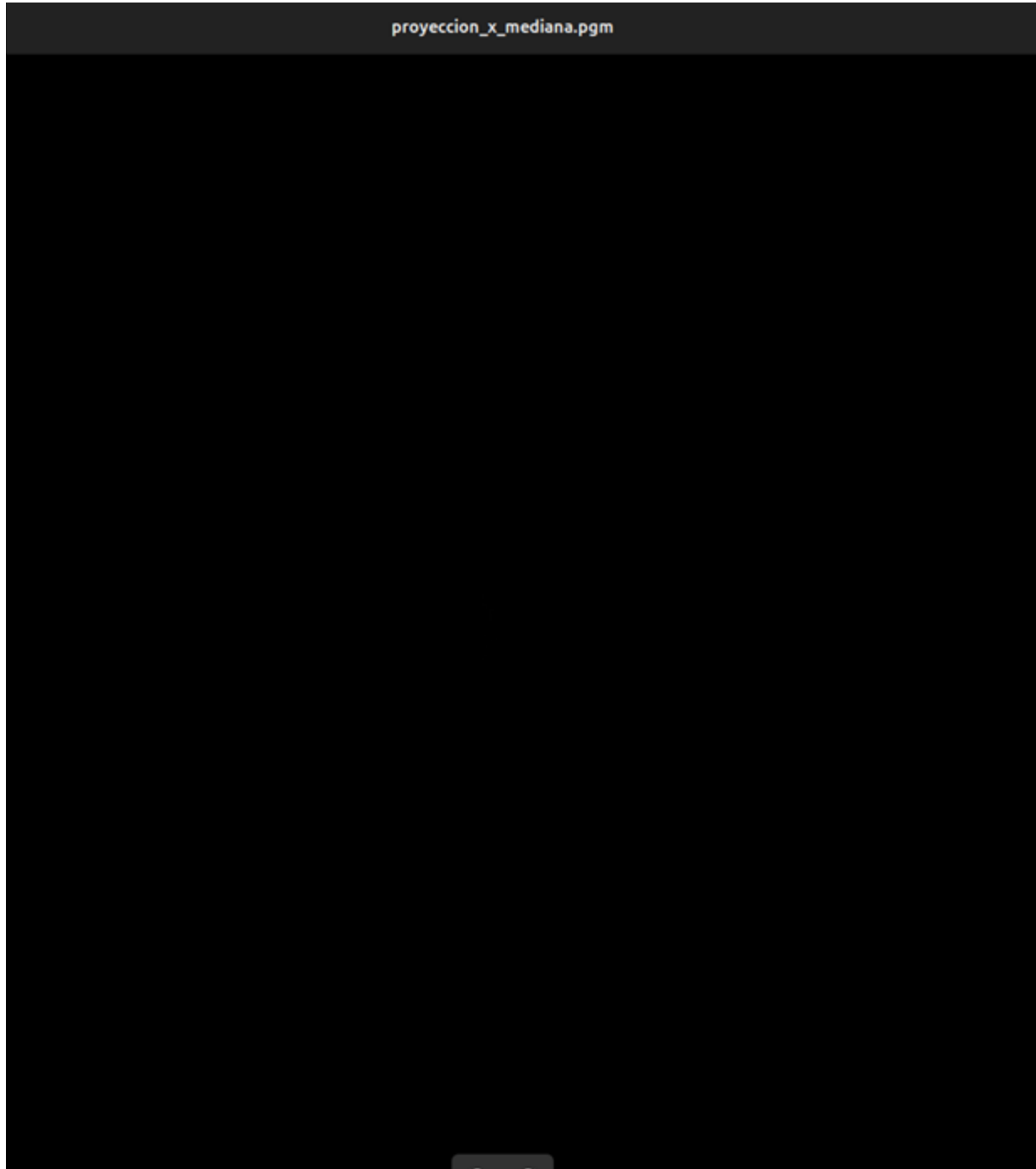
proyeccion2D x promedio proyeccion_x_promedio.pgm



4.Criterio mediana:

Copiar

proyeccion2D x mediana proyeccion_x_mediana.pgm

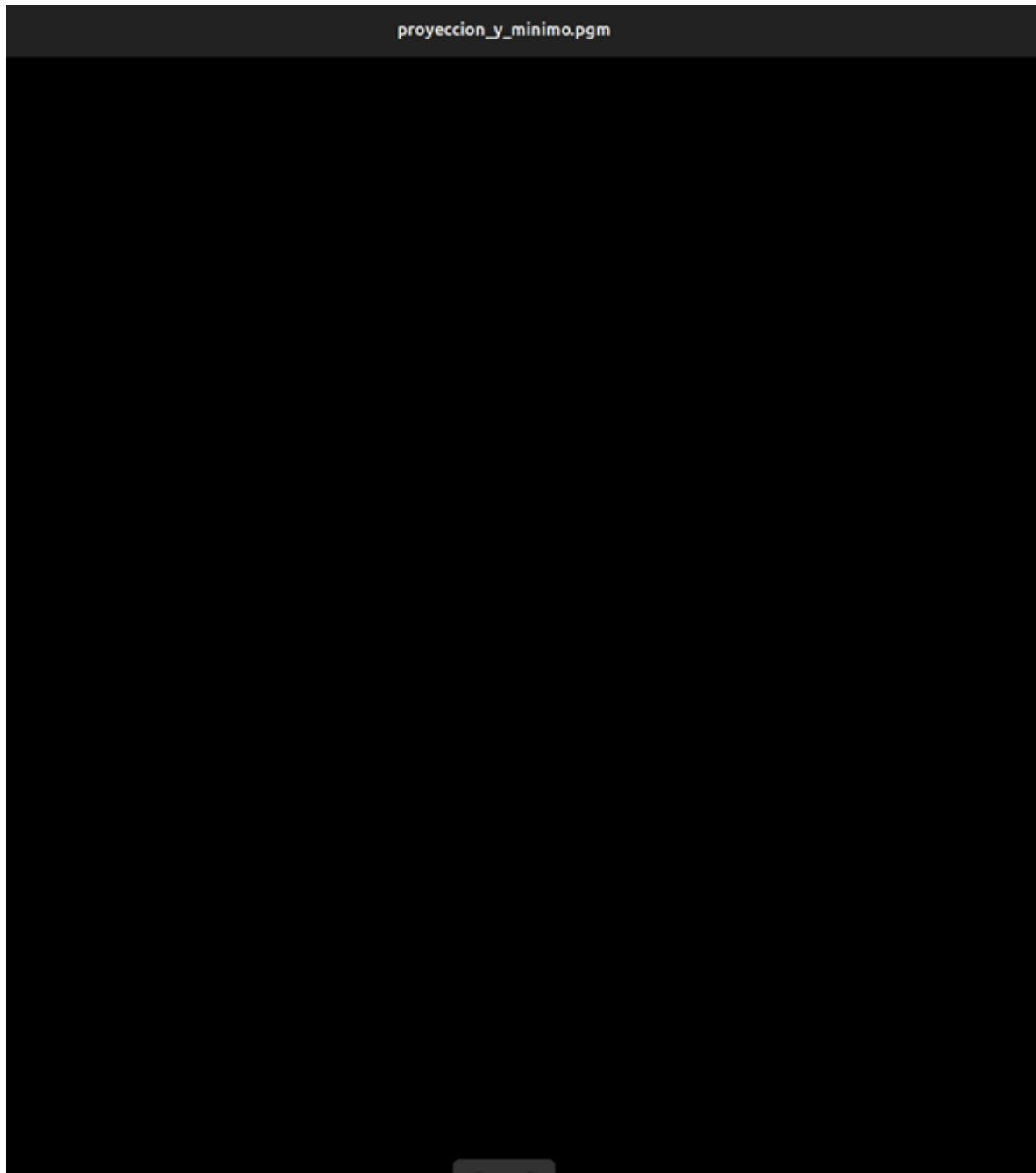


Proyecciones en el eje Y:

1. Criterio mínimo:

Copiar

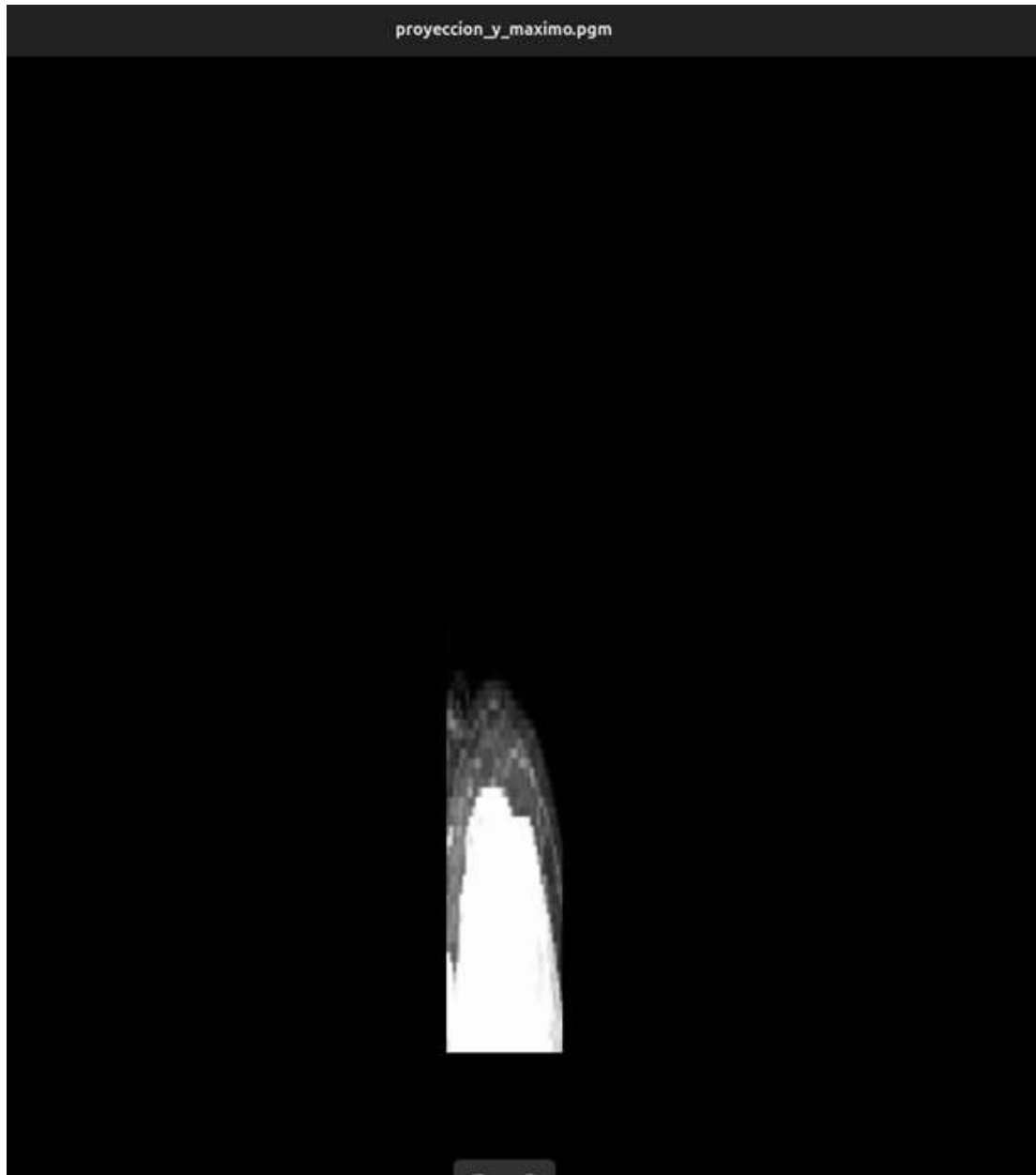
proyeccion2D y minimo proyeccion_y_minimo.pgm



2. Criterio máximo:

Copiar

proyeccion2D y maximo proyeccion_y_maximo.pgm



3. Criterio promedio:

Copiar

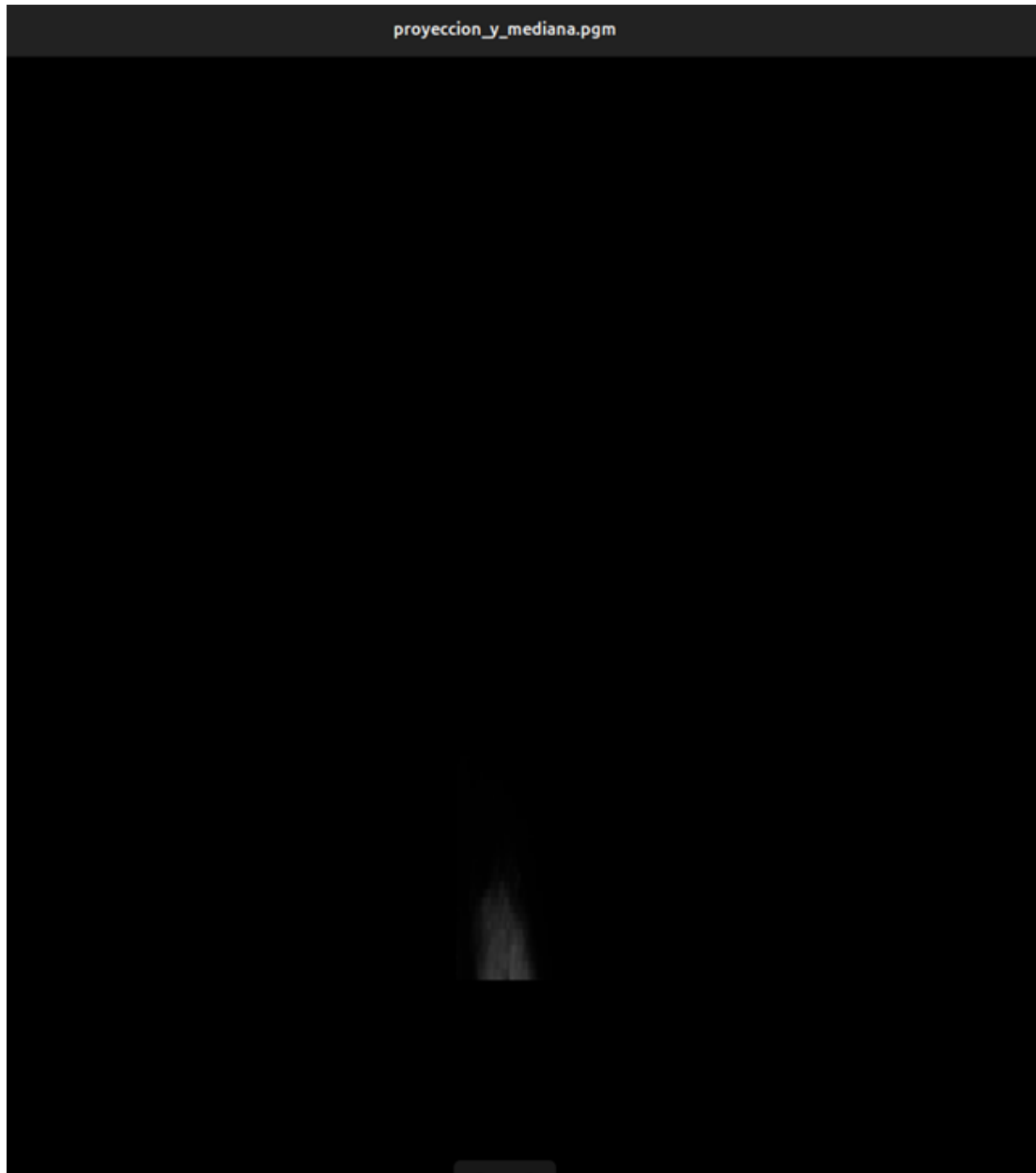
proyeccion2D y promedio proyeccion_y_promedio.pgm



4. Criterio mediana:

Copiar

proyeccion2D y mediana proyeccion_y_mediana.pgm

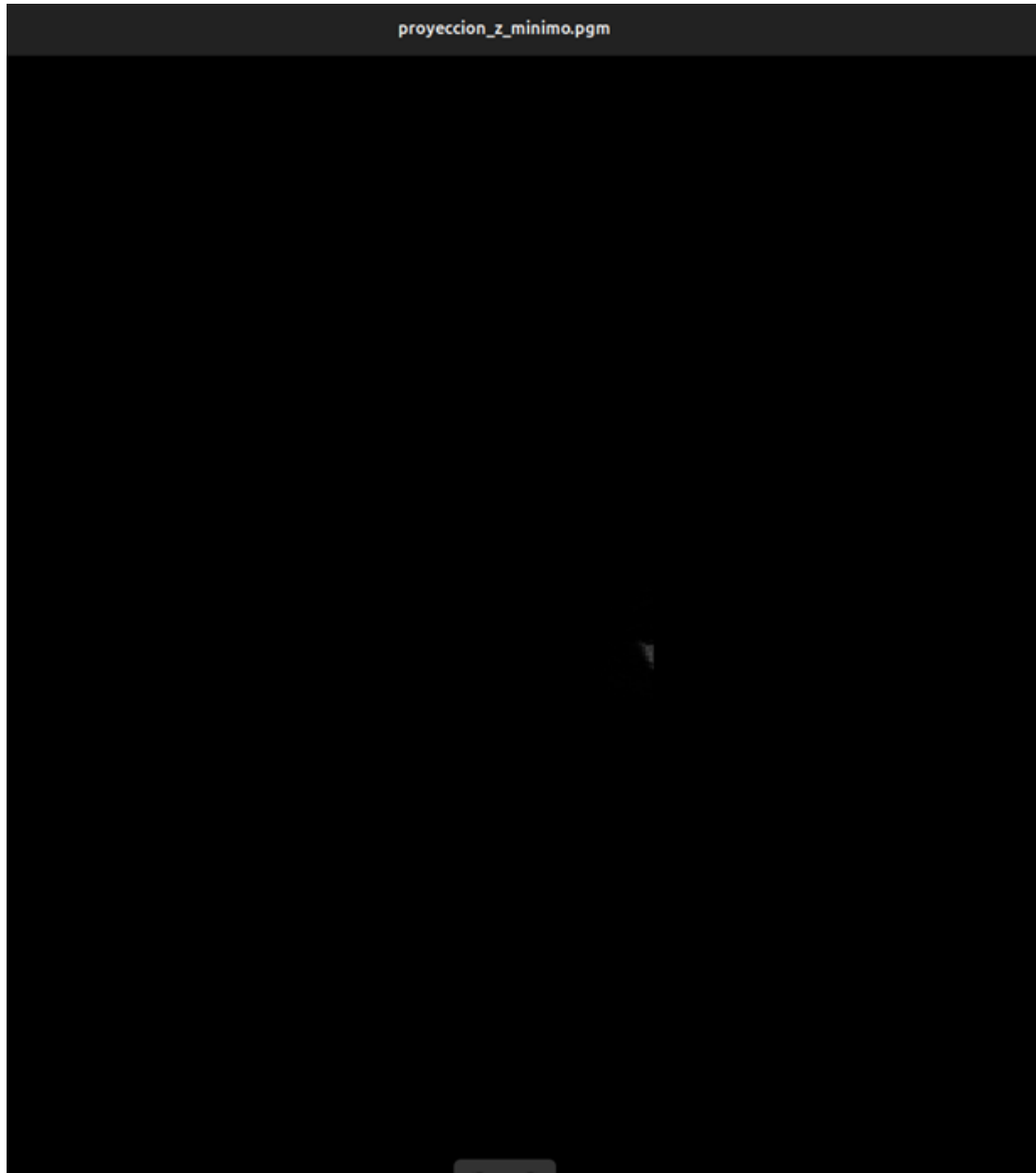


Proyecciones en el eje Z:

1. Criterio mínimo:

Copiar

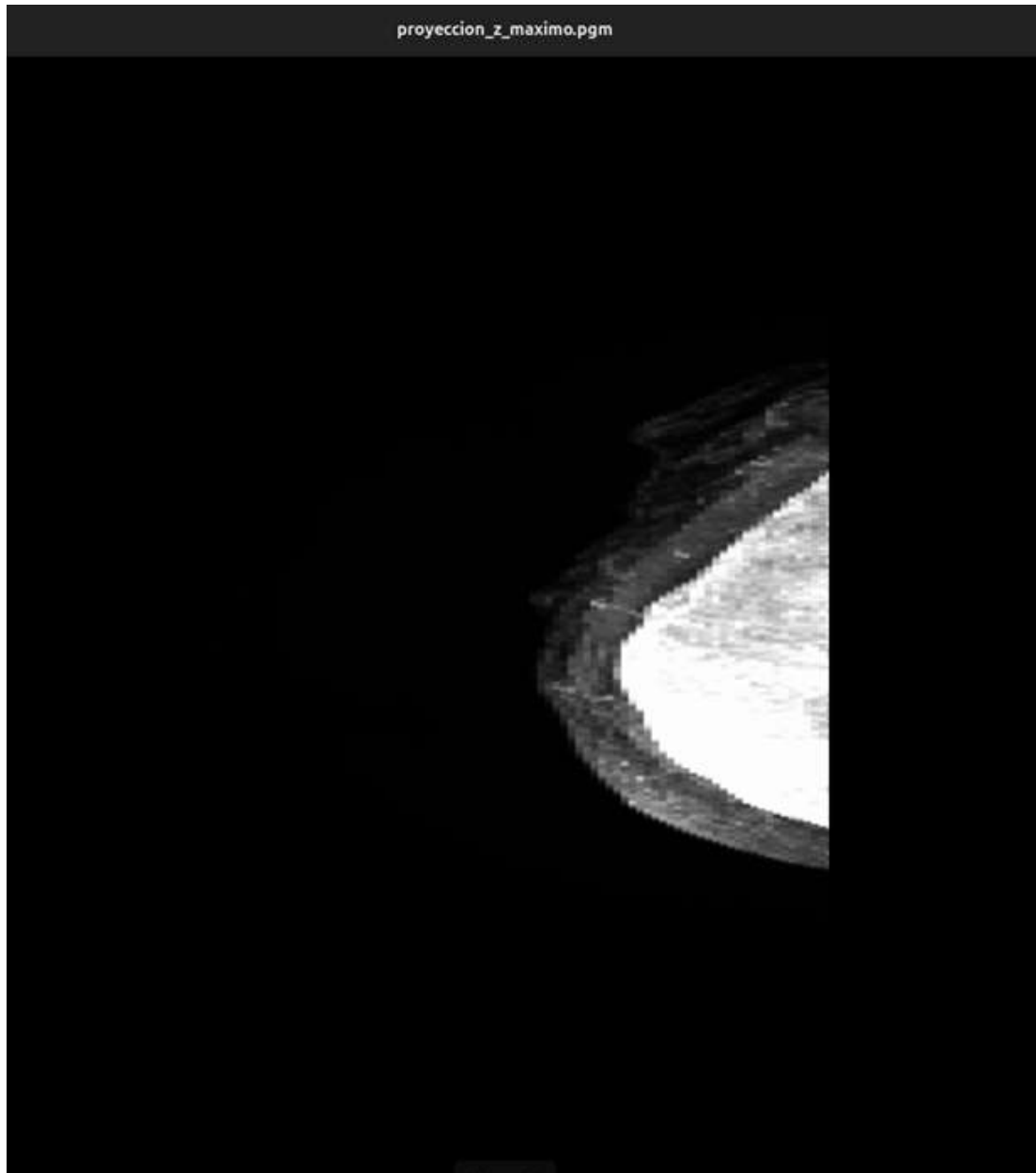
proyeccion2D z minimo proyeccion_z_minimo.pgm



2. Criterio máximo:

Copiar

proyeccion2D z maximo proyeccion_z_maximo.pgm



3. Criterio promedio:

Copiar

proyeccion2D z promedio proyeccion_z_promedio.pgm



4. Criterio mediana:

Copiar

proyeccion2D z mediana proyeccion_z_mediana.pgm



- **Conclusiones**

Los resultados obtenidos confirman que la implementación en C++ desarrollada permite la correcta construcción de volúmenes tridimensionales a partir de imágenes individuales en formato PGM y PPM. Se verificó que el sistema carga de manera eficiente los datos, asegurando que todas las imágenes cumplan con los requisitos de homogeneidad en dimensiones y formato.

Además, el sistema facilita la generación de proyecciones en distintas direcciones (X, Y, Z), permitiendo el análisis del volumen desde diversas perspectivas. La implementación de diferentes criterios de proyección (mínimo, máximo, promedio y mediana) ofrece flexibilidad en la manipulación de los datos, asegurando una representación ajustada a las necesidades del usuario.

Las pruebas realizadas validaron la robustez del sistema, asegurando que las operaciones de carga de imágenes, carga de volumen, recuperación de información y generación de proyecciones funcionan correctamente. Se confirmó que el manejo de errores es adecuado, impidiendo la carga de imágenes inexistentes o con formatos incorrectos.