

Proyecto de programación orientada al rendimiento

J. Daniel Garcia (coordinador)
Arquitectura de Computadores
Departamento de Informática
Universidad Carlos III de Madrid

15 de octubre de 2025

1. Objetivo

Este proyecto tiene como objetivo fundamental hacer el que los estudiantes se familiaricen con la **optimización de programas secuenciales**.

En concreto, la práctica se centrará en el desarrollo de software secuencial en el lenguaje de programación C++ (incluyendo las mejoras de C++23).

2. Visión General

En este proyecto se construirá una aplicación de síntesis de imágenes 3D (*3D rendering*). El software tomará una figura que describe una escena con objetos 3D y utilizará un conjunto de parámetros de configuración para generar una imagen con una representación bidimensional.

El software desarrollado utilizará dos archivos de configuración en formato de texto:

- Un archivo de descripción de la escena.
- Un archivo de configuración del motor de síntesis.

El formato de las imágenes de salida será PPM (tipo P3), que representa una imagen como una secuencia de píxeles en codificación RGB.

La figura 1 presenta una imagen generada con muchas esferas y cilindros.

2.1. Descripción de una escena

Una escena está formada por una colección de objetos en el espacio tridimensional. Además cada uno de estos objetos tiene un material asociado que define las propiedades que tiene dicho objeto cuando los rayos inciden sobre él.

2.1.1. Objetos en el espacio tridimensional

La aplicación de síntesis de imagen genera una imagen 2D a partir de la representación de una escena con objetos 3D. Los objetos que deben soportarse son cilindros y esferas. Cada uno de estos objetos tendrá asociado un material con unas propiedades determinadas.

Una **esfera** viene dada por las siguientes propiedades:

- La posición de su **centro** $C \equiv (c_x, c_y, c_z)$.

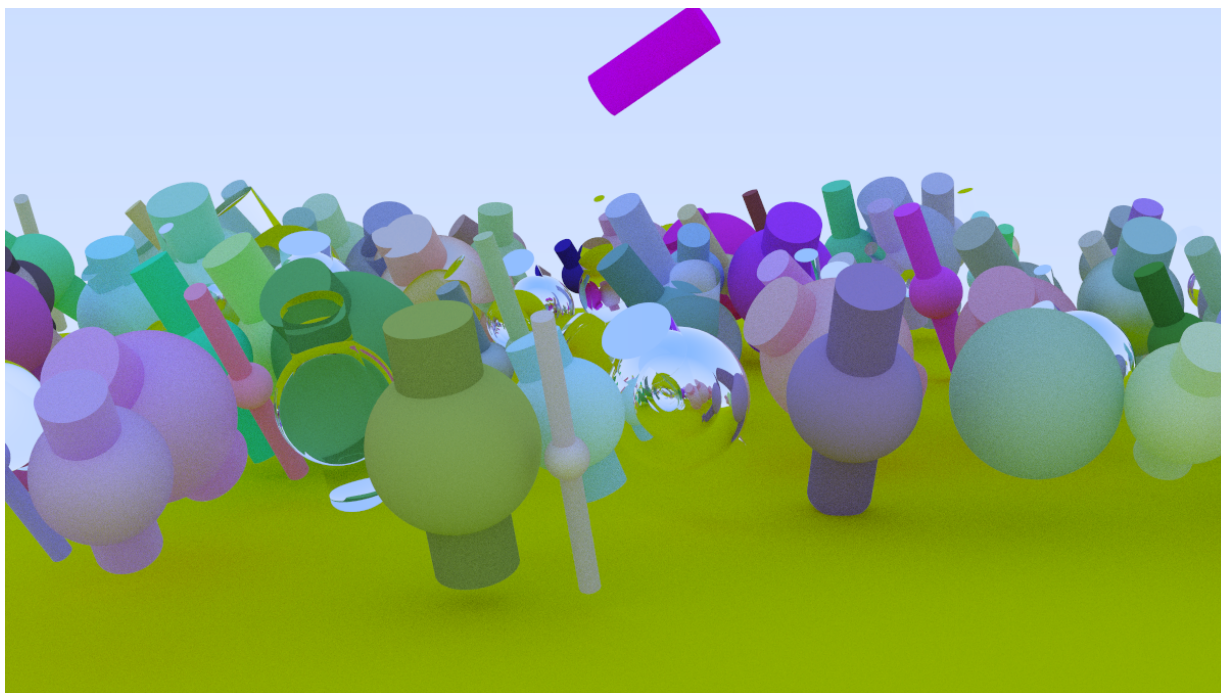


Figura 1: Imagen generada por la aplicación con múltiples esferas y cilindros.

- La longitud de su **radio** r .

Un **cilindro** viene dado por las siguientes propiedades:

- La posición de su **centro** $C \equiv (c_x, c_y, c_z)$. Esta posición se corresponde con el punto medio entre los centros de sus bases superior e inferior.
- El **radio** r del cilindro.
- El vector que define el **eje** del cilindro $\vec{a} = (a_x, a_y, a_z)$.

Ten en cuenta que la altura del cilindro h se puede obtener como el módulo de dicho vector.

$$h = \|\vec{a}\|$$

2.1.2. Materiales de los objetos

Cada objeto tiene asociado un material que define sus propiedades a la hora de realizar la síntesis de imagen. La aplicación considera tres tipos de materiales: **mate**, **metal** y **refractivo**.

Un material **mate** tiene asociada una **reflectancia** que es la relación entre la cantidad de luz reflejada por un objeto con respecto a la luz recibida. La reflectancia se representa como un vector tridimensional con los valores para los canales RGB. Cada uno de los valores se encuentra en el intervalo $[0, 1]$.

Un material **metálico** tiene asociada también una **reflectancia**. Además tiene asociado un factor de **difusión** de la luz que es un valor de tipo real.

Por último, un material **refractivo** tiene asociado un **índice de refracción**, que indica la razón entre los senos del ángulo de incidencia y el ángulo de refracción.

2.1.3. Archivo de descripción de escena

Un archivo de descripción de escena contiene un conjunto de líneas de texto. Cada línea de texto puede tener la especificación de un material o bien la especificación de un objeto.

Especificaciones de material Una línea de especificación de material contiene los siguientes campos separados por blancos:

- **Tipo de material:** Puede ser una de las siguientes etiquetas: **matte:**, **metal:** o **refractive:**. Esta etiqueta indica el tipo de material que se está definiendo.
- **Nombre del material:** Es una única cadena sin blancos que identifica el material definido (p.ej **mat1**, **r512** o **nuevo**).
- **Parámetros adicionales:** Estos parámetros dependen del tipo de material seleccionado:
 - **Mate:** Tres valores numéricos que especifican la reflectancia del material.
 - **Metal:** Cuatro valores numéricos. Los tres primeros especifican la reflectancia del objeto y el cuarto el factor de difusión.
 - **Refractivo:** Un único valor numérico que indica el índice de refracción.

Especificaciones de objetos Una línea de especificación de objeto contiene los siguientes campos separados por blancos:

- **Tipo de objeto:** Puede ser una de las siguientes etiquetas: **sphere:** o **cylinder:**. Esta etiqueta indica el tipo de objeto que se está definiendo.
- **Parámetros del objeto:** Son parámetros que dependen del tipo de objeto
 - **Esfera:** Tres valores numéricos que indican las coordenadas del centro de la esfera y otro valor numérico para el radio.
Si el valor para el radio es menor o igual que cero, se considera que el valor es inválido.
 - **Cilindro:** Tres valores numéricos que indican las coordenadas del centro del cilindro, otro valor numérico para el radio y otros tres valores numéricos para el vector que define el eje del cilindro.
Si el valor para el radio es menor o igual que cero, se considera que el valor es inválido.
- **Nombre del material:** Nombre del material del que está hecho el objeto. Debe ser un material definido con anterioridad en el archivo.

Como ejemplo, el listado 1 presenta el contenido de un archivo de configuración para una escena.

Listado 1: Archivo de configuración para una escena simple

```
matte: mat1 0 0.8 0.8
metal: metal1 0 0.8 0 2.0
refractive: ref99 1.3
sphere: 0 0 0 0.65 mat1
cylinder: 0 0 0 0.5 20 10 -5 metal1
```

Este archivo de configuración de escena contiene tres materiales cuyos nombres son **mat1**, **metall** y **ref99**. También incluye dos figuras: una esfera y un cilindro.

El material **mat1** es un material de tipo **mate** con una reflectancia de $(r = 0, g = 0,8, b = 0,8)$. El material **metall** es un material de tipo **metal** con reflectancia de $(r = 0, g = 0,8, b = 0)$ y factor de difusión de 2,0. Por último, el material **ref99** es un material de tipo **refractivo** con un índice de refracción de 1,3.

La esfera que se incluye en la línea 4, tiene su centro en las coordenadas $C \equiv (c_x = 0, c_y = 0, c_z = 0)$ y un radio $r = 0,65$.

Así mismo, el cilindro que se incluye en la línea 5 tiene su centro en las coordenadas $C \equiv (c_x = 0, c_y = 0, c_z = 0)$, un radio de $r = 0,5$ y como eje el vector $\vec{a}(v_x = 20, v_y = 10, v_z = -5)$. Como consecuencia, su altura es $h = 22,91$.

2.1.4. Tratamiento de errores

Al cargar un archivo de escena deben tenerse en cuenta los posibles errores en el mismo. En caso de encontrarse un error, se notificará el error y se terminará la ejecución.

Etiquetas no reconocidas Si se encuentra un nombre de entidad no válido, se imprimirá un mensaje de error como el siguiente:

```
Error: Unknown scene entity: triangle
```

Estructura de líneas La estructura del archivo está organizada en líneas. De este modo una línea debe contener toda la información de un material o un objeto.

Si una línea está vacía o compuesta exclusivamente por caracteres en blanco y tabuladores, dicha línea se ignora.

Líneas con información insuficiente Si una línea no contiene toda la información necesaria, para el tipo de entidad correspondiente, se emitirá un mensaje de error como el siguiente:

```
Error: Invalid matte material parameters
Line: "matte: mat1 0 0.8 "
```

Líneas con información excesiva Si una línea contiene más información de la necesaria, para el tipo de entidad correspondiente, se emitirá un mensaje de error como el siguiente:

```
Error: Extra data after configuration value for key: [sphere:]
Extra: "3"
Line: "sphere: 0 0 0 0.65 mat1 3"
```

Líneas con información inválida Si una línea contiene información que no es del tipo esperado o tiene un valor fuera del rango admisible, para el tipo de entidad correspondiente, se emitirá un mensaje de error como el siguiente:

```
Error: Invalid sphere parameters
Line: "sphere: 0 0 0 a mat1 3"
```

Nombres de material repetidos Si aparecen dos líneas definiendo un material con el mismo nombre, se emitirá un mensaje de error como el siguiente:

```
Error: Material with name [mat1] already exists
Line: "matte: mat1 0 0.8 0.8"
```

Referencias a materiales no definidos Si un objeto referencia un material que no se ha definido previamente, se emitirá un mensaje de error como el siguiente:

```
Error: Material not found: [metal12]
Line: "cylinder: 0 0 0 0.5 20 10 -5 metal12"
```

2.2. Parámetros de la generación de imagen

La aplicación tiene una serie de parámetros configurables, cada uno de ellos con un valor por defecto.

A continuación se describe cada uno de ellos, y se incluye su valor por defecto, en caso de indicarse en el correspondiente archivo de configuración.

- **Relación de aspecto:** Es la proporción entre la anchura y la altura de una imagen. Se especifica por dos valores enteros positivos. El primero es la anchura y el segundo la altura. Ambos valores deben ser enteros positivos.

Si no se especifica, su valor por defecto es el par (16,9).

Si alguno de los dos valores es menor a igual que cero, se considera que tienen un valor inválido.

Su etiqueta en un archivo de configuración es **aspect_ratio:**.

- **Anchura de imagen:** Es la anchura en píxeles de la imagen que se debe generar.

Si no se especifica, su valor por defecto es 1920.

Si el valor es menor o igual que cero, se considera que tiene un valor inválido.

Su etiqueta en un archivo de configuración es **image_width:**.

Ten en cuenta que la **altura de la imagen** se puede calcular dividiendo la anchura entre la relación de aspecto.

- **Parámetro gamma:** Es el valor del parámetro para la corrección gamma a aplicar sobre la imagen resultante.

Si no se especifica, su valor por defecto es 2,2

Su etiqueta en un archivo de configuración es **gamma:**.

- **Parámetros del punto de vista:** Son parámetros específicos del punto de vista desde el que se observa la escena.

- **Posición** del punto de vista: Son las coordenadas del punto de vista.

Si no se especifica, su valor por defecto es (0,0,−10).

Su etiqueta en un archivo de configuración es **camera_position:**.

- **Destino** de la visión: Son las coordenadas del punto destino de la visión. Es decir el punto hacia el que se está mirando.

Si no se especifica, su valor por defecto es (0,0,0).

Su etiqueta en un archivo de configuración es **camera_target:**.

- **Dirección norte** del punto de vista: Es el vector de dirección que apunta hacia el norte del punto de vista.

Si no se especifica, su valor por defecto es (0,1,0).

Su etiqueta en un archivo de configuración es **camera_north:**.

- **Ángulo de campo de visión:** Es el ángulo que determina el campo de visión, expresado en grados.
Si no se especifica, su valor por defecto es 90.
Si el valor es menor o igual que cero o mayor o igual que 180, se considera que tiene un valor inválido.
Su etiqueta en un archivo de configuración es `field_of_view:`.
- **Muestras por píxel:** Es el número de rayos que se proyectan en el entorno de cada píxel.
Si no se especifica, su valor por defecto es 20.
Si el valor es menor o igual que cero, se considera que tiene un valor inválido.
Su etiqueta en un archivo de configuración es `samples_per_pixel:`.
El número de muestras por píxel tiene impacto en la calidad de la imagen, ya que con un mayor número de muestras por píxel se obtiene una imagen con bordes más suavizados.
- **Profundidad máxima:** Es el número de rebotes de cada rayo sobre la escena.
Si no se especifica, su valor por defecto es 5.
Si el valor es menor o igual que cero, se considera que tiene un valor inválido.
Su etiqueta en un archivo de configuración es `max_depth:`.
Un mayor nivel de profundidad da lugar a una imagen más realista.
- **Semilla para materiales:** Es una semilla para el generador de números aleatorios utilizado para los materiales. Ten en cuenta que todos los materiales comparten un único generador de números aleatorios, que utilizan en los cálculos de reflexión.
Si no se especifica, su valor por defecto es 13.
Si el valor es menor o igual que cero, se considera que tiene un valor inválido.
Su etiqueta en un archivo de configuración es `material_rng_seed:`.
- **Semilla para rayos:** Es una semilla para el generador de números aleatorios utilizado para la generación de rayos.
Si no se especifica, su valor por defecto es 19.
Si el valor es menor o igual que cero, se considera que tiene un valor inválido.
Su etiqueta en un archivo de configuración es `ray_rng_seed:`.
- **Color de fondo oscuro:** Es el valor más oscuro para el fondo. El color de fondo se genera como un gradiente a través de la coordenada vertical entre el color de fondo oscuro y el color de fondo claro.
Si no se especifica, su valor por defecto es (0,25,0,5,1).
Si el valor de alguna de las tres componentes es menor que 0 o mayor que 1, se considera que tiene un valor inválido.
Su etiqueta en un archivo de configuración es `background_dark_color:`.
- **Color de fondo claro:** Es el valor más claro para el fondo. El color de fondo se genera como un gradiente a través de la coordenada vertical entre el color de fondo oscuro y el color de fondo claro.
Si no se especifica, su valor por defecto es (1,1,1).
Si el valor de alguna de las tres componentes es menor que 0 o mayor que 1, se considera que tiene un valor inválido.
Su etiqueta en un archivo de configuración es `background_light_color:`.

El listado 2 muestra un ejemplo de archivo de configuración.

Listado 2: Ejemplo de archivo de configuración config.txt

```
image_width: 1200
gamma: 2.2

camera_position: 13 2 3
camera_target: 0 0 0
camera_north: 0 1 0
field_of_view: 20

samples_per_pixel: 10
max_depth: 5

material_rng_seed: 45
ray_rng_seed: 133

background_dark_color: .25 .5 1
background_light_color: 1 1 1
```

2.2.1. Tratamiento de errores

Si al leer un archivo de configuración se detecta un error se deberá imprimir un mensaje de error y terminar la ejecución del programa.

Etiquetas no reconocidas Todas las etiquetas están formadas por una única palabra terminada con el carácter dos puntos. Si se encuentra una etiqueta que no se reconoce se escribirá un mensaje de error indicando la situación.

Por ejemplo, si se encuentra una línea que comienza con la etiqueta `image_xwidth:`, se presentará el siguiente mensaje de error:

```
Error: Unknown configuration key: [image_xwidth:]
```

Del mismo modo, si se encuentra una línea que comienza con una etiqueta válida, pero que no termina en el carácter dos puntos (como `width`), se presentará el mensaje de error siguiente:

```
Error: Unknown configuration key: [width:]
```

Estructura de líneas Cada parámetro de configuración debe aparecer exactamente en una línea que siempre estará formada por el nombre del parámetro, seguido por su valor. Puede haber caracteres en blanco y tabuladores antes del nombre de la etiqueta, entre los distintos parámetros, y al final de la línea. Sin embargo toda la información de un parámetro debe aparecer en una línea.

Si hay algún error al procesar una línea del archivo con la información de un parámetro se presentará un mensaje de error como el siguiente:

```
Error: Invalid value for key: [aspect_ratio:]
Line: "aspect_ratio: nulo"
```

Parámetros repetidos Si un archivo de configuración contiene varias líneas con la misma etiqueta de parámetros, prevalecerá la información aportada en la última aparición.

En el siguiente ejemplo de archivo de configuración, se tomará para el valor de parámetro `gamma` el valor `2.5`:

```
gamma: 1.8  
image_width: 900  
gamma: 2.5
```

Líneas en blanco Un archivo de configuración puede tener líneas en blanco, que serán ignoradas.

Línea con información no esperada Si una línea del archivo de configuración contiene información innecesaria, se emitirá un mensaje de error.

Por ejemplo, si un archivo de configuración contiene una línea como la siguiente:

```
gamma: 2.1 2.2 99
```

Se emitirá el siguiente mensaje de error:

```
Error: Extra data after configuration value for key: [gamma:]  
Extra: "2.2 99"
```

Línea con información insuficiente Si una línea no contiene toda la información necesaria para un parámetro, se emitirá un mensaje de error indicando que el valor no es válido.

Por ejemplo, si un archivo de configuración contiene una línea como la siguiente:

```
camera_position: 500 500
```

Se emitirá el siguiente mensaje de error:

```
Error: Invalid value for key: [camera_position:]  
Line: "camera_position: 500 500"
```

2.3. Representación de una imagen

El resultado del programa es un archivo que representa una imagen 2D. No obstante, debe distinguirse entre la representación en memoria y la representación en el archivo. En la representación en memoria se implementarán dos alternativas distintas (estructuras de arrays frente a arrays de estructuras). Sin embargo, el archivo de salida estará en el formato PPM, variante P3.

2.3.1. Representación de una imagen en memoria

Una imagen en memoria se puede representar de dos maneras como una **estructura de arrays** o como un **array de estructuras**. Independientemente de la representación en memoria elegida una imagen tiene un tamaño, que viene definido por una anchura (el número de columnas) y una altura (el número de filas). Por convención, el píxel de la fila 0 y columna 0 se corresponde con la esquina superior izquierda de la imagen.

Cada píxel de la imagen se representa mediante un color que viene dado por tres valores para las componentes de rojo (R), verde (G) y azul (B). Cada uno de estos valores es un número entero en el rango de 0 a 255.

En la representación en memoria de tipo **estructura de arrays** la imagen se representa por tres arrays o vectores independientes (uno para cada uno de los tres canales R, G y B). Cada uno de estos tres vectores contiene los valores del canal para todos los píxeles de la imagen.

En la representación en memoria de tipo **array de estructuras** la imagen se representa por un único array o vector. Este array contiene en cada posición los valores R, G y B para el correspondiente píxel.

2.3.2. Representación de una imagen en formato PPM

Cuando se genera una imagen en un archivo se utilizará siempre el formato PPM, en su variante P3. En esta variante el archivo es un archivo de texto compuesto por una cabecera y un cuerpo.

La cabecera del archivo está compuesta por tres líneas:

- La primera línea contiene la cadena **P3** seguida de un salto de línea.
- La segunda línea contiene dos valores numéricos separados por un espacio. Estos valores se corresponden con la anchura (número de columnas) y la altura (número de filas) de la imagen. La línea acaba con un salto de línea.
- La tercera línea contiene el valor **255** seguido de un salto de línea.

El resto del archivo es una sucesión de líneas, cada línea se corresponde con un píxel de la imagen y está compuesto por tres valores numéricos separados por espacios. Los píxeles aparecen en el archivo recorriendo la imagen por filas. Primero aparecen todos los píxeles de la primera línea, después los de la segunda línea, y así sucesivamente.

En el listado 3 se presenta un archivo PPM para una imagen de 3 filas por 2 columnas. El color que corresponde a la fila 1, columna 0, es el color blanco (255, 255, 255). Así mismo, el color que corresponde a la fila 2, columna 1, es el valor azul (0, 0, 255).

Listado 3: Ejemplo de archivo PPM

```
P3
2 3
255
0 0 0
128 128 128
255 255 255
255 0 0
0 255 0
0 0 255
```

2.3.3. Imagen de ejemplo

Como ejemplo del resultado esperado, si procesa la escena presentada en el listado 1 con la configuración presentada en el listado 2, se obtiene una imagen como la de la figura 2.

3. Generación de una imagen

La generación de una imagen tiene tres etapas:

- Generación del **punto de vista**. Este es el lugar en el que se sitúa la cámara que apunta a la escena y que define la ventana de proyección.
- **Trazado de rayos** para cada uno de los píxeles de la imagen sobre la ventana de proyección.
- Determinación de la **contribución al color** de cada rayo trazado.

3.1. Punto de vista de la imagen

Para generar una imagen, además la descripción de la escena es necesario definir un punto de vista. El punto de vista viene dado por tres parámetros: la geometría del punto de vista, el tamaño de la imagen resultante y una semilla de números aleatorios.

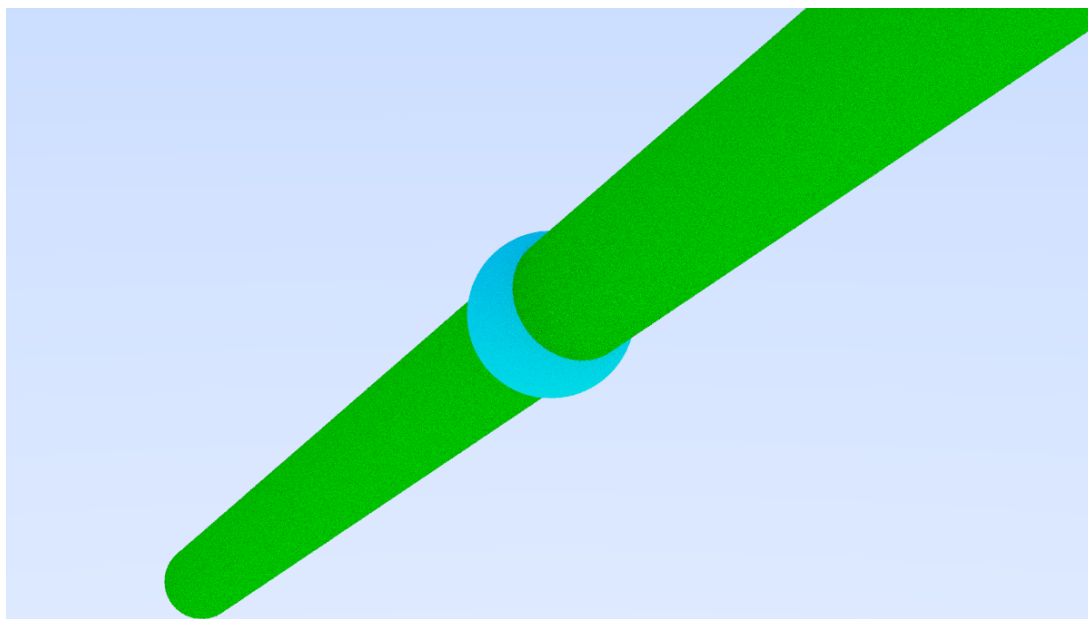


Figura 2: Imagen generada con una esfera y un cilindro.

Geometría del punto de vista La **geometría del punto de vista** viene dada por los siguientes parámetros:

- La **posición** del punto de vista $P \equiv (p_x, p_y, p_z)$. Si no se especifica, su valor por defecto es la posición $(0, 0, -10)$.
- El punto **destino** de la visión $D \equiv (c_x, c_y, c_z)$. Es decir es el punto al que se mira desde la posición del punto de vista. Si no se especifica, su valor por defecto es la posición $(0, 0, 0)$.
- La dirección que indica el **norte** de la visión, y viene dada por el vector $\vec{n} = (n_x, n_y, n_z)$. Ten en cuenta que con una **posición** y un **destino** se podría ver en distintas direcciones. Por tanto este vector define la dirección norte de la imagen que se genera. Si no se especifica, su valor por defecto es $(0, 1, 0)$.
- El **campo de visión** que indica la apertura de la lente con la que se observa la imagen. Este ángulo se expresa en grados. Si no se especifica, su valor por defecto es 90° .

Tamaño de la imagen a generar El **tamaño de la imagen** indica la anchura y altura medido en píxeles de la imagen resultante. Estos valores son importantes para determinar la ventana de proyección, que es un rectángulo en el espacio tridimensional sobre la que se proyectan los objetos.

Generación de números aleatorios El punto de vista también recibe un parámetro de tipo entero que es la **semilla de un generador de números aleatorios** que se utiliza durante el algoritmo de síntesis de imagen. En esta aplicación se utilizará para este proceso un generador *Mersenne-Twister* de 64 bits. En C++, se puede utilizar el tipo `std::mt19937_64` (ver https://en.cppreference.com/w/cpp/numeric/random/mersenne_twister_engine.html).

Cálculo de la ventana de proyección A partir de estos parámetros, se pueden derivar el valor de la ventana de proyección. La ventana de proyección es un rectángulo en el espacio tridimensional que tiene el mismo tamaño que la imagen 2D finalmente generada.

Dicha ventana además de por el tamaño, viene definida por un punto **origen** (coordenadas de la esquina superior izquierda) y los vectores de dirección **horizontal** y **vertical**. Estos vectores determinan el plano de orientación de la ventana.

Para determinar la ventana de proyección se siguen los siguientes pasos:

1. Determinación del **vector focal**. Este es el vector (\vec{v}_f) que va desde el punto de **destino** de la visión hasta la **posición** del punto de vista.

$$\vec{v}_f = P - D$$

2. Determinación de la **distancia focal**. Esta se calcula como la magnitud del **vector focal**.

$$d_f = \|\vec{v}_f\|$$

3. Determinación de la **altura de la ventana de proyección**. Esta altura (h_p) se calcula mediante la siguiente expresión:

$$h_p = 2 \cdot \tan\left(\frac{\alpha}{2}\right) \cdot d_f$$

Donde α es el ángulo del **campo de visión** (convertido a radianes) y d_f es la **distancia focal**.

4. Determinación de la **anchura de la ventana de proyección** (w_p). Como la ventana debe tener la misma proporción entre altura y anchura que la imagen final, la anchura w_p se determina multiplicando la anchura por la **razón de aspecto** (razón entre la anchura y la altura de la imagen en píxeles).

$$w_p = h_p \frac{w}{h}$$

5. Determinación de los **vectores directores** de la ventana de proyección (\vec{u} y \vec{v}).

$$\hat{v}_f = \frac{\vec{v}_f}{\|\vec{v}_f\|} \quad \vec{u} = \frac{\vec{n} \times \hat{v}_f}{\|\vec{n} \times \hat{v}_f\|} \quad \vec{v} = \hat{v}_f \times \vec{u}$$

donde \vec{n} es el vector que marca la dirección **norte**.

6. Determinación de los vectores **horizontal** (\vec{p}_h) y **vertical** (\vec{p}_v). Estos vectores se obtienen multiplicando los vectores unitarios \vec{u} y \vec{v} por la anchura w_p y altura h_p de la ventana de proyección respectivamente. En el caso del vector vertical, además debe tenerse en cuenta que en una imagen la coordenada de la esquina superior izquierda tiene el valor 0 y sus valores crecen hacia abajo, lo que se corrige cambiando el signo del vector correspondiente.

$$\vec{p}_h = w_p \cdot \vec{u} \quad \vec{p}_v = h_p \cdot (-\vec{v})$$

7. Determinación del **origen de la ventana de proyección**. El origen O parte de la **posición** del punto de vista y se desplaza restando el **vector focal** (con lo que se llega) a la posición del **punto destino**. Como ese punto está en el centro de la **ventana de proyección**, se realiza un desplazamiento negativo de la mitad de los vectores \vec{p}_h y \vec{p}_v para llegar a la esquina superior izquierda de la ventana de proyección. A partir de aquí, para llegar al centro del píxel superior izquierdo, se suma la mitad de los vectores de desplazamiento Δx y Δy . Estos últimos se obtienen dividiendo los vectores \vec{p}_h y \vec{p}_v por las dimensiones de la imagen en píxeles.

$$\Delta x = \frac{\vec{p}_h}{w} \quad \Delta y = \frac{\vec{p}_v}{h} \quad O = P - \vec{v}_f - \frac{1}{2} \cdot (\vec{p}_h + \vec{p}_v) + \frac{1}{2} \cdot (\Delta x + \Delta y)$$

3.2. Proceso de trazado de rayos

Para generar una imagen 2D se procede a trazar rayos que parten del punto de vista y pasan por cada píxel de la ventana de proyección hasta que colisionan con un objeto.

Ten en cuenta que cada píxel en la ventana de proyección, no se corresponde con un punto sino con un rectángulo que tiene una anchura $\|\Delta x\|$ y una altura $\|\Delta y\|$. Para mejorar la calidad de la imagen en vez de trazar un único rayo se determina un conjunto de rayos que pasan en posiciones aleatorias dentro del rectángulo del píxel. El número de rayos que se lanzan por cada píxel se denomina **número de muestras por píxel**.

Dada una posición en la imagen final determinada por su coordenada (f, c) se genera una posición Q en la ventana de proyección. Para ello, se generan primeramente dos valores aleatorios δ_x y δ_y , ambos en el intervalo $[-\frac{1}{2}, \frac{1}{2}]$. A continuación, se calcula la posición Q como una posición aleatoria en los siguientes intervalos:

$$Q = P + \vec{\Delta x} \cdot (c + \delta_x) + \vec{\Delta y} \cdot (f + \delta_y)$$

Para cada una de las posiciones Q_i generadas en el rectángulo definido por el píxel, se traza un rayo cuyo origen es la **posición** del punto de vista P y que pasa por la posición Q_i . De esta manera un rayo puede verse como una semi-recta que tiene un origen P y un vector de dirección \vec{d}_r .

Para cada rayo generado se calcula el color obtenido al proyectar el rayo sobre la escena. Este color se obtiene como un valor RGB donde cada componente se encuentra en el espacio de los números reales $[0, 1]$. Posteriormente, las contribuciones de todos los rayos que se corresponden con un mismo píxel se promedian, siendo este valor promedio el valor que se asigna a dicho píxel.

Antes de generar el valor final en la escala de 0 a 255, se deben realizar dos operaciones finales:

1. Aplicación de **corrección gamma**. Cada valor de intensidad en el rango $[0, 1]$ se eleva al exponente $\frac{1}{\gamma}$. Esta corrección mejora la calidad de la imagen.
2. Escalado de **intensidad**. Cada valor en el rango $[0, 1]$ obtenido después de la corrección gamma se escala al intervalo discreto $[0, 255]$. De esta manera al valor original 0 le corresponde un valor discreto de 0 y al valor original 1 le corresponde un valor discreto de 255. Si es necesario, se trunca el valor obtenido.

3.3. Contribución al color de un rayo

Cuando se traza un rayo desde el punto de vista hasta la escena, se incorpora una **profundidad**. Este valor es el número de rebotes que el rayo produce en otros objetos. Es decir, si la profundidad es 1 solamente se tiene en cuenta la colisión del rayo con el primer objeto. Sin embargo, si esta profundidad fuese mayor que 1 se tendrán en cuenta los rebotes posteriores del rayo con otros objetos. Estos rebotes dependerán del material del objeto con el que el rayo colisiona.

De esta manera, hay tres elementos que intervienen en la proyección del rayo: el propio **rayo**, la **escena** con los objetos y el nivel de **profundidad** con el que se realiza el cálculo. Con todo esto se calcula la contribución al color de cada colisión de un rayo.

- Cuando la profundidad es menor o igual que 0 no hay ninguna contribución al color.
- Si la profundidad del rayo es positiva, se calcula la **intersección** del rayo con la escena.
 - Si **varios objetos** tienen intersección con el rayo, se selecciona la **intersección más cercana al punto de vista**.

- Una vez determinada la intersección, se calcula la **contribución** al color correspondiente a esta **intersección**.
- Si no se produce intersección del rayo con ningún objeto de la escena se genera como **contribución el color de fondo**.

3.4. Intersección de un rayo con la escena

Un rayo puede tener una intersección con varios objetos de la escena. En este caso se calculará la intersección del rayo con cada objeto de la escena manteniendo en cada iteración la información de la intersección más cercana y la distancia desde el origen del rayo a dicha intersección. En cualquier caso no se tendrán en cuenta intersecciones a una distancia menor que 10^{-3} .

El cálculo de la intersección de un rayo con un objeto depende del tipo de objeto concreto. En este proyecto se consideran las intersecciones con **esferas** y **cilindros**.

3.4.1. Intersección de un rayo con una esfera

Todos los puntos de un rayo se pueden representar mediante la expresión $P(\lambda)$, como la posición del origen el rayo O_r sumada con el vector de dirección \vec{d}_r multiplicado por el valor del parámetro λ :

$$P(\lambda) = O_r + \vec{d}_r \lambda$$

Por otra parte, si un punto $P(\lambda)$ se encuentra en la superficie de una esfera de centro C y radio r debe cumplirse que:

$$\|C - P(\lambda)\|^2 = r^2$$

Combinando estas dos expresiones y recordado la relación entre módulo y producto escalar ($\|v\|^2 = \vec{v} \cdot \vec{v}$), se tiene la siguiente expresión:

$$(C - (O_r + \vec{d}_r \lambda)) \cdot (C - (O_r + \vec{d}_r \lambda)) = r^2$$

Si se define el vector \vec{r}_c como el vector que va del origen del rayo O_r hasta el centro de la esfera (es decir, $\vec{r}_c = C - O_r$), se tiene:

$$(\vec{r}_c - \vec{d}_r \lambda) \cdot (\vec{r}_c - \vec{d}_r \lambda) = r^2$$

Esta ecuación vectorial también se puede representar como:

$$\vec{d}_r \cdot \vec{d}_r \lambda^2 - 2\vec{d}_r \cdot \vec{r}_c \lambda + \vec{r}_c \cdot \vec{r}_c - r^2 = 0$$

Que es una ecuación de segundo grado en la que:

$$a = \vec{d}_r \cdot \vec{d}_r \quad b = 2\vec{d}_r \cdot \vec{r}_c \quad c = \vec{r}_c \cdot \vec{r}_c - r^2$$

Y para calcular los valores de λ se aplica la solución de la ecuación de segundo grado:

$$\lambda = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Deben tenerse en cuenta tres casos dependiendo del valor del discriminante:

- Si el discriminante es negativo, no hay ninguna intersección entre el rayo y la esfera.

- Si el discriminante es 0, hay un único punto de intersección.
- Si el discriminante es positivo, hay dos puntos de intersección.

Ten en cuenta que solamente debe considerarse el punto de intersección más cercano al punto de vista y solamente si no se ha encontrado otra intersección más cercana con otro objeto.

En caso de localizarse un punto de intersección, se calculan además:

- La posición en el espacio del **punto de intersección**, que se obtiene evaluando la expresión vectorial $I = O_r + \vec{d}_r \lambda$.
- El **vector normal** a la esfera en el punto de intersección, que se obtiene evaluando la expresión vectorial $\vec{d}_n = (I - C)/r$.
- La **longitud del rayo** hasta el punto de intersección, que viene dada por el valor λ .
- El **sentido** del vector normal que puede ser hacia **afuera** o hacia **adentro** de la esfera. Si el producto escalar de la dirección del rayo \vec{d}_r y el vector normal \vec{d}_n es negativo, el sentido es hacia **afuera**. En otro caso es hacia **adentro**.

Ten en cuenta que si el **sentido** es **hacia adentro**, se debe cambiar el signo del **vector normal**.

3.4.2. Intersección de un rayo con un cilindro

El cálculo de la intersección con un cilindro tiene dos etapas:

1. Cálculo de la intersección más cercana con la superficie curva del cilindro.
2. Cálculo de las intersecciones con cada una de las dos bases del cilindro si alguna de ellas es más cercana que la intersección ya encontrada.

Para realizar los cálculos se parte de los siguientes valores:

- C : Punto que es el centro del cilindro.
- r : Radio del cilindro.
- \hat{a} : Vector unitario de la dirección del eje del cilindro.
- h : Altura del cilindro.

Intersección con la superficie curva del cilindro La intersección del rayo con la superficie curva del cilindro puede dar lugar a cero intersecciones (no hay intersección), una intersección (el rayo es tangente al cilindro) o dos intersecciones (el rayo corta la superficie curva del cilindro).

En primer lugar se calcula la intersección del rayo con un cilindro equivalente pero de altura infinita.

Para ello se parte del vector \vec{r}_c que va del origen del rayo O_r hasta el centro del cilindro C (es decir, $\vec{r}_c = O_r - C$).

Ten en cuenta que para cualquier vector \vec{v} se puede calcular su componente perpendicular la eje \hat{a} como:

$$\vec{v}_{\perp \hat{a}} = \vec{v} - (\vec{v} \cdot \hat{a})\hat{a}$$

Por otra parte, cualquier punto P en el rayo de origen O_r y dirección \vec{d}_r se puede expresar en función del parámetro λ como:

$$P(\lambda) = O_r + \vec{d}_r \lambda$$

Para que un punto $P(\lambda)$ esté en la superficie del cilindro debe cumplirse que:

$$\|(P(\lambda) - C)_{\perp \hat{a}}\|^2 = r^2$$

En el caso de un punto del rayo $P(\lambda)$ (que es $O_r + \vec{d}_r \lambda$)

$$\|(O_r + \vec{d}_r \lambda - C)_{\perp \hat{a}}\|^2 = r^2$$

Teniendo en cuenta que $\vec{r}_c = O_r - C$

$$\|(\vec{r}_c + \vec{d}_r \lambda)_{\perp \hat{a}}\|^2 = r^2$$

Aplicando la definición de componente perpendicular al eje \hat{a} , se tiene:

$$\|(\vec{r}_c + \vec{d}_r \lambda) - ((\vec{r}_c + \vec{d}_r \lambda) \cdot \hat{a})\hat{a}\|^2 = r^2$$

Que se puede expresar como:

$$\|\vec{r}_c - (\vec{r}_c \cdot \hat{a})\hat{a} + \vec{d}_r \lambda - (\vec{d}_r \lambda \cdot \hat{a})\hat{a}\|^2 = r^2$$

Teniendo en cuenta la definición de componente perpendicular al eje \hat{a} , se puede expresar como:

$$\|\vec{r}_{c \perp \hat{a}} + \vec{d}_{r \perp \hat{a}} \lambda\|^2 = r^2$$

Aplicando la relación entre producto escalar y norma se tiene:

$$\|\vec{d}_{r \perp \hat{a}}\|^2 \lambda^2 + 2(\vec{r}_{c \perp \hat{a}} \cdot \vec{d}_{r \perp \hat{a}}) \lambda + \|\vec{r}_{c \perp \hat{a}}\|^2 - r^2 = 0$$

Y se obtiene una ecuación de segundo grado en la que los coeficientes a , b y c son:

$$a = \|\vec{d}_{r \perp \hat{a}}\|^2 \quad b = 2(\vec{r}_{c \perp \hat{a}} \cdot \vec{d}_{r \perp \hat{a}}) \quad c = \|\vec{r}_{c \perp \hat{a}}\|^2 - r^2$$

Y para calcular los valores de λ se aplica la solución de la ecuación de segundo grado:

$$\lambda = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Deben tenerse en cuenta tres casos dependiendo del valor del discriminante:

- Si el discriminante es negativo, no hay ninguna intersección entre el rayo y la superficie del cilindro.
- Si el discriminante es 0, hay un único punto de intersección.
- Si el discriminante es positivo, hay dos puntos de intersección.

Ten en cuenta que solamente debe considerarse el punto de intersección más cercano al punto de vista y solamente si no se ha encontrado otra intersección más cercana con otro objeto.

En caso de localizarse un punto de intersección, se calculan además:

- La posición en el espacio del **punto de intersección**, que se obtiene evaluando la expresión vectorial $I = O_r + \vec{d}_r \lambda$.

Debe comprobarse que el punto de intersección se encuentra dentro de los límites del cilindro. Para ello se calcula el vector que va del punto de intersección I al centro C y se calcula su producto escalar con el eje unitario \hat{a} $((I - C) \cdot \hat{a})$. Si la distancia obtenida es mayor de $\frac{h}{2}$ se descarta la intersección.

- El **vector normal** al cilindro en el punto de intersección, que se obtiene evaluando la expresión vectorial

$$\vec{d}_n = (I - C)_{\perp \hat{a}} = (I - C) - ((I - C) \cdot \hat{a})\hat{a}$$

- La **longitud del rayo** hasta el punto de intersección, que viene dada por el valor λ .
- El **sentido** del vector normal que puede ser hacia **afuera** o hacia **adentro** del cilindro. Si el producto escalar de la dirección del rayo \vec{d}_r y el vector normal \vec{d}_n es negativo, el sentido es hacia **afuera**. En otro caso es hacia **adentro**.

Ten en cuenta que si el **sentido** es **hacia adentro**, se debe cambiar el signo del **vector normal**.

Intersección con las bases del cilindro La intersección con cada una de las bases del cilindro también debe considerarse. Para ello se parte de los planos definidos por cada una de las bases. Cada plano está definido por un punto y un vector normal a dicho plano:

- La **base superior** está definida por:

- El **punto** $P = C + \frac{h}{2}\hat{a}$.
- El **vector normal** $\vec{p}_n = \hat{a}$.

- La **base inferior** está definida por:

- El **punto** $P = C - \frac{h}{2}\hat{a}$.
- El **vector normal** $\vec{p}_n = -\hat{a}$.

Para cada plano, se debe determinar la longitud de la intersección del rayo con el plano correspondiente:

- Se determina el vector que va del origen del rayo O_r hasta el punto del plano P como $\vec{r}_p = P - \vec{o}_r$.
- Se calcula la distancia d_p como:

$$\lambda = \frac{\vec{r}_p \cdot \vec{p}_n}{\vec{r}_d \cdot \vec{p}_n}$$

Ten en cuenta que si el valor absoluto del denominador es muy pequeño (menor que 10^{-8}) se considera que no hay intersección porque se obtendría una distancia muy elevada.

Si la distancia obtenida es más cercana que otras distancias previamente obtenidas, se procede a calcular la intersección.

- La posición en el espacio del **punto de intersección**, que se obtiene evaluando la expresión vectorial $I = O_r + \vec{d}_r\lambda$.

Si la distancia entre el punto de intersección I y el punto del plano P es mayor que el radio r se considera que no hay intersección.

- El **vector normal** a la base es \vec{p}_n .
- La **longitud del rayo**, que viene dada por el valor λ .
- El **sentido** del vector normal que puede ser hacia **afuera** o hacia **adentro** del cilindro. Si el producto escalar de la dirección del rayo \vec{d}_r y el vector normal \vec{d}_n es negativo, el sentido es hacia **afuera**. En otro caso es hacia **adentro**.

Ten en cuenta que si el **sentido** es **hacia adentro**, se debe cambiar el signo del **vector normal**.

3.5. Contribución del color en una intersección

Cuando se ha determinado la intersección más cercana de un rayo con la escena el color se corresponde con la intersección. Esto se hace en varios pasos:

- **Generación** de un **nuevo rayo** r_n , que tiene como origen O_r la posición de la intersección I y como vector de dirección \vec{d}_r el vector determinado por la reflexión dependiendo del tipo de material.
- **Cálculo** del **color reflejado** por el nuevo rayo en la escena decrementando el nivel de profundidad en una unidad.
- **Atenuación** del color reflejado obtenido en el paso anterior con la reflectancia del material de la intersección actual.

$$c_x = c_x \cdot r_x$$

$$c_y = c_y \cdot r_y$$

$$c_z = c_z \cdot r_z$$

3.5.1. Reflexión en materiales mate

En el caso de un material mate, el rayo reflejado se genera en una dirección aleatoria, para ello se calcula la dirección de reflexión \vec{d}_r sumando al vector normal \vec{d}_n de la intersección un valor aleatorio entre -1 y 1 en cada una de sus tres componentes.

Cabe la posibilidad de que la dirección resultante sea demasiado pequeña. Se considera que un vector v es demasiado pequeño si se cumplen las condiciones (deben cumplirse todas):

$$|v_x| < 10^{-8}$$

$$|v_y| < 10^{-8}$$

$$|v_z| < 10^{-8}$$

En ese caso, la dirección de reflexión \vec{d}_r es el vector normal \vec{d}_n .

Como resultado de la reflexión se tiene:

- La **reflectancia** resultante es la del material mate.
- La **dirección** del nuevo rayo es \vec{d}_r .

3.5.2. Reflexión en materiales metálicos

En el caso de un material metálico la dirección de reflexión \vec{d}_r se calcula a partir de la dirección del rayo original \vec{d}_o y del vector normal \vec{d}_n de la intersección.

En primer lugar se calcula la dirección de reflexión inicial \vec{d}_r^1 :

$$\vec{d}_r^1 = \vec{d}_o - 2(\vec{d}_o \cdot \vec{d}_n)\vec{d}_n$$

A continuación, el vector de reflexión final se calcula sumando un vector de difusión a partir del factor de difusión del material Φ . Para ello se genera un vector aleatorio $\vec{\phi}$ en el que cada componente toma un valor entre $-\Phi$ y Φ . El vector de reflexión \vec{d}_r se calcula sumando a la dirección de reflexión inicial (normalizada) el vector de difusión:

$$\vec{d}_r = \frac{\vec{d}_r^1}{\|\vec{d}_r^1\|} + \vec{\phi}$$

Como resultado de la reflexión se tiene:

- La **reflectancia** resultante es la del material metálico.
- La **dirección** del nuevo rayo es \vec{d}_r .

3.5.3. Reflexión en materiales refractivos

En el caso de un material refractivo debe determinarse el ángulo de refracción θ a partir del vector unitario de dirección del rayo original \hat{d}_o (que se calcula como $\frac{\vec{d}_o}{\|\vec{d}_o\|}$) y el vector normal de la intersección \vec{d}_n :

$$\cos \theta = \min(-\hat{d}_o \cdot \vec{d}_n, 1) \quad \sin \theta = \sqrt{1 - \cos^2 \theta}$$

Al determinarse la dirección del nuevo rayo, se utiliza el índice refracción corregido:

$$\rho' \begin{cases} \rho & \text{si dirección hacia afuera} \\ \frac{1}{\rho} & \text{si dirección hacia adentro} \end{cases}$$

Una vez corregido el índice de refracción deben considerarse dos casos:

- El valor $\rho' \cdot \sin \theta$ es mayor que 1:

- La dirección de reflexión se obtiene como

$$\vec{d}_r = \hat{d}_o - 2(\hat{d}_o \cdot \vec{d}_n)\vec{d}_n$$

- El valor $\rho' \cdot \sin \theta$ es menor o igual que 1:

- La dirección de reflexión se obtiene a partir de la suma de dos vectores \vec{u} y \vec{v} :

$$\vec{u} = \rho'(\hat{d}_o + (\cos \theta)\vec{d}_n) \quad \vec{v} = -(\sqrt{1 - \|\vec{u}\|^2})\vec{d}_n$$

Como resultado de la reflexión se tiene:

- La **reflectancia** resultante es siempre (1, 1, 1). Es decir, del 100 % en todos sus componentes
- La **dirección** del nuevo rayo es \vec{d}_r .

3.6. Color de fondo

El color de fondo se calcula como un combinación entre dos colores: el **color de fondo oscuro** \vec{c}_d y el **color de fondo claro** \vec{c}_l . El porcentaje de cada uno de estos dos colores en la mezcla depende de la coordenada Y .

Para ello se determina el vector unitario de la dirección del rayo que se está trazando:

$$\hat{d}_r = \frac{\vec{d}_r}{\|\vec{d}_r\|}$$

La componente Y de este vector unitario \hat{d}_{ry} será un valor en el intervalo $[-1, 1]$. Se puede utilizar como factor de mezcla m el promedio entre este valor y el valor máximo que sería 1.

$$m = \frac{\hat{d}_{ry} + 1}{2}$$

El color resultante para el fondo se compone mezclando los colores \vec{c}_d y \vec{c}_l usando el factor m :

$$\vec{c} = (1 - m) \cdot \vec{c}_l + m \cdot \vec{c}_d$$

4. Tareas

4.1. Aplicación a desarrollar

Se desarrollarán dos aplicaciones con distintas estrategias de implementación. La aplicación **render-soa** utilizará la estrategia de implementación SOA (*structure of arrays* o estructuras de arrays) y la aplicación **render-aos** utilizará la estrategia de implementación AOS (*arrays of structures* o arrays de estructuras). Ambas aplicaciones tendrán una interfaz de línea de mandatos.

4.1.1. Parámetros de la aplicación

La aplicación tomará los siguientes parámetros:

- Ruta del archivo de configuración.
- Ruta del archivo de escena.
- Ruta del archivo de salida.

4.1.2. Análisis de los argumentos de la aplicación

Si el número de argumentos recibidos por la aplicación no es exactamente 3, se generará un mensaje de error y el programa terminará con un código de error.

```
$ render
Error: Invalid number of arguments: 0
$ render cfg.txt
Error: Invalid number of arguments: 1
$ render cfg.txt scn.txt
Error: Invalid number of arguments: 2
$ render cfg.txt scn.txt out.ppm out2
Error: Invalid number of arguments: 4
```

4.2. Desarrollo de software

4.2.1. Programa a desarrollar

Esta tarea consiste en el desarrollo de una versión secuencial de la aplicación descrita utilizando C++23. Por favor, ten en cuenta que en esta versión no se permite la utilización de varios hilos de ejecución (*multithreading*) ni de paralelismo de ninguna clase.

Se desarrollarán dos programas ejecutables independientes: **render-soa** e **render-aos** que implementarán respectivamente las estrategias **SOA** y **AOS**.

- **SOA – Structure of Arrays**: Se representarán los píxeles de una imagen como tres secuencias independientes. Cada una de las secuencias contendrá elementos que deberán estar en el rango de **0** a **255**.
- **AOS – Arrays of Structures**: Se representarán los píxeles de una imagen como una única secuencia de valores. Cada valor de la secuencia estará formada por tres campos que deberán estar en el rango de **0** a **255**.

Adicionalmente, se valorarán otros puntos del programa donde el uso de estructuras de arrays pueda mejorar el rendimiento.

Recomendaciones

- Evalúa los distintos contenedores de secuencia que ofrece la biblioteca estándar. Puedes encontrar una lista de los mismos en <https://en.cppreference.com/w/cpp/container>.
- Identifica los tipos enteros más apropiados para cada contexto ofrecidos por el lenguaje (https://en.cppreference.com/w/cpp/language/types#Integral_types) y por la biblioteca estándar (<https://en.cppreference.com/w/cpp/types/integer>).

4.2.2. Estructura del proyecto

En el repositorio de GitHub <https://github.com/comparchuc3m/render-2025-template> encontrarás una plantilla para tu proyecto. Puedes generar tu proyecto inicial a partir de este proyecto.

Este repositorio contiene las siguientes características:

- El directorio **.devcontainer** contiene los archivos de configuración de un contenedor de desarrollo de Docker que te permitirá generar un contenedor con todas las herramientas de desarrollo necesarias.
- El directorio **cmake** contiene un archivo con utilidades adicionales que te serán útiles para realizar análisis de cobertura de tus pruebas unitarias.
- Los directorios **common**, **aos** y **soa** están diseñados para que incluyas ahí el código fuente que debes desarrollar.
- Los directorios **utcommon**, **utaos** y **utsoa** están diseñados para que incluyas ahí el código fuente de tus pruebas unitarias.
- En el directorio raíz encontrarás, entre otros los siguientes archivos:
 - Archivos de configuración **.clang-format** y **.clang-tidy**. Por favor, no modifiques estos archivos a menos que recibas indicaciones explícitas en este sentido.
 - Archivo **CMakeLists.txt** con las instrucciones de construcción del proyecto.
 - Archivo **CMakePresets.json** con la configuración predefinida de opciones de compilación.

Configuración del proyecto: archivo principal de CMake El archivo principal **CMakeLists.txt** establece las siguientes opciones:

- Versión mínima de CMake: 4.0.
- Lenguajes a usar en el proyecto: exclusivamente C++.
- No se admite ninguna biblioteca externa, con la excepción de las siguientes bibliotecas:
 - **GSL** (C++ Core Guideline Support Library). Esta biblioteca ofrece utilidades que extienden la biblioteca estándar y que pueden ser útiles.
 - **GoogleTest**. Esta biblioteca podrás utilizarla para preparar tus pruebas unitarias.

Durante el proceso de compilación estas bibliotecas se descargan y se configuran.

- Se establece una opción **ENABLE_CLANG_TIDY** que se puede activar y desactivar. Cuando esta opción está activa se aplican las reglas establecidas por la herramienta **clang-tidy** a todo el código fuente del proyecto.

- Se añaden los directorios **common**, **soa** y **aos**, así como los correspondientes directorios de pruebas unitarias **utcommon**, **utaos** y **utsoa**.

IMPORTANTE

Todos los archivos fuente deben compilar sin problemas y no emitirán ninguna advertencia del compilador.

También deben compilar sin ningún problema cuando se active la opción **ENABLE_CLANG_TIDY**.

Configuración del proyecto: Archivos de *presets* El archivo **CMakePresets.json**, que también se suministra para su descarga, contiene varios conjuntos de opciones pre-establecidas:

- Opciones de configuración:
 - default**: Es la configuración por defecto que se utiliza para compilar sin ejecutar **clang-tidy**. La compilación será algo más rápida, pero no se realiza análisis estático de código.
 - clang-tidy**: Añade sobre la configuración por defecto la activación de **ENABLE_CLANG_TIDY**. La compilación será algo más lenta, pero se realiza también análisis estático de código.
- Opciones de construcción:
 - gcc-debug**: Compilación en modo de depuración con opciones por defecto.
 - gcc-release**: Compilación en modo optimizado con opciones por defecto.
 - clang-tidy-debug**: Compilación en modo de depuración con opciones por defecto. Además realiza análisis estático de código.
 - clang-tidy-release**: Compilación en modo optimizado con opciones por defecto. Además realiza análisis estático de código.

Las opciones que comparten todos los perfiles de configuración y compilación establecidas en este archivo son las siguientes:

- Uso del generador interno de compilación **Ninja-Multiconfig**.
- Desactivación de opciones de compilación con módulos en C++.
- Uso del compilador **g++-14**.
- Compilación en modo C++23 estricto.
- Activación de las opciones de advertencia y error para una compilación más estricta.

Compilación del proyecto La compilación se puede realizar en modo **Debug** (objetivos **gcc-debug** y **clang-tidy-debug**) o en modo **Release** (objetivos **gcc-release** y **clang-tidy-release**).

IMPORTANTE

Recuerda que todas las evaluaciones se realizarán con las optimizaciones del compilador activadas, utilizando el perfil de compilación **gcc-release**.

Si lo estimas oportuno, puedes añadir opciones de optimización adicionales a dicho perfil, modificando el archivo **CMakePresets.json** e indicándolo en la memoria del proyecto.

Reglas de calidad de código El código fuente debe estar bien estructurado y organizado, así como apropiadamente documentado. Se recomienda (aunque no se requiere) seguir las **C++ Core Guidelines** (<http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines>).

No obstante, si deben seguirse todas las reglas especificadas en el documento **Reglas de codificación para el lenguaje C++** que se publica separadamente.

Para facilitar, el trabajo de los equipos, se suministrará también un archivo de configuración para la herramienta **clang-tidy**.

4.2.3. Componentes software

El desarrollo realizado se estructura en tres componentes software, que se colocan en las carpetas ya existentes **common**, **aos** y **soa**.

Carpeta common Esta carpeta contiene todos los archivos fuente con los tipos de datos y funciones que son comunes a las dos versiones del programa. A su vez, contiene dos subdirectorios **include** (con todos los archivos de cabecera) y **src** (con todos los archivos **.cpp**).

El archivo **CMakeLists.txt** define una biblioteca llamada **common**. En el código que se suministra, la directiva **target_sources** depende solamente del archivo **src/vector.cpp**. Deberás añadir a esta lista todos los archivos fuente que quieras incluir en la biblioteca **common**.

El objetivo **common** utiliza como directorio de inclusión **include**. Además, se establece dependencia de la biblioteca **GSL**.

Carpeta aos Esta carpeta contiene todos los archivos fuente con los tipos de datos y funciones que son específicos de la versión **AOS**. funciones que son comunes a las dos versiones del programa. A su vez, contiene dos subdirectorios **include** (con todos los archivos de cabecera) y **src** (con todos los archivos **.cpp**).

Además, en el directorio **src** debe haber un archivo **main.cpp** que contendrá exclusivamente la función **main()**.

El archivo **CMakeLists.txt** define un ejecutable llamado **render-aos**. En el código que se suministra, la directiva **target_sources** depende solamente del archivo **src/main.cpp**. Deberás añadir a esta lista todos los archivos fuente que quieras incluir en el ejecutable **render-aos**.

El objetivo **render-aos** utiliza como directorio de inclusión **include**. Además, se establece dependencia de las bibliotecas **GSL** y **common**.

Carpeta soa Esta carpeta contiene todos los archivos fuente con los tipos de datos y funciones que son específicos de la versión **SOA**. funciones que son comunes a las dos versiones del programa. A su vez, contiene dos subdirectorios **include** (con todos los archivos de cabecera) y **src** (con todos los archivos **.cpp**).

Además, en el directorio **src** debe haber un archivo **main.cpp** que contendrá exclusivamente la función **main()**.

El archivo **CMakeLists.txt** define un ejecutable llamado **render-soa**. En el código que se suministra, la directiva **target_sources** depende solamente del archivo **src/main.cpp**. Deberás añadir a esta lista todos los archivos fuente que quieras incluir en el ejecutable **render-soa**.

El objetivo **render-soa** utiliza como directorio de inclusión **include**. Además, se establece dependencia de las bibliotecas **GSL** y **common**.

4.2.4. Pruebas unitarias

Se definirá un conjunto de pruebas unitarias que también se entregarán. Se recomienda, el uso de Google-Test (<https://github.com/google/googletest>). Si se desea utilizar otro marco de trabajo para pruebas unitarias se deberá obtener la autorización del coordinador de la asignatura.

En cualquier caso, se entregarán evidencias de que hay pruebas unitarias suficientes.

Pruebas utcommon El directorio **utcommon** contiene todas las pruebas unitarias que se corresponden con los componentes de la biblioteca **common**.

En **CMakeLists.txt** se definen dos variables:

- **COMMON_SRC_FILES**: Contiene la lista de archivos fuente que se están probando.
- **CURRENT_DIR_SRC_FILES**: Contiene la lista de archivos fuente que contiene las pruebas que deben ejecutarse.

Como ejemplo, se presenta un archivo de pruebas **test_vector.cpp** con pruebas para el tipo **vector**.

Pruebas utaos y utsoa Contienen las pruebas para los correspondientes componentes **aos** y **soa**, siguiendo una estructura similar.

4.2.5. Uso de herramientas de IA

El uso de herramientas basadas en IA durante el proyecto está permitido. No obstante, se debe tener en cuenta lo siguiente:

- Si usas una herramienta basada en IA, se deben declarar los usos en la sección de diseño de la memoria del proyecto. No se realizará ninguna penalización en la declaración siempre que se incluya una declaración de uso.
- No se dará soporte al uso de dichas herramientas. En particular, si tienes preguntas sobre tu código, deberás ser capaz de explicar dicho código.
- Ten en cuenta que algunas herramientas de IA pueden generar código inseguro o código con bajo rendimiento.
- Cualquiera de tus profesores podrá requerirte una explicación de tu propio código.

4.3. Evaluación del rendimiento y la energía

Esta tarea consiste en la realización de una evaluación comparativa del rendimiento y el consumo energético. Para llevar a cabo la evaluación del rendimiento, se medirá el tiempo total de ejecución con la herramienta **perf**. Además, también se medirá la energía y se derivará la potencia.

Todas las evaluaciones del rendimiento se realizarán en un nodo del clúster **avignon**.

Representa en una gráfica los tiempos totales de ejecución, uso de energía y potencia para distintas imágenes. Realiza un análisis de escalabilidad para aquellas opciones de la aplicación para las que tenga sentido.

La memoria del proyecto incluirá conclusiones derivadas de los resultados. Por favor, no te limites a una mera descripción de los datos. Trata de encontrar explicaciones convincentes de estos resultados.

5. Estructura de la memoria a entregar

La memoria deberá limitarse a describir el trabajo realizado. Se valorará especialmente la concisión en las descripciones. Deberá entregarse en formato PDF. Deberá contener, al menos, las siguientes secciones:

- **Página de título**: contendrá los siguientes datos:

- Nombre de la práctica.
 - Nombre del grupo reducido en el que están matriculados los estudiantes.
 - Número de equipo asignado.
 - Nombre y NIA de todos los autores.
- **Diseño.** Deberá explicar el diseño global de la aplicación complementándolo con diagramas de diseño utilizando alguna notación de uso común (p.ej. diagramas de estructura, diagramas de comportamiento, ...).

Debe incluir además, las principales decisiones de diseño.

IMPORTANTE

No repitas en esta sección lo que pueda especificarse en comentarios del código, como por ejemplo cuáles son los parámetros de una función, o el algoritmo de la misma.

Longitud máxima: 3 páginas.

- **Optimización.** Debe contener una discusión de las optimizaciones aplicadas y su impacto.
- En particular, deberán indicarse optimizaciones realizadas sobre el código fuente original, así como optimizaciones activadas con flags de compilación adicionales que se estime oportuno.

Longitud máxima: 1 página.

- **Pruebas realizadas:** Descripción del plan de pruebas realizadas para asegurar la correcta ejecución. Debe incluir pruebas unitarias, así como pruebas funcionales de sistema.
- En la memoria debe darse solamente una visión general del enfoque de las pruebas. La lista de pruebas concretas y como ejecutarlas deben encontrarse en scripts de ejecución (**utest.sh/utest.py** y **fctest.sh/fctest.py**).

Longitud máxima: 2 páginas.

- **Evaluación de rendimiento y energía:** Deberá incluir las evaluaciones de rendimiento y energía llevadas a cabo.

Longitud máxima: 5 páginas.

- **Organización del trabajo:** Deberá describir la organización del trabajo entre los miembros del equipo haciendo explícita las tareas llevadas a cabo por cada persona.
- Debe contener una división del proyecto en tareas.
 - Las tareas deben ser suficientemente pequeñas como para que se pueda asignar una única persona a una tarea.
 - Todos los integrantes del equipo deberán realizar contribuciones relevantes en el proceso de desarrollo de software.
 - No se podrá asignar más de una persona a una tarea. En tal caso, la tarea debe subdividirse en subtareas.
 - Se debe indicar el tiempo dedicado por cada persona a cada tarea.

Longitud máxima: 2 páginas.

Se podrá entregar una tabla en formato ODS con al menos las siguientes columnas: **nombre de tarea**, **nombre de persona**, **número de horas dedicadas**.

- **Conclusiones.** Se valorará especialmente las derivadas de los resultados de la evaluación del rendimiento, así como las que relacionen el trabajo realizado con el contenido de la asignatura.

Longitud máxima: 2 páginas.

6. Procedimiento de entrega

La entrega de la memoria del proyecto se entregará a través de un entregador habilitado al efecto en Aula Global. En este entregador se colocará un único archivo que deberá llamarse **report.pdf**.

Todo el proyecto desarrollado en el proyecto de GitHub asignado se entregará además en un archivo ZIP a través de Aula Global en un entregador habilitado al efecto. En este entregador se colocará un único archivo comprimido en formato ZIP que deberá llamarse **render.zip**.

7. Calificación

Las calificaciones finales para este proyecto se obtienen de la siguiente forma:

- Rendimiento: 20 %.
- Energía usada: 20 %.
- Pruebas unitarias: 7 %.
- Pruebas funcionales: 3 %.
- Calidad del diseño: 5 %.
- Calidad del código: 5 %.
- Evaluación de rendimiento y energía en la memoria: 15 %.
- Contribuciones de cada miembro: 20 %.
- Conclusiones: 5 %.

ADVERTENCIAS IMPORTANTES

- Si el código entregado no compila, la nota final de la práctica será de **0**.
- Si el código contiene modificaciones no autorizadas de los archivos de CMake suministrados, la nota final de la práctica será **0**.
- Si se ignora injustificadamente alguna norma de calidad de código, la nota final de la práctica será de **0**.
- Si el tiempo de ejecución se considera desmesuradamente alto, se calificará la práctica con un **0**.
- En caso de copia todos los grupos implicados obtendrán una nota de **0**. Además, se notificará a la dirección de la EPS para la correspondiente apertura de expediente disciplinario.

A continuación se indican los criterios específicos de evaluación para cada uno de las dimensiones de evaluación. Por favor, ten en cuenta que estos criterios tiene **carácter meramente indicativo** y están sujetos a la interpretación de tus profesores.

Rendimiento y energía Se definirán diferentes escenarios y configuraciones de evaluación con diferente grado de dificultad. Para cada evaluación se establecerán valores de referencia para obtener 5 puntos sobre 10 en la evaluación. También se establecerá un umbral para obtener 0 puntos sobre 10 en la evaluación.

La calificación final para rendimiento y energía se obtendrá como la media aritmética de las calificaciones de todas las pruebas.

Importante: Estos valores de referencia se publicarán entre el 13 y el 17 de octubre.

Pruebas unitarias El requisito es que haya un número de pruebas unitarias suficiente para todo el software desarrollado utilizando el framework GoogleTest. No se han definido criterios de cobertura específicos, así que no se va a analizar la cobertura. Se considerará que hay pruebas suficientes si hay pruebas para cada función o función miembro de cada clase.

Se definen los siguientes niveles:

■ **Excelente**

- Hay tres ejecutables de pruebas unitarias generados: **utest-common**, **utest-soa** y **utest-aos**.
- Hay un script (**utest.sh/utest.py**) que lanza todas las pruebas unitarias.
- Se utiliza como framework GoogleTest.
- Existen casos de prueba para cada función y para cada función miembro que contemplan tanto los casos básicos como los distintos casos de error.
- Las pruebas unitarias no dependen de la existencia de archivos concretos en ningún directorio.

■ **Aceptable**

- Se utiliza como framework GoogleTest.
- Existe un conjunto de pruebas que se corresponde con cada archivo .cpp del código fuente, pero no hay pruebas para todas las funciones o funciones miembro.

■ **Insuficiente**

- Se utiliza como framework GoogleTest
- Hay algunas pruebas pero son insuficientes.
- Se produce alguna violación de segmento u otro error catastrófico al ejecutar las pruebas.

■ **Deficiente**

- No se aportan pruebas unitarias que utilicen Google Test.
- No hay explicaciones de las pruebas unitarias en la memoria.

Pruebas funcionales El requisito aquí es que haya evidencias de que se han realizado pruebas de funcionamiento de la aplicación completa. Se debe tratar de automatizar las pruebas.

■ **Excelente**

- Existen pruebas funcionales completas que contemplan tanto los casos de éxito como los casos de error.
- Las pruebas funcionales se han automatizado con algún script y son reproducibles.

■ **Aceptable**

- Existen pruebas funcionales que son incompletas y solamente contemplan los casos que no son de error.
- Las pruebas funcionales se han automatizado con algún script y son reproducibles.

Insuficiente

- Se describen las pruebas funcionales en la memoria pero no se aporta ninguna manera de ejecutarlas de forma automatizada.
- Se intenta realizar pruebas funcionales desde un programa escrito en C++ en vez de comprobar el funcionamiento del ejecutable desde un script.

Deficiente

- No hay ninguna evidencia de la existencia de pruebas funcionales.
- No se mencionan las pruebas funcionales en la memoria.

Calidad del diseño El diseño comprende el diseño lógico (estructuras de datos, clases, funciones,...) y el diseño físico (estructura del código fuente).

■ **Excelente**

- Se aportan diagramas que dan una visión clara de la estructura del software.
- Se justifica de forma clara el diseño de las clases y funciones.
- Está clara la diferencia entre las estructuras usadas para AOS y SOA y estas son adecuadas.
- Las estructuras de datos auxiliares son las más adecuadas.
- Se justifican las decisiones principales para cada funcionalidad y las decisiones de diseño orientadas al rendimiento, incluyendo optimizaciones.
- Se mencionan flags específicos de optimización utilizadas.

■ **Aceptable**

- Se justifica el diseño de las clases y funciones, aunque hay decisiones poco claras o no justificadas.
- Hay diferencia entre las estructuras usadas para SOA y AOS y estas son adecuadas.
- Las estructuras de datos auxiliares son poco adecuadas.

■ **Insuficiente**

- El diseño no está suficientemente justificado y es difícil de entender.
- Las estructuras usadas para AOS y SOA no son adecuadas.

■ **Deficiente**

- No se describe el diseño de la aplicación.

Calidad del código En este apartado se valora que el código sea de buena calidad y no haya malas prácticas. La mayoría de estas comprobaciones ya se realizan con **clang-tidy**.

■ **Excelente**

- El código es claro, conciso y está bien formateado. Los nombres de las variables, funciones y tipos de datos son coherentes.
- Existen comentarios documentando cada función y tipo de datos.

■ **Aceptable**

- El código es claro, conciso y está bien formateado. Los nombres de las variables, funciones y tipos de datos son coherentes.
- No existen comentarios documentando cada función y tipo de datos.

■ **Insuficiente**

- El código carece de claridad y legibilidad.
- No existen comentarios documentando cada función y tipo de datos.
- Existe algún incumplimiento de **clang-tidy**, pero son escasos.
- Se han anulado reglas de **clang-tidy** mediante **NOLINT** de forma injustificada, pero son pocos.

■ **Deficiente**

- Existen incumplimientos de **clang-tidy** o bien se han anulado reglas de **clang-tidy** de forma injustificada (es decir **NOLINT** no autorizados) .

Evaluación de rendimiento y energía en la memoria En este apartado se valorará la discusión que haya en la memoria sobre evaluación del rendimiento y la energía.

■ **Excelente**

- Se ha realizado una evaluación del rendimiento y la energía para las distintas opciones del programa.
- Se ha realizado un análisis de la escalabilidad y se aportan gráficas.
- Se incluyen conclusiones a partir de los datos experimentales.

■ **Aceptable**

- Se ha realizado una evaluación del rendimiento y la energía para las distintas opciones del programa.
- Se ha realizado un análisis de la escalabilidad para al menos un parámetro significativo.
- No se aportan gráficas, pero si se aportan tablas de datos.
- Se incluyen conclusiones a partir de los datos experimentales.

■ **Insuficiente**

- Se ha realizado una evaluación del rendimiento o la energía solamente para algunas opciones del programa o solamente se aportan capturas de pantalla sin ninguna gráfica.
- No se ha incluido análisis de escalabilidad.

- No hay conclusiones a partir de los datos experimentales.

■ **Deficiente**

- No se ha realizado la evaluación del rendimiento y la energía.

Contribuciones de cada miembro Este es el único apartado que se valora de forma independiente para cada uno de los miembros del equipo de proyecto.

Observación inicial: Todos los miembros de un equipo recibirán una calificación de **DEFICIENTE** si se da alguna de las siguientes circunstancias:

- No se presenta división del trabajo en tareas individuales.
- No se presenta una lista de tareas realizadas.

■ **Excelente**

- Las contribuciones están bien descritas y son de granularidad adecuada.
- Para cada contribución se cuantifica la dedicación de la persona en horas. Las contribuciones y su cuantificación son creíbles.
- Las contribuciones de la persona son adecuadas en cantidad y calidad.

■ **Aceptable**

- Las contribuciones están bien descritas y son de granularidad adecuada.
- Para cada contribución se cuantifica la dedicación de la persona en horas. Las contribuciones y su cuantificación son creíbles.
- Las contribuciones de la persona son algo inferiores a lo esperable en cantidad y calidad.

■ **Insuficiente**

- Solamente se cuantifica en horas la dedicación total de la persona al proyecto, pero no el esfuerzo dedicado a cada tarea en concreto.
- No se puede determinar la cantidad total de horas dedicadas por la persona al proyecto.

■ **Deficiente**

- No se cuantifica en horas el esfuerzo dedicado por la persona al proyecto.

Conclusiones En este apartado se valoran las conclusiones del trabajo.

■ **Excelente**

- Las conclusiones incluyen lo inferido a partir de los datos experimentales del proyecto y las afirmaciones son coherentes y adecuadas.
- Se presentan conclusiones en términos de rendimiento y energía.
- Se incluyen conclusiones sobre lo aprendido.
- Se conecta el trabajo con los temas vistos en la asignatura.
- Las conclusiones presentadas no contienen incorrecciones y son relevantes.

■ **Aceptable**

- Las conclusiones son una mera repetición de los resultados experimentales sin ir más allá.
- Se incluyen conclusiones sobre lo aprendido o bien se conecta el trabajo con los temas vistos en la asignatura.
- Las conclusiones presentadas no contienen incorrecciones y son relevantes.

■ **Insuficiente**

- Las conclusiones son una mera repetición de los resultados experimentales sin ir más allá.
- No se incluyen conclusiones sobre lo aprendido ni se conecta el trabajo con los temas vistos en la asignatura.
- Las conclusiones presentadas contienen leves incorrecciones, pero en general son relevantes.

■ **Deficiente**

- No hay conclusiones o estas son irrelevantes.