

COMPUTACIÓN CIENTÍFICA MINIMALISTA

DIEGO ALBERTO VÉLEZ HENAO

UNIVERSIDAD PONTIFICIA BOLIVARIANA  
ESCUELA DE INGENIERÍAS  
MAESTRÍA EN INGENIERÍA  
MEDELLÍN  
2020

COMPUTACIÓN CIENTÍFICA MINIMALISTA

DIEGO ALBERTO VÉLEZ HENAO

TRABAJO DE GRADO PARA OPTAR POR EL TÍTULO DE  
MAGÍSTER EN INGENIERÍA

DIRECTOR:  
PhD. JORGE MARIO LONDOÑO PELÁEZ

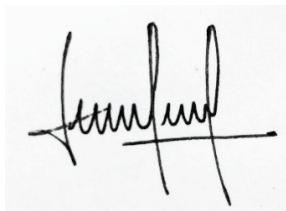
UNIVERSIDAD PONTIFICIA BOLIVARIANA  
ESCUELA DE INGENIERÍAS  
MAESTRÍA EN INGENIERÍA  
MEDELLÍN  
2020

## DECLARACIÓN DE ORIGINALIDAD

Calgary, Abril de 2020

DIEGO ALBERTO VÉLEZ HENAO

“Declaro que este trabajo de grado no ha sido preparado para optar a un título, ya sea en igual forma o con variaciones, en esta o cualquier otra universidad”. Art. 82 Régimen Discente de Formación Avanzada.

A handwritten signature in black ink, appearing to read "Diego Alberto Vélez Henao". The signature is fluid and cursive, with a prominent initial 'D' and 'A'. It is written over a horizontal line.

---

Firma

## **Reconocimientos:**

Deseo agradecer en primer lugar a mi Director de Tesis, Dr. Jorge Mario Londoño Peláez de la Escuela de Ingeniería de la Universidad Pontificia Bolivariana, por sus valiosas observaciones, recomendaciones, y por su cuidadoso seguimiento. Su profundidad en estos temas, apego a la calidad, y paciencia, son tan conocidos que su mención resulta redundante. Su influencia en mi formación, tanto durante el pregrado como ahora en esta Maestría, ha sido magnífica, por lo cual estoy obligado a reafirmar mi gratitud.

A mi mentor de carrera durante los últimos once años, Ing. Kenn Jorgensen, por la orientación técnica que me ha provisto a lo largo de este tiempo y por su ejemplo de integridad académica y profesional.

Quiero exaltar la labor de mis maestros del pregrado en Ingeniería Electrónica, pero muy especialmente la de mis formidables [ver nota] profesores Jairo Augusto Lopera Pérez, Rubén Darío Arboleda Vélez, Carlos Alberto Builes Restrepo, Iván Darío Mora Orozco, Sergio Cock Ramírez y Giovani Orozco Arbeláez.

A la memoria de nuestro querido maestro, Ing. Quím. Luis Fernando Montoya Valencia, quien siempre estuvo dispuesto a prestarnos atención y brindarnos su consejo en nuestros tiempos de principiantes.

A mis padres Orlando de Jesús y María Elena, por su amor y cuidados, y por la magnífica DTK XT-8088/10MHz/640KB/Seagate 20MB con MS-Basic y Lotus 123, con la que me inicié en estas exploraciones.

Diego Alberto Vélez Henao

*[Nota] Formidable:*

1. *Muy temible y que infunde asombro y miedo.*
2. *adj. Excesivamente grande en su línea.*
3. *adj. magnífico.*

## **Acknowledgments:**

I wish to first thank my Thesis Director, Dr. Jorge Mario Londoño Peláez, at the School of Engineering of the Pontifical Bolivarian University, for his invaluable observations, recommendations and ongoing follow-up. His depth of knowledge on these matters, attachment to quality, and patience are already widely known, turning redundant any further mention. His influence on my formation as an Engineer, both during my undergraduate's as well as during this Master's have turned out to be magnificent, forcing me to reaffirm my gratitude.

Also, I wish to thank my career mentor during the last eleven years, Eng. Kenn Jorgensen, for the technical guidance he has provided me over these years, and for setting the example in terms of both academic and professional integrity.

I would like also to recognize the hard work of my professors of the Electronics Engineering program at UPB, but most specially that of my formidable [see note below] professors Jairo Augusto Lopera Pérez, Rubén Darío Arboleda Vélez, Carlos Alberto Builes Restrepo, Iván Darío Mora Orozco, Sergio Cock Ramírez and Giovani Orozco Arbeláez.

To the memory of our beloved professor, Chemical Eng. Luis Fernando Montoya Valencia, who was always keen on paying attention to us and on providing us with his counsel in our times as freshmen.

To my parents Orlando de Jesús and María Elena, for their love and care, and for that magnificent DTK XT-8088/10MHz/640KB/Seagate 20MB with MS-Basic and Lotus 123, with which I got into the trade.

Diego Velez

*[Note] Formidable*

1. *causing fear, dread, or apprehension*
- 2: *having qualities that discourage approach or attack*
- 3: *tending to inspire awe or wonder: impressive*

## 1. Introducción

Los grandes avances en materia de electrónica y computación han brindado a la humanidad productos con gran impacto y aplicabilidad cotidiana. Los teléfonos inteligentes son un buen ejemplo de la magia de esta era: podemos tener, a un costo razonable, un pequeño dispositivo que además de ser un teléfono móvil incluye un procesador de gran desempeño; una cámara digital de alta resolución; posee capacidad local de almacenamiento de datos (con la posibilidad de ampliación ilimitada gracias a las tecnologías de “nube”), etc. En la actualidad, los procesadores en los teléfonos como el iPhone X se manufacteran con procesos en la escala de los 10 nanómetros, y están casi al alcance del consumidor procesadores con escalas de integración de 7 nanómetros, como en el caso de los procesadores Ryzen de AMD.

Sin embargo, en la medida en que el desarrollo nos lleva a sistemas computacionales cada vez más avanzados, parece darse una pérdida de control sobre los productos mismos, tanto para el usuario como para los desarrolladores de aplicaciones. Esta pérdida no se restringe a los sistemas de cómputo: por ejemplo, los nuevos vehículos son diseñados para asegurar su servicio únicamente por parte de centros autorizados por el fabricante, quedando muchas reparaciones rutinarias por fuera del alcance del dueño, pero también por fuera del de mecánicos generales, quienes pueden confirmar esta referencia.

Se presenta también en la actualidad la tendencia por parte de los fabricantes de aplicaciones de software comercial, de migrar del modelo de adquisición perpetua del permiso para usar el software, hacia uno de alquiler, en el cual se paga una cuota mensual o anual por usarlo, como en el caso, muy común, de Microsoft Office 365. Tanto en el caso de las licencias perpetuas como en el de las licencias por suscripción al usuario no se le otorgan derechos de inspección o modificación del producto por el que paga, sino solamente el derecho a usarlo.

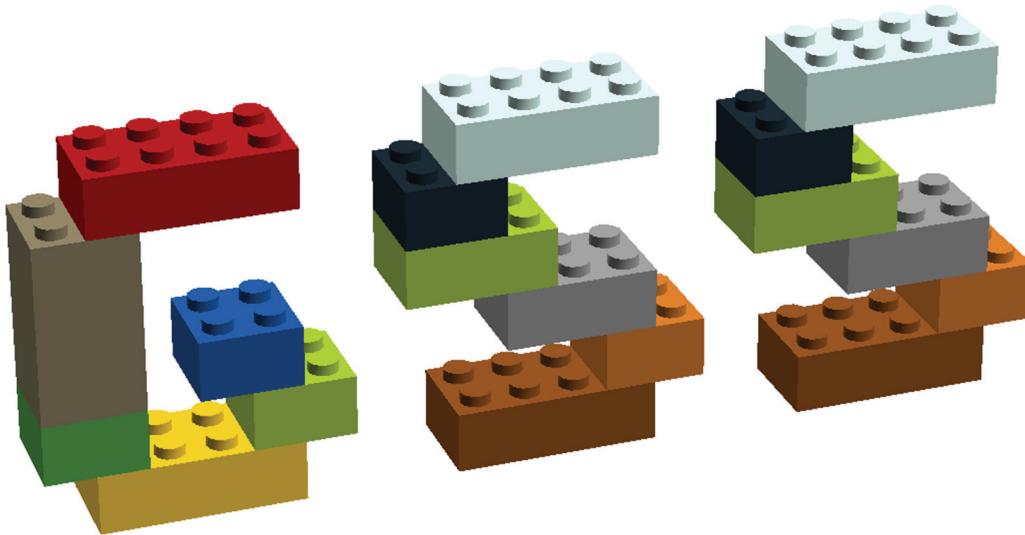
Otra estrategia comercial consiste en la entrega de aplicaciones con capacidad básica limitada, en la cual deben efectuarse pagos adicionales para hacer uso de otros módulos, siendo un caso cercano a la academia el del producto Matlab de la compañía Mathworks, con sus decenas de opciones.

Los movimientos del Software Libre, el del Software Abierto, y ahora, el de Hardware Abierto, tienen, entre otras metas, brindar alternativas a todo tipo de sistemas comerciales por medio de la cooperación e intercambio de ideas, sin costo, para el beneficio de todos. Estas metas son cercanas, como se verá, a las de este emprendimiento.

Este proyecto surge de un fuerte interés personal en los temas de computación numérica, gráficas por computadora, programación, y simulación, por parte del autor. Paradójicamente, mi ocupación ha consistido por varios años en el diseño de sistemas electrónicos cerrados y con software propietario, para sistemas de tipo embebido en aplicaciones de telecomunicaciones y de telemedida.

Sin embargo, cuando los cálculos demandan una mayor complejidad, cuando son iterativos, o cuando requieren un despliegue gráfico sofisticado, herramientas como Microsoft Excel (y su Visual Basic para Aplicaciones) no resultan las más adecuadas, y menos si se requiere portar parte del código a una una plataforma distinta como Unix/Linux o incluso a un sistema embebido. La insatisfacción debida al alto costo de sistemas comerciales (Matlab, Mathcad, y otros), el ámbito (restringido a Unix/Linux) de otros de tipo libre (como GNU Octave) y la dificultad de portar código a microcontroladores me llevaron a plantear y a tomar como meta el desarrollo de un tipo de herramienta diferente. Se enumeran algunas características del sistema que se desearía desarrollar:

- Que no sea una aplicación terminada, sino un juego de herramientas para hacer lo que se deseé, al estilo de un juego de Lego®. Esta filosofía se refleja el nombre del proyecto: “Graphical Simulation Sandbox” o “Caja de Arena para Simulación”.



**Figura 1:** Logotipo del proyecto. [Recurso propio].

- Que otorgue al usuario control sobre aspectos gráficos a un bajo nivel y que mantenga un buen desempeño.
- Que permita la elaboración de gráficas en software independientes de la plataforma y modificables o extensibles por parte el usuario.
- Que posea un enfoque de “caja de cristal”, es decir, que facilite la inspección y cambios por parte del usuario.
- Que incorpore algunas librerías básicas para operaciones algebraicas, cálculo numérico, variable compleja, para la evaluación funcional de expresiones en notación matemática, y para interconexión a dispositivos electrónicos externos.

## 2. Modelo de Software

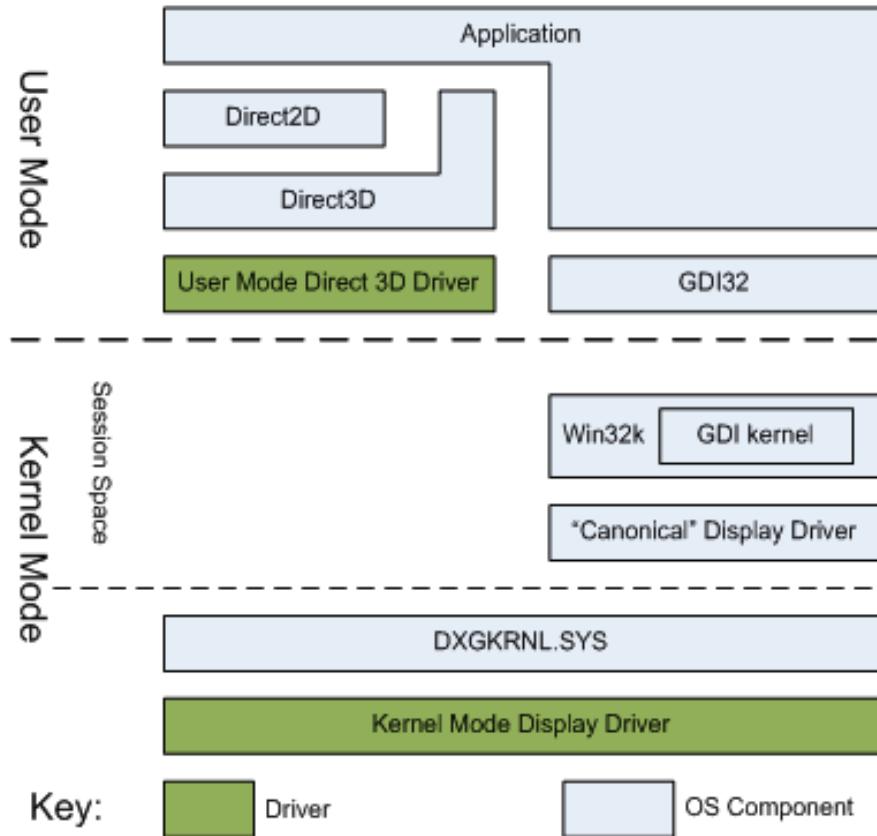
Dado que el proyecto hace énfasis en las gráficas por computadora como recurso para realizar simulaciones, y a que cada sistema operativo establece la operación de uno o más subsistemas gráficos, se considera necesario revisar algunos aspectos técnicos de dichos sistemas antes de presentar el modelo de software seguido. Posteriormente, se discuten también otros modelos que se evaluaron y que fueron finalmente descartados. El orden que se seguirá es el siguiente:

- a) Revisión de los modelos y características de los módulos gráficos bajo los sistemas Unix/Linux y Windows, con una referencia breve al sistema Mac OS.
- b) Presentación del modelo finalmente elaborado.
- c) Discusión del desarrollo, resultados, y debilidades encontradas en otros modelos explorados, los cuales fueron finalmente excluidos.

### 2.1. Sistemas gráficos bajo Unix/Linux y Windows

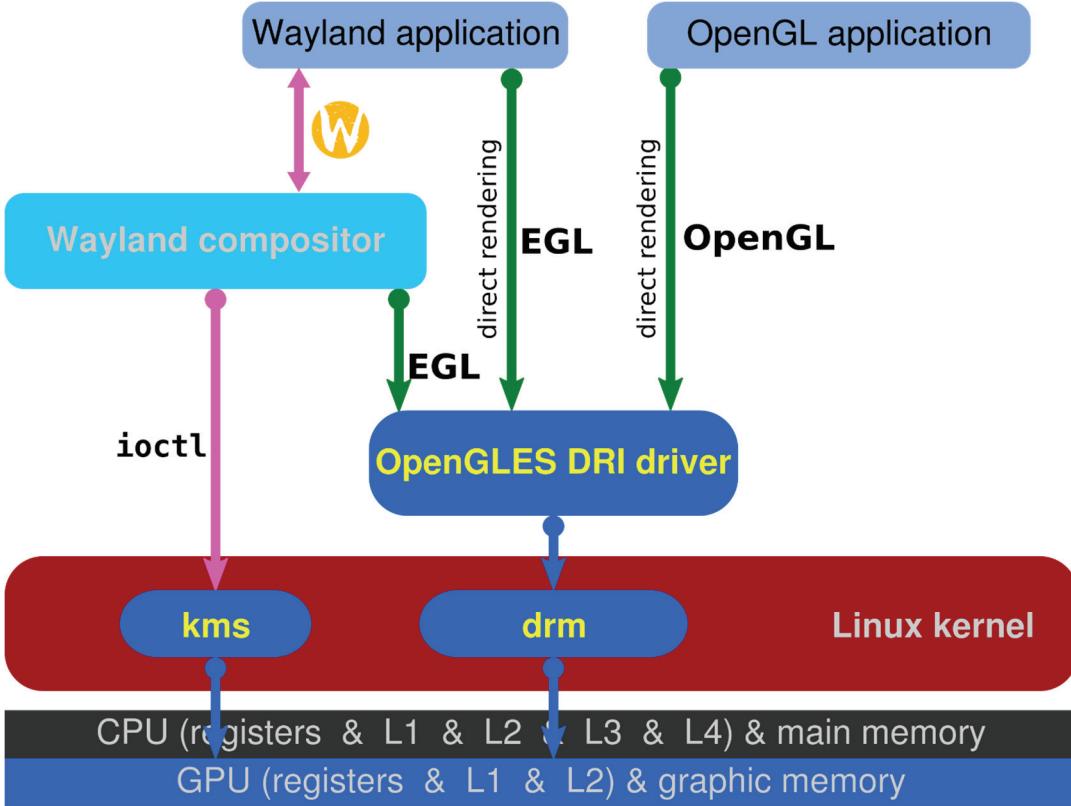
El manejo gráfico es un aspecto central de este proyecto. Las gráficas por computadora constituyen un campo vastísimo y en permanente evolución. El dilema consiste en definir el nivel al cual se desea llegar, entendiendo como bajo nivel aquel cercano al hardware, y alto nivel aquel cercano al usuario. Es oportuno recordar (con algo de nostalgia para quienes alcanzamos a ver parte de los desarrollos de los años 80's y 90's) que los primeros sistemas computacionales entregaban el control al usuario, quien era también el desarrollador. Por ejemplo, la mayor parte de los sistemas y sus lenguajes de desarrollo permitían manipular directamente la memoria gráfica con instrucciones como `PSET(X, Y) [,Color]` de QuickBasic; `putpixel(x, y, color)` de la memorable Interfaz Gráfica de Borland; o accediendo a la memoria del sistema mediante instrucciones como `POKE(address, value)` de QuickBasic y otros.

En los sistemas modernos, las gráficas se realizan mediante una serie de capas o niveles de abstracción, los cuales buscan independizar los dispositivos físicos, del sistema operativo, y finalmente, de las aplicaciones. Los lenguajes de alto nivel normalmente proveen métodos para crear ventanas y realizar gráficas en ellas. Por ejemplo, las aplicaciones de Windows, pueden utilizar llamadas a métodos de la GDI (Graphics Device Interface) o de Direct2D para el despliegue de elementos de una ventana o dibujo en un “lienzo”, con diversos niveles de aceleración. La clasificación, características, y detalles de las interfaces gráficas son un tema extenso. La siguiente gráfica el modelo seguido en los sistemas Windows. El documento [1] ofrece detalles de la operación y diferencias entre GDI y Direct2D.



**Figura 2:** Modelo de Interfaces Gráficas en Sistemas Microsoft Windows.  
 Microsoft, Windows Developer Center. (2018). *Comparing Direct2D and GDI Hardware Acceleration*. [Figura]. Recuperado de <https://docs.microsoft.com/en-us/windows/win32/direct2d/comparing-direct2d-and-gdi>

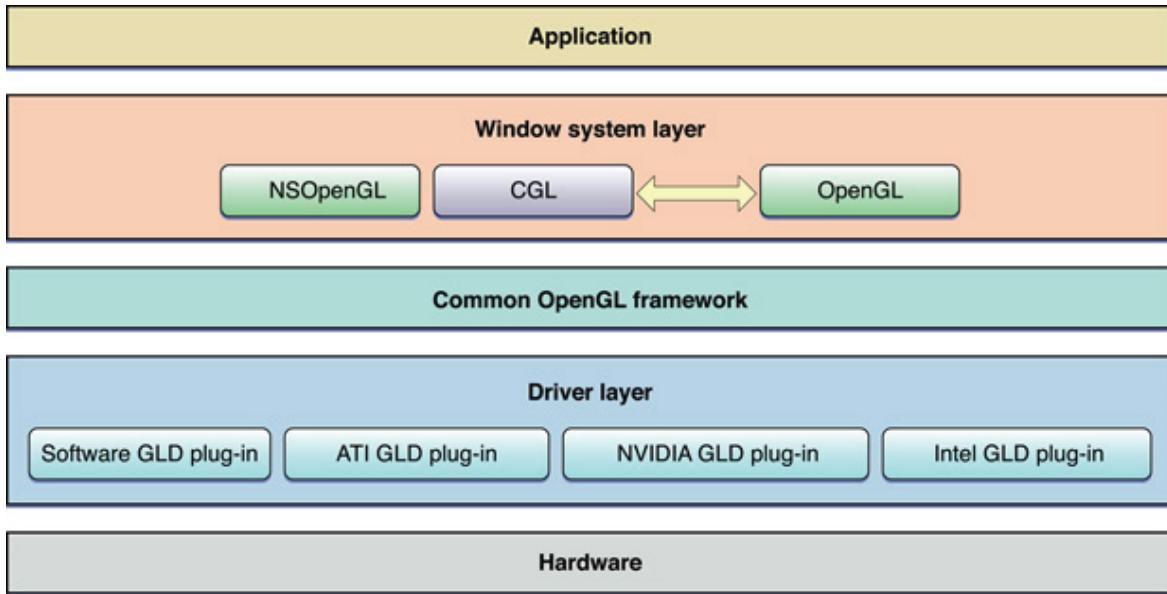
Para el caso de Unix/Linux, en distribuciones con kernel versión 3.2 o superior existen en la actualidad dos tipos principales de acceso al hardware: Wayland [2] (EGL, Embedded-System Graphics Library) y OpenGL. Se aclara que este es un modelo actual simplificado, pero que cada sistema puede soportar otros métodos. Por ejemplo, el driver “framebuffer” `/fbdev`, que opera al nivel de kernel es aún soportado en muchos sistemas, incluyendo las últimas distribuciones de Linux Mint (19.2) y los sistemas Raspberry Pi. La tendencia, según se puede ver en los foros del tema, es que los desarrolladores abandonen el framebuffer y en su lugar utilicen DRM (Direct Rendering Manager, subsistema del kernel a cargo de la interfaz a la GPU o Graphics Processing Unit).



**Figura 3:** Modelo de Controladores Gráficos en sistemas Unix y Linux.  
 Wikipedia. (2019). *Wayland (display server protocol)*. [Figura]. Recuperado de [https://en.wikipedia.org/wiki/Wayland\\_\(display\\_server\\_protocol\)](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol))

Bajo el sistema Mac OS, el modelo gráfico ha tenido cambios significativos, principalmente por la decisión de Apple de abandonar el soporte de OpenGL (a partir de la versión 10.14 de Mac OS, “Mojave”, liberada en septiembre de 2018) y forzar a los desarrolladores a utilizar el nuevo sistema llamado “Metal” [3]. Esta decisión tiene un alto impacto en algunos círculos, especialmente en el de los desarrolladores de juegos [4] [5].

Metal, básicamente, es un sistema en el cual las aplicaciones establecen un enlace con la GPU y ejecutan comandos en ella. Esta es una diferencia con OpenGL, en el cual esta comunicación es transparente al usuario.



**Figura 4:** Modelo gráfico en Mac OS para versiones anteriores a 10.14.

Horwitz, J. (2018). *Apple defends end of OpenGL as Mac game developers threaten to leave.* [Figura]. Recuperado de <https://venturebeat.com/2018/06/06/apple-defends-end-of-opengl-as-mac-game-developers-threaten-to-leave/>

Este proyecto de tesis, sin embargo, es enfocado a la computación en sistemas abiertos y, preferiblemente, de bajo costo. Por consiguiente, no se harán esfuerzos tendientes a alcanzar o mantener compatibilidad con el sistema Mac OS.

## 2.2. Concepto y diagrama de capas del sistema de Simulación finalmente elegido.

Teniendo en cuenta que el objetivo es lograr una herramienta que permita la elaboración simple de simulaciones de tipo general, se propone un esquema simplificado, basado en el lenguaje C, compuesto de librerías de soporte para la elaboración de gráficas en una forma independiente del dispositivo. Para el proyecto se elaboraron también librerías de tipo matemático y numérico, las cuales son en su mayor parte implementaciones propias de algoritmos convencionales y, como tales, no presentan restricciones de propiedad intelectual. Estos módulos de soporte pueden ser llamados directamente por el programa principal, y como tales, solo dependen de las librerías convencionales estándar del lenguaje C, tipo `<stdlib.h>` y `<math.h>`. Por ello, se representan simplemente en el diagrama del modelo como conectados directamente a la aplicación (simulación), sin estratificación.

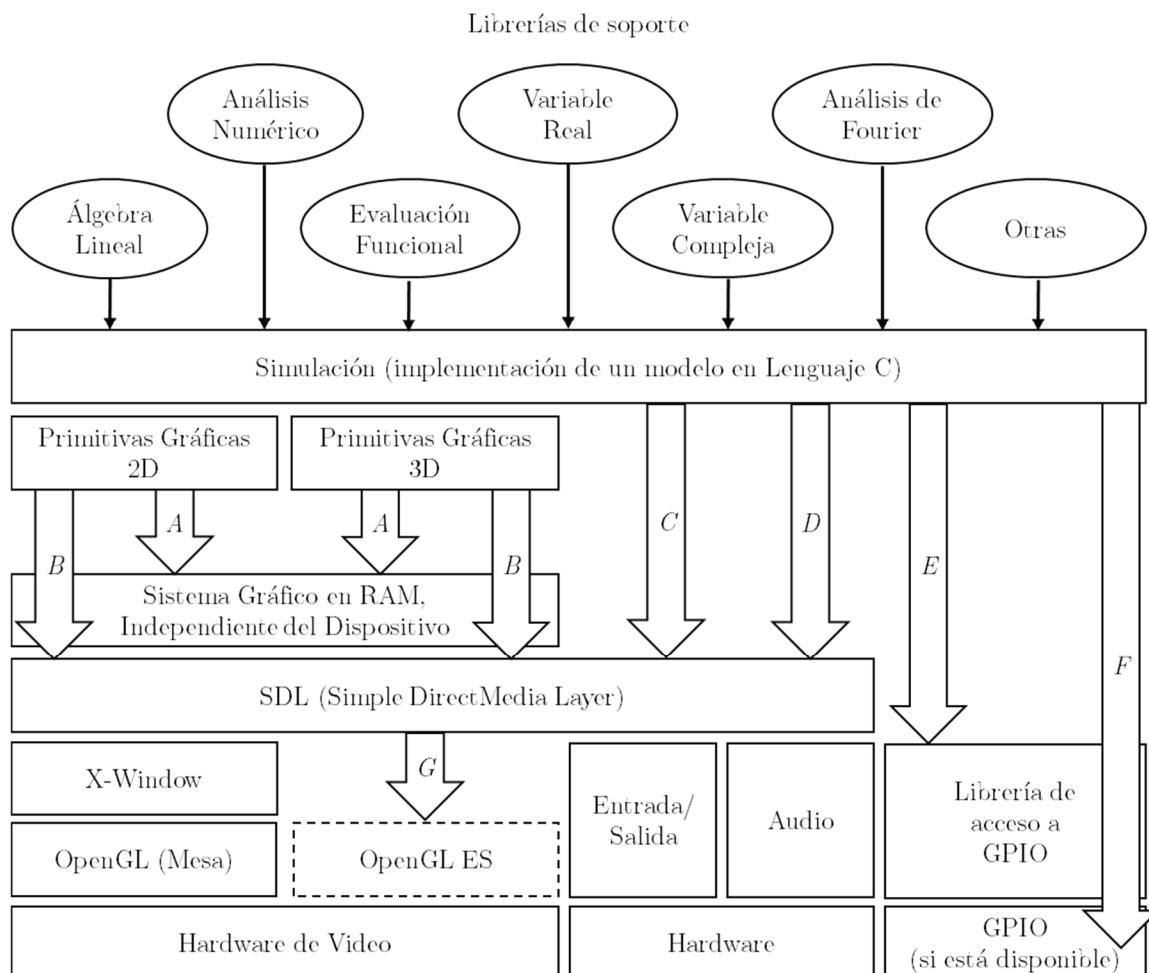
El sistema busca la generación de ejecutables y no es interpretado, por lo cual (en contraste con los sistemas numéricos tipo Matlab, GNU Octave, o lenguajes interpretados como Python), no

se desea incluir una interfaz de comandos. La elaboración de un programa numérico sigue la misma línea tan familiar de los programas en C:

- Se incluyen los nombres de las librerías a llamar.
- Se declaran las variables globales o estructuras de datos que se requieran.
- Se definen las funciones que se necesiten.
- Se crea un cuerpo principal del programa o “main()”, en el cual se establece el flujo del mismo.

Si la simulación o programa a elaborar requiere gráficas, se inicializa el sistema gráfico con una sola llamada a la función `lb_gr_init()`. Al cerrar el programa, se deberá llamar la función complementaria `lb_gr_close()`.

El modelo de software, y la interacción con otros elementos del sistema se muestra con el siguiente diagrama:



**Figura 5:** Modelo general del Software. [Recurso propio].

Descripción del modelo:

- a) Las operaciones gráficas se pueden llevar a cabo en una o varias imágenes, creadas en RAM y definidas por medio de una estructura de datos adecuada llamada `PICTURE_T`. Este modo en principio es independiente de la plataforma y del dispositivo.
- b) Se ha provisto un modo de acceso directo a un “framebuffer”, compatible con SDL (Simple DirectMedia Layer), el cual evita la declaración de estructuras de datos para simulaciones simples o para usuarios principiantes.
- c) Los procesos de entrada y salida (como el teclado) se pueden realizar a través de SDL. Más precisamente, la entrada y salida se deben realizar a través de SDL si ésta se ha inicializado.
- d) Es posible incorporar entrada y salida de audio a las simulaciones por medio de SDL, evitando el desarrollo de complejas interfaces que en la mayor parte de los casos son específicas a cada sistema.
- e) Acceso al GPIO (General Purpose Input and Output) en sistemas embebidos como Raspberry Pi haciendo uso de la librería `<lb_gpio.c>`.
- f) Acceso directo al GPIO mediante registros.
- g) SDL puede acceder directamente a OpenGL ES (“Embedded Systems”) sin hacer uso de X, para programas de pantalla completa.

Este tipo de modelo ofrece las siguientes características:

- El procesamiento gráfico está hecho en software, en forma independiente de la plataforma y de los dispositivos. Se aprovecha la compatibilidad y omnipresencia de OpenGL pero se utiliza a modo de “driver”, sin hacer uso de sus funciones internas, por ser un sistema avanzado, y que no está orientado a usuarios básicos o intermedios.
- Se aprovecha el soporte de SDL de dispositivos de entrada y salida, y para el hardware de sonido, permitiendo la posible adecuación del sistema a plataformas como Windows, con pocos cambios.
- Permite el reuso de partes del código incluso en microcontroladores que soporten C. Es posible utilizar librerías numéricas avanzadas como la reconocida GSL (GNU Scientific Library). Enlace: <https://www.gnu.org/software/gsl/>
- El enfoque modular permite la readecuación del sistema para otro tipo de aplicaciones. Se dedica posteriormente una sección a las posibles aplicaciones del sistema.

### **2.3. Elección de SDL (Simple DirectMedia Layer) como opción para un sistema de simulación**

SDL es una librería orientada a simplificar la creación de juegos y aplicaciones multimedia en diversos sistemas operativos como Android, iOS, Linux, macOS y Windows. Es gratuita, de código abierto, y su licencia permite el enlace estático por parte de proyectos de código cerrado

o comerciales sin pago de regalías. SDL busca proveer una interfaz común bajo la mayor parte de los sistemas operativos para el control de aspectos como el video, el sonido, el acceso a archivos, funciones de temporización, “threads” y otros. SDL utiliza OpenGL y lo complementa, pues facilita otros aspectos como la creación de ventanas. Bajo Microsoft Windows, SDL puede utilizar OpenGL o DirectX.

Al ser SDL una librería que se apoya en las capacidades de OpenGL, este último considerado un estándar para gráficas computacionales e industriales de alta calidad, es razonable esperar que provea un buen desempeño y que sea soportado por una amplia gama de hardware gráfico. OpenGL, sin embargo, es un sistema relativamente complejo, y orientado a usuarios avanzados, y no ofrece soporte a dispositivos de entrada y salida, audio, y otros que pueden ser de interés para construir un sistema de simulación.

SDL, por sí misma, no incorpora primitivas gráficas avanzadas, salvo las más elementales como funciones para el dibujo de líneas y rectángulos, y funciones para la escritura de píxeles, como `SDL_RenderDrawPoint()` y `SDL_SetRenderDrawColor()`. Estas funciones pueden ser útiles para pruebas simples o para la generación de imágenes estáticas, pero no resultan adecuadas para su uso en la elaboración de una librería gráfica, pues además de requerir dos pasos (definir el color y luego escribirlo), son funciones demasiado lentas debido a múltiples niveles de anidamiento y a conversiones internas del formato de datos.

Una técnica de mejor desempeño y de uso extendido a lo largo de este proyecto consiste en establecer o identificar el formato de datos usado por SDL para las imágenes, crear en memoria arreglos que sigan este formato, operar en ellos, y transferirlos directamente para su presentación en pantalla. Es oportuno mencionar que las imágenes en SDL 2.0 se denominan formalmente “textures” (en la versión 1.0 eran llamadas “surfaces”), nombres que posiblemente no reflejan muy bien el arreglo o estructura representados por ellos, aunque posiblemente se eligieron para evitar confusión con términos ambiguos como “buffer”, “bitmap”, “framebuffer”, etc.

Este método abstrae completamente las gráficas y las independiza de SDL (y, de hecho, de cualquier librería), dejando abierta la opción de utilizar otros dispositivos de salida que se requieran o que ofrezcan características específicas tales como dimensiones físicas definidas o bajo costo. Por ejemplo, es posible adaptar fácilmente la salida del sistema a módulos de pantalla LCD/OLED con interfaces seriales o paralelas<sup>1</sup>; a impresoras tridimensionales para generar sólidos capa por capa; a matrices táctiles para lectura por personas invidentes; a sistemas de visualización vectoriales tipo osciloscopio (mediante la adición de conversores Digitales/Analógicos a la salida del sistema) y a otros tipos de hardware.

---

<sup>1</sup> Excepto aquellos con interfaces VGA (Video Graphics Array) o HDMI (High-Definition Multimedia Interface), los módulos gráficos LCD/OLED no suelen ser directamente compatibles con el hardware de video convencional debido a obvias diferencias tanto a nivel físico como lógico.

Otro beneficio de este método es que facilita la creación de animaciones, pues se pueden crear tantas imágenes como se deseen (su número está limitado únicamente por la cantidad de memoria disponible), y definir estructuras de control que vayan efectuando el procesamiento de los siguientes cuadros de la animación, rotándolos si se desea, y despachando a pantalla aquellos que estén terminados al momento exacto.

En síntesis: se hace uso de SDL para acceder a las grandes ventajas en términos de compatibilidad y desempeño de OpenGL con Unix/Linux, Mac OS y Microsoft Windows, pero manteniendo un grado de separación que permite adaptar el sistema a otros controladores o dispositivos de salida, lo cual se logra implementando las gráficas en forma independiente, mediante un solo nivel de abstracción. En última instancia, también se podría abandonar SDL y portar a un nuevo sistema de video la mayor parte del código.

#### **2.4. Ubicación y uso de las librerías**

Desde el punto de la organización de las librerías, el sistema es bastante plano, y todas las librerías se ubican por defecto en el directorio principal del proyecto, aunque podrían reorganizarse de otra manera.

Se aclara que, a diferencia del enfoque tradicional en el cual hacen llamados a librerías o “shared objects” (archivos con extensión .so en sistemas Unix/Linux) o archivos .DLL en sistemas Windows) como entidades independientes, hechas por expertos y lejanas al usuario, se busca una mayor interacción (al menos a modo de lectura o inspección) por parte del mismo. Es decir, el código fuente no solo debe ser asequible a petición del interesado, sino que se espera sea incluido para su lectura, estudio, modificación y mejoras.

#### **2.5. Otros modelos de software evaluados, desarrollos y resultados**

Recordando que el objetivo es el desarrollo minimalista de simulaciones con soporte gráfico, se exploraron y desarrollaron otros dos modelos (ahora abandonados, pero que pueden ser consultados bajo el directorio [/docs/OBSOLETE\\_CODE](#)), antes de llegar al modelo actual. Cronológicamente, el modelo final, basado en Simple DirectMedia Layer pareció adecuado desde el comienzo y fue el primero en el cual se alcanzó inicialmente un grado básico de funcionalidad. Posteriormente, se investigaron los siguientes métodos:

1. Sistema basado en X-Window
2. Sistema basado en el uso de la interfaz “framebuffer”

Debido a las limitaciones y desventajas generales encontradas con estos métodos (las cuales se discuten posteriormente), se regresó al modelo basado en SDL y se mejoró.

## 2.6. Modelo basado en X-Window (descartado)

Un parámetro que fue central durante la exploración inicial fue la independencia de librerías externas. Se buscó, por consiguiente, realizar todas las operaciones gráficas a través del sistema X-Window. La ventaja inherente a X-Window es su disponibilidad inmediata en la mayor parte de los sistemas Unix/Linux, sin la instalación de librerías adicionales, y este aspecto ameritó la asignación de una parte sustancial de tiempo para su implementación y pruebas.

Se hace un recuento breve de X-Window y su operación:

- Se conoce también como X11 o ‘X’.
- Es un sistema que data de mediados de los 80’s y se basa en un modelo de cliente-servidor.
- El servidor despliega las gráficas y los clientes (los cuales pueden estar en la misma o en otras máquinas), envían comandos que indican al servidor las operaciones de texto o gráficas a realizar.
- El servidor, por otra parte, puede comunicar eventos a los clientes, por ejemplo, la pulsación de una tecla, o notificar un cambio en la exposición de la ventana.

Para desarrollar pruebas con X11, un obstáculo inicial es la obtención de información, pues no es un sistema orientado a ser amigable al programador: son notorias las diferencias entre versiones y la ausencia de una guía oficial. Existen algunos libros de texto respecto al tema, como “Introduction to the X-Window System” por Oliver Jones [6], cuyo contenido en general suelen es anterior a las versiones actuales. El documento más útil que se encontró para iniciar el trabajo con el sistema X, es de autoría del profesor Douglas Thain del la Universidad de Notre Dame en Indiana [7], y describe los conceptos esenciales para inicializar la conexión al servidor y ejecutar algunos comandos.

Al avanzar el trabajo con X, se hicieron evidentes ciertos aspectos relacionados con la forma en que este fue concebido e implementado. Por ejemplo, es un sistema que en principio asume que las necesidades gráficas son elementales y no sofisticadas. Incorpora primitivas simples, como para el trazo de líneas, rectángulos y elipses, y posee cierto soporte de texto y fuentes, pero por ejemplo no ofrece primitivas avanzadas como aquellas con técnicas de anti-escalamiento<sup>2</sup>. Parece, al menos hasta donde es posible establecer, que no hay soporte para acceso individual a nivel de pixeles: toda operación desde el cliente debe hacerse a través de bloques rectangulares (el truco usual es definir un rectángulo de ancho y alto unitarios, y copiarlo). Cabe decir que el desempeño es apenas aceptable para imágenes estáticas, y resulta poco satisfactorio cuando

---

<sup>2</sup> El término “anti-aliasing” del inglés es ampliamente utilizado para referirse a las técnicas que permiten evitar o reducir los saltos o discontinuidades, o que ayudan a obtener una apariencia suave en el contexto del Procesamiento Digital de Señales, pero especialmente en el de las gráficas impresas o computacionales.

se presentan imágenes en secuencia para presentar una animación, como suele ser el caso al realizar simulaciones físicas.

X, a raíz también de ser un sistema cliente-servidor, asume también que cada cliente conectándose al servidor podría utilizar parámetros diferentes como por ejemplo el número de bits por cada canal de color, y el orden o formato de los mismos. Esto implica que las operaciones entre el cliente y el servidor normalmente requieren conversiones, las cuales internamente se efectuan mediante operaciones desplazamientos, reordenamiento de bits, truncamiento y consulta de tablas. Por consiguiente, el desempeño puede ser limitado. Un caso bastante común es que tanto el servidor (el cual genera los gráficos) y el cliente (el agente que indica mediante comandos las operaciones gráficas a realizar) se encuentren ambos en la misma máquina, y sin embargo deben seguir el protocolo de comunicación establecido, con la respectiva pérdida de eficiencia. Nota: existen técnicas en uso que pueden mitigar este problema si son instaladas y habilitadas, como son:

1. Unix Domain Sockets, que permite que la comunicación entre el cliente y el servidor (ubicados en la misma máquina) se efectúe a nivel de kernel.
2. MIT Shared Memory Extension or MIT-SHM, la cual permite que el cliente y el servidor comparten áreas de memoria.

Otra gran limitación encontrada en X, la cual dificulta su uso como sistema para simulaciones es que no presenta un método directo para leer los datos en pantalla. El método que se siguió consiste en leer un bloque rectangular, transferirlo al cliente, y reinterpretar los colores de cada uno de los pixeles. Este proceso no solamente es lento sino aproximado: es posible escribir un color y solamente obtener uno similar al leerlo nuevamente. Las técnicas gráficas más avanzadas suelen requerir acceso directo, no solo de escritura sino también de lectura para poder, por ejemplo, difuminar el contenido del pixel a escribir considerando el contenido del pixel de destino y quizás también el de aquellos adyacentes. Este problema fue mitigado mediante el uso de un “buffer fantasma” que mantenga en el cliente una copia local de los datos gráficos en la memoria del servidor en un formato compatible con el mismo, a fin de permitir su copia con una sola operación de X.

El consenso entre desarrolladores familiares con el tema respecto al manejo de eventos (como la entrada de teclado) bajo X11 es que es intrincado. Por ejemplo, se pudo corroborar que una pulsación de teclado puede generar uno, dos, tres, o más códigos, hecho que no sería un problema per-se si al menos existiera una mejor documentación o un standard. Se escribió una función que interpreta los códigos asociados a un evento de teclado y los convierte al código ASCII convencional con un resultado aceptable.

Para referencia, el código de pruebas con X11 (ahora abandonado y excluido del proyecto), se encuentra disponible en [`/docs/x1\_example/lb\_x11.c`](#) tras descargar el proyecto completo.

En resumen, el plan de uso de X11 como sistema gráfico ofrece las siguientes características:

Ventajas:

- No requerir la instalación de librerías adicionales, excepto en las distribuciones de Linux de “consola”. Para el programador, basta incluir las cabeceras `<Xlib.h>`, `<Xutil.h>`, y `<XKBlib.h>` y llamar las funciones respectivas.

Desventajas:

- Baja disponibilidad de documentación precisa, consistente y actual.
- Es un sistema poco amigable al programador, con cambios alrededor de su plan inicial, los cuales fueron necesarios para sortear las limitaciones que posee. Se considera un sistema “remendado” o “patchy”.
- Tiene un desempeño bajo para operaciones gráficas intensivas.
- Carece de métodos de acceso directo al buffer sobre el cual opera el servidor, lo cual obliga a operaciones más lentas como copia de bloques.
- No es posible leer en forma directa píxeles del buffer de pantalla. Esto puede hacerse, pero de forma indirecta, mediante copia de bloques y conversiones de colores.
- X-Window está restringido al ámbito de Unix y Linux y es en general incompatible con Windows (salvo proyectos limitados y de baja difusión como Xming). Mac OS no ofrece soporte nativo de X11, aunque existe cierto grado de soporte mediante la instalación de XQuartz.
- X-Window no cubre otros aspectos de hardware como el soporte de dispositivos de audio.

Por estas razones, pese a haber alcanzado un estado funcional, se estableció que este modelo no es el más adecuado para un sistema de simulación, y se abandona como soporte del sistema gráfico.

## 2.7. Modelo basado en el controlador “Framebuffer”

La idea de alcanzar un mejor desempeño llevó a explorar la posibilidad de obviar “del todo” el sistema X-Window. Esto es posible mediante el uso de una interfaz de más bajo nivel.

El “framebuffer”, también llamado “fbdev” es una abstracción del hardware gráfico, el cual es representado como un dispositivo llamado `/dev/fb0` y solamente aplica a los sistemas Unix/Linux. En caso de que existan varias tarjetas gráficas (o que estas posean múltiples salidas), pueden existir otros dispositivos, numerados `/dev/fb1`, `/dev/fb2`, etc. Cada uno de estos dispositivos opera independientemente del resto.

Una característica interesante para el desarrollador es que esta interfaz se comporta como un bloque de memoria, el cual puede ser leído o escrito según se necesite. Inclusive, puede ser accedido a través de la línea de comandos, por ejemplo: `cp /dev/fb0 nombre_archivo` descarga el contenido del framebuffer a un archivo.

Para utilizar fbdev, se deben incluir la librería `<linux/fb.h>`. La configuración del dispositivo se realiza a través de llamadas a funciones de control `ioctl()` las cuales permiten leer o definir los parámetros de video, como la resolución, el número de bits por canal, el uso o no de un canal de alfa (transparencia), y otros más avanzados como las frecuencias para definir el sincronismo horizontal y vertical; estos últimos provienen del mundo análogo, pero pueden no ser relevantes para las interfaces digitales.

“Framebuffer” permite en teoría el soporte de doble-buffering (de hecho, de n-buffering) mediante la creación y mapeo al dispositivo de un bloque de memoria dos, tres, o ‘n’ veces mayor al requerido, y el uso de un parámetro de “offset”, el cual, como se dijo, se establece a través de la función de control `ioctl()`. A fin de garantizar la fluidez de una animación, se debe también hacer uso del parámetro `FBIO_WAITFORSYNC`, el cual debe sincronizar el cambio en valor del offset con el “barrido” vertical de pantalla (o con la actualización de la misma en sistemas digitales). Desafortunadamente, estas características especiales no funcionan bajo todas las configuraciones, como se pudo comprobar.

Una característica de “framebuffer” es que no hay manejador de ventanas, pues no opera bajo X-Window, sino al revés: X (en caso de que esté instalado y sea inicializado manualmente con `startx` o automáticamente, como ser el caso más común) puede estar utilizando el framebuffer (o, su competidor, DRM, Direct Rendering Manager). Por consiguiente, el desarrollador toma el control total de la pantalla, perdiendo la compatibilidad con X-Window y los beneficios con los que estamos familiarizados como usuarios, como son: la ejecución de varias aplicaciones al mismo tiempo, la posibilidad de intercambiar información entre ellas, el salto de una aplicación a otra, etc. Intentar controlar el framebuffer desde X causa errores y comportamientos anómalos. Sin embargo, es válido abrir una consola virtual (Control-Alt-F1 en Ubuntu, Linux Mint, y otras distribuciones) y desde ellas ejecutar aplicaciones “framebuffer”, pudiendo retornar a X con Control-Alt-F7.

A propósito, una forma de establecer las consolas en uso es por medio del comando `w` (“who”), el cual muestra los usuarios activos y sus respectivas consolas.

Existe un importante detalle adicional que hay que tener en cuenta a la hora de desarrollar aplicaciones “framebuffer”: la consola, que permite al usuario la entrada y salida en forma de texto, utiliza también el framebuffer y, por consiguiente, debe deshabilitarse su salida en

pantalla. Esto se realiza con el comando `ioctl(fd_tty, KDSETMODE, KD_GRAPHICS)`. Similarmente, para retornar al modo de consola, se utiliza la misma función con el parámetro `KD_TEXT`.

Para la ejecución de programas en modo gráfico, el desarrollador deberá crear su propio mecanismo para representar la entrada de usuario en pantalla, definiendo o utilizando fuentes de tipo fijo o vectorial, con el esfuerzo y nivel de detalle que esto conlleva. Es necesario, igualmente, tomar cierto tipo de precauciones: una de ellas es que un programa gráfico que sea cerrado (o que termine debido a un error) no retornará automáticamente el control al usuario en modo de texto. Interrumpir el programa con `^C` deja el sistema en un modo en el que no refleja la entrada del teclado ni presenta un “prompt” o símbolo del sistema, lo cual en la práctica solamente se resuelve reiniciando el equipo (o cerrando la consola virtual respectiva). Esta condición se puede prevenir mediante el redireccionamiento de la señal de terminación de programa a una función de cierre ordenado que restaure la salida de consola al framebuffer (es decir, reestablezca el modo de texto) y cierre el programa. El redireccionamiento de la señal de terminación, a propósito, se puede realizar con la función `signal(SIGINT, function_ptr)`. Otra solución remedial, no implementada, podría ser la elaboración de un programa corto que restaure el modo de texto, utilidad que en todo caso debería ser llamada sin que la consola represente en pantalla los caracteres digitados, es decir “a ciegas”.

El método del framebuffer presentó un buen desempeño considerando que no son gráficas aceleradas y que estas son realizadas enteramente en software, pues permitió alcanzar tasas de 60 a 120 cuadros por segundo a resoluciones de 1920x1080 en la mayor parte de los sistemas probados, incluyendo sistemas de escritorio tipo i5, i7, y máquinas virtuales. El desempeño en sistemas Raspberry Pi fue inferior pero aún comparable al de sus contrapartes de escritorio, siendo del orden de 15 a 24 cuadros por segundo. Es de resaltar que la interfaz framebuffer funcionó sin cambios para este sistema, pese a que inicialmente se esperaban diferencias significativas por tratarse de otra plataforma.

Se hace un recuento de las principales ventajas y desventajas de este método:

Ventajas:

- Es un método que permite acceso a nivel del controlador de video, o “más cercano al metal”.
- Brinda al desarrollador un mayor control.
- Puede ser un sistema adecuado para sistemas de bajo costo con pocos recursos o donde se requiera reducir tanto como sea posible el costo del hardware, tales como puntos de venta, avisos automáticos, sistemas publicitarios, juguetes, o para aplicaciones en las cuales se deseé que el sistema realice una tarea concreta y bien definida, por ejemplo, instrumentos de medida.
- Es un esquema relativamente sencillo, pues el mapeo del dispositivo a un bloque de memoria simplifica enormemente el trabajo

Desventajas:

- Demanda del desarrollador un mayor esfuerzo en términos del control de la entrada/salida, pues la consola no está disponible mientras se esté en el modo gráfico.
- Las aplicaciones que hagan uso directo del framebuffer no se ejecutan bajo X, y por ello no se benefician de las ventajas de un ambiente multitarea a los ojos del usuario. Se aclara que internamente pueden ejecutarse varias aplicaciones X al mismo tiempo que una o más aplicaciones framebuffer (siempre y cuando cada una tenga su propia consola virtual). Técnicamente, dichas aplicaciones se ejecutan simultáneamente y como tales podrían intercambiar información, pero no son asequibles simultáneamente en forma gráfica al usuario.
- Es un sistema que se espera caiga en desuso, pues la dirección indicada por los desarrolladores del kernel como Tomi Valkeinen es hacia el desarrollo de drivers DRM (Direct Rendering Manager) en lugar de framebuffer [8] [9]. Una razón importante que motiva la migración es que framebuffer no resulta adecuado para aprovechar la aceleración y muchas otras características de las tarjetas gráficas modernas.
- Framebuffer se restringe, por supuesto, al aspecto gráfico. La incorporación de otras características como audio, dispositivos apuntadores, o puertos de juegos quedan a cargo del usuario.
- Es un sistema que se restringe inherentemente al ámbito de Unix/Linux.

En conclusión, se considera que los avances en la implementación de este modelo fueron fructíferos pero que, a la larga, no es el modelo más adecuado. Aunque este podría ser viable para el desarrollo de aplicaciones gráficas independientes, sacrifica la conveniencia de ejecutar varios programas al mismo tiempo como en X. Estas razones, unidas al preaviso de obsolescencia por los desarrolladores del kernel, llevan a abandonar el modelo basado en el framebuffer y a dejar en este punto la exploración de este sistema.

### 3. Instalación del sistema

La siguiente guía es basada en sistemas Linux tipo Ubuntu 18.04, Linux Mint 18.2 y 19.2.

Es una práctica recomendable actualizar el sistema:

```
sudo apt-get update
```

Opcionalmente, puede instalarse Emacs, sistema de desarrollo de gran aceptación por su flexibilidad y aplicabilidad, con el cual se ha elaborado la totalidad del proyecto.

```
sudo apt-get install emacs
```

La versión en curso del proyecto, sujeta siempre a cambios por tratarse de un desarrollo de largo plazo, está disponible en GitHub. Para descargarla, es necesario instalar primero la herramienta git, lo cual puede efecturarse de la siguiente manera:

```
sudo apt-get install git
```

La librería SDL (Simple DirectMedia Layer), la cual provee una forma razonablemente simple de acceder al hardware del sistema (video, audio, y dispositivos de entrada/salida), y que es un prerequisito del proyecto, puede instalarse de la siguiente manera:

```
sudo apt-get install libsdl2-dev
```

Finalmente, puede ser de interés exportar gráficos comprimidos en formato jpeg, y de opcionalmente encadenarlos para generar videos o animaciones. En este caso, la librería jpeg brinda una manera directa de incorporar esta capacidad al sistema:

```
sudo apt-get install libjpeg-dev
```

La librería ffmpeg puede ser de utilidad para la creación de audio o videos en una amplia variedad formatos y niveles de compresión. Para su instalación, se puede ejecutar el comando:

```
sudo apt-get install ffmpeg
```

Satisfechos los requisitos, basta hacer una copia idéntica del repositorio del proyecto en GitHub:

```
git clone https://github.com/deltavelez/gss.git
```

El proyecto queda descargado en el folder `/gss`.

## 4. Notación y convenciones

Las librerías que forman parte del proyecto tienen la notacion lb\_xxx, por ejemplo:

<lb\_graphics.c> → librería de funciones gráficas  
<lb\_complex.c> → librería de funciones en variable compleja

Los nombres de las funciones indican la librería en la cual se encuentran definidas:

```
void lb_gr_draw_circle()
```

lb: librería

gr: graphics.c

Algunas funciones especiales se anteceden por el símbolo de guión bajo '\_', indicando que son funciones reservadas o sin verificación de argumentos, las cuales deben usarse con cuidado. Si se usan incorrectamente, pueden causar errores de segmentación, derrame de memoria, pausas extendidas, o detener por completo el sistema. Por ejemplo, `_lb_gr_fb_setpixel_XRGB()` permite escribir en el framebuffer sin verificar que las coordenadas del pixel estén dentro de los confines del bloque de memoria, lo cual permite alcanzar un mejor desempeño.

## 5. Casos de uso

En esta sección se presentan tres casos que demuestran la aplicabilidad del sistema a la solución de problemas en diferentes áreas de la ingeniería. El código asociado a estos casos se encuentra disponible en GitHub (<https://github.com/deltavelez/gss.git>) en el archivo `<demos.c>`. Las secciones correspondientes a los casos de estudio se encuentran bajo las etiquetas `CASE_N_BODY_PROBLEM`, `CASE_LEVER`, y `CASE_DEMODULATION_MONTECARLO`. El resto del archivo contiene demostraciones menores y ejemplos de pruebas.

Se han buscado problemas no triviales que se beneficien de algunas características gráficas o numéricas del sistema y cuya solución equivalente sería dispendiosa si no se hiciera uso de estas. Se aclara, sin embargo, que el énfasis en el marco de este documento se hace en la descripción y justificación matemática de los problemas más no en el diseño de algoritmos ni en su codificación. Se considera conveniente detallar y justificar los casos de estudio mas no el código porque a la larga existen diversas formas de resolver cada uno de los problemas, y muchas más aún de codificarlas, según el estilo, técnicas y preferencias de cada uno. La descripción de los problemas, por consiguiente, intenta solamente reflejar la esencia de lo que se está haciendo.

En forma adicional a los tres casos de uso que fueron propuestos, se presenta una aplicación de industria que surgió durante el curso del proyecto: un sistema para el control de una máquina de pruebas destructivas de dispositivos.

### 5.1. Caso de uso: el problema de los n-cuerpos

El código fuente para una solución a este problema se encuentra en el archivo de demostraciones `<demos.c>` en la sección `CASE_N_BODY_PROBLEM`. Para habilitar la demostración se debe incluir la instrucción `#define CASE_N_BODY_PROBLEM` (y deshabilitar cualquier otra demostración que pudieran estar siendo ejecutada).

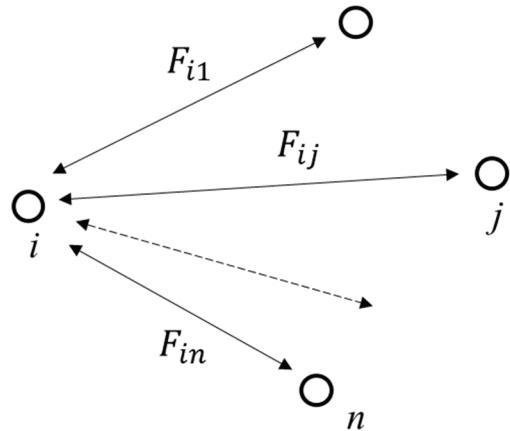
El problema de los n-cuerpos es un caso clásico en física mecánica. Consiste en un sistema de objetos, cada uno con una masa determinada, y condiciones iniciales de posición y velocidad conocidas; y que están sujetos únicamente a la atracción gravitatoria entre ellos mismos.

Desde el punto de vista computacional, el problema es interesante porque permite explorar el caso de sistemas de muchas partículas interactuando entre sí por fuerzas gravitatorias. Con este tipo de simulación es posible calcular los movimientos de los planetas y trayectorias espaciales para sondas y satélites. Este es también un caso de interés en ciencia computacional porque permite examinar en forma cualitativa el desempeño o límites de un sistema determinado.

Se presenta a continuación una solución numérica al problema. Puede considerarse de tipo “básico”, pues existen innumerables mejoras que van desde el uso de métodos más refinados que los de Euler y Runge-Kutta-4; optimizaciones considerando principios físicos como la conservación de energía; y mejoras por agrupación por “clústeres”, ésta última cuando un grupo de cuerpos está lo suficientemente cerca entre sí en relación a otros.

El método a seguir asume los cuerpos como partículas, lo cual de entrada descarta su aplicabilidad para aplicaciones que requieran alta precisión, como, por ejemplo, el cómputo de trayectorias de satélites en órbita baja<sup>3</sup> alrededor de la Tierra, la cual (al igual que cualquier otro objeto espacial) no tiene una distribución de masa uniforme. Otros supuestos, comprobados incorrectos por la física, son que las fuerzas actúan instantáneamente, y que las partículas no tienen límites de velocidad, ni de masas.

En un sistema de  $n$ -partículas, una partícula  $i$  determinada está sujeta a las fuerzas ejercidas sobre ella por todas las demás partículas.



**Figura 6:** Interacción gravitatoria entre partículas. [Recurso propio].

El efecto neto  $\vec{F}_i$  sobre la partícula  $i$  es la suma vectorial de las fuerzas  $\vec{F}_{ij}$  (siempre que  $i \neq j$ ):

$$\vec{F}_i = \sum_{\substack{j=1 \\ i \neq j}}^n \vec{F}_{ij}$$

La interacción gravitatoria entre dos partículas  $i$  y  $j$  en un sistema de  $n$ -partículas está dada por la ley de Newton:

<sup>3</sup> Aquellas ubicadas a una distancia inferior a 2,000 Km de la superficie de la Tierra. El acrónimo en inglés es LEO, Low Earth Orbit.

$$\vec{F}_{ij} = G \frac{m_i \cdot m_j (\vec{P}_j - \vec{P}_i)}{|\vec{P}_j - \vec{P}_i|^3}$$

Donde:

- $F_{ij}$  Es la fuerza ejercida sobre  $i$  por la partícula  $j$
- $m_n$  Es la masa de cada partícula  $n$
- $G$  Es la constante gravitatoria universal,  $6.67408 \cdot 10^{-11} \frac{N \cdot m^2}{Kg^2}$
- $P_n$  Es el vector de posición de la partícula  $n$

Por consiguiente, la fuerza neta es:

$$\vec{F}_i = \sum_{\substack{j=1 \\ i \neq j}}^n G \frac{m_i \cdot m_j (\vec{P}_j - \vec{P}_i)}{|\vec{P}_j - \vec{P}_i|^3}$$

Las fuerzas causan una aceleración sobre la partícula  $i$  en forma inversamente proporcional a su masa:

$$\begin{aligned} \vec{F} &= m \cdot \vec{a} \\ \vec{a}_i &= \frac{1}{m_i} \sum_{\substack{j=1 \\ i \neq j}}^n G \frac{m_i \cdot m_j (\vec{P}_j - \vec{P}_i)}{|\vec{P}_j - \vec{P}_i|^3} \\ \vec{a}_i &= \sum_{\substack{j=1 \\ i \neq j}}^n G \frac{m_j (\vec{P}_j - \vec{P}_i)}{|\vec{P}_j - \vec{P}_i|^3} \end{aligned}$$

Ahora bien, la posición está dada por la ecuación diferencial:

$$\frac{d^2}{dt^2} \vec{P}_j(t) = \vec{a}_i$$

Esta ecuación tiene solución analítica para  $n = 2$ ; posee soluciones restringidas para  $n = 3$  (por ejemplo, cuando se considera un objeto como “fijo” por ser más masivo que los demás); pero no existen soluciones generales para  $n \geq 3$ , por lo cual el problema se debe tratar numéricamente. La solución, a propósito, es caótica, pues su error aumenta exponencialmente con el número de pasos.

Se delinean a continuación los pasos fundamentales para la simulación, y se compararán soluciones con los métodos de Euler y Runge-Kutta 4. El código detallado se encuentra en los archivos del proyecto.

### 5.1.1. Algoritmo para solución por el Método de Euler:

1. Se define una estructura de datos adecuada para almacenar las variables de las partículas y se crea un arreglo con ‘n’ campos:

$$ASTRO\_T = \begin{cases} \vec{P} & \text{masa} \\ \vec{V} & \text{posición (vector 3D)} \\ \vec{V}^* & \text{velocidad (vector 3D)} \\ \vec{a} & \text{velocidad (vector 3D, variable auxiliar)} \\ & \text{aceleración (vector 3D, variable auxiliar)} \end{cases}$$

2. Para cada partícula, se cargan las masas y condiciones iniciales de posición y velocidad. Los campos libres pueden cargarse con cero.

$$\begin{bmatrix} m_1 & m_2 & m_3 & \cdots & m_i & \cdots & m_n \\ \vec{P}_1 & \vec{P}_2 & \vec{P}_3 & \cdots & \vec{P}_i & \cdots & \vec{P}_n \\ \vec{V}_1 & \vec{V}_2 & \vec{V}_3 & \cdots & \vec{V}_i & \cdots & \vec{V}_n \\ 0 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 & \cdots & 0 \end{bmatrix}$$

3. Para cada cuerpo  $i = 1$  hasta  $n$ , copiar cada una de las velocidades en el campo auxiliar  $\vec{V}^*$

$$\vec{V}_i^* \leftarrow \vec{V}_i$$

$$\begin{bmatrix} m_1 & m_2 & m_3 & \cdots & m_i & \cdots & m_n \\ \vec{P}_1 & \vec{P}_2 & \vec{P}_3 & \cdots & \vec{P}_i & \cdots & \vec{P}_n \\ \vec{V}_1 & \vec{V}_2 & \vec{V}_3 & \cdots & \vec{V}_i & \cdots & \vec{V}_n \\ \vec{V}_1 & \vec{V}_2 & \vec{V}_3 & \cdots & \vec{V}_i & \cdots & \vec{V}_n \\ 0 & 0 & 0 & \cdots & 0 & \cdots & 0 \end{bmatrix}$$

Para  $j = 1$  hasta  $n$ , si  $i \neq j$

$$\vec{a}_i = \sum_{\substack{j=1 \\ i \neq j}}^n G \frac{m_j (\vec{P}_j - \vec{P}_i)}{|\vec{P}_j - \vec{P}_i|^3}$$

$$\begin{bmatrix} m_1 & m_2 & m_3 & \cdots & m_i & \cdots & m_n \\ \vec{P}_1 & \vec{P}_2 & \vec{P}_3 & \cdots & \vec{P}_i & \cdots & \vec{P}_n \\ \vec{V}_1 & \vec{V}_2 & \vec{V}_3 & \cdots & \vec{V}_i & \cdots & \vec{V}_n \\ \vec{V}_1 & \vec{V}_2 & \vec{V}_3 & \cdots & \vec{V}_i & \cdots & \vec{V}_n \\ a_1 & a_2 & a_3 & \cdots & a_i & \cdots & a_n \end{bmatrix}$$

4. Para cada partícula  $i = 1$  hasta  $n$ , integrar la posición y la velocidad a partir, respectivamente, de la velocidad y de la aceleración.

$$\frac{d^2\vec{P}_i}{dt^2} = \vec{a}_i$$

Al ser una ecuación diferencial de segundo orden, se puede hacer uso de una sustitución de variable para descomponerla en un sistema de dos ecuaciones diferenciales de primer orden. Sobre cada una de ellas, es posible aplicar los métodos numéricos convencionales:

Sea:

$$\frac{d\vec{P}_i}{dt} = \vec{V}_i^* \quad \text{Ecuación (1)}$$

$$\frac{d}{dt}(\vec{V}_i^*) = \vec{a}_i \quad \text{Ecuación (2)}$$

Siguiendo el método de Euler, se tiene el siguiente sistema de ecuaciones recurrentes:

$$\vec{P}_i|_{t+\Delta t} \leftarrow \vec{P}_i|_t + \Delta t \cdot \vec{V}_i^*|_t \quad \text{Integración de la Ecuación (1)}$$

$$\vec{V}_i^*|_{t+\Delta t} \leftarrow \vec{V}_i^*|_t + \Delta t \cdot \vec{a}_i|_t \quad \text{Integración de la Ecuación (2)}$$

El valor de la variable auxiliar  $V^*$  puede ahora almacenarse en el campo  $\vec{V}$ :

$$\vec{V}|_{t+\Delta t} \leftarrow \vec{V}_i^*|_{t+\Delta t}$$

Almacenar, imprimir o graficar el resultado de la posición  $\vec{P}$  y velocidad  $\vec{V}$ .

5. Repetir para el siguiente instante de tiempo (ir a Paso 3)

### 5.1.2. Algoritmo para solución por el método de Runge-Kutta 4.

En general, el planteamiento del problema es el mismo que en el caso de Euler y solamente difiere en las fórmulas recurrentes. Para una descripción detallada del método de RK4, se sugiere el clásico texto de Burden Faires [10], o consultar recursos en línea como la página web del profesor Robert B. Israel de la Universidad de British Columbia [11].

Recordemos que el método RK4 consiste en evaluar 4 predictores y ponderarlos. Para una ecuación diferencial de primer orden, normalmente se presenta el método de la siguiente manera:

$$\frac{dy}{dt} = f(t, y) \quad y(t_0) = y_0$$

Se calculan los coeficientes:

$$k_1 = \Delta t \cdot f(t_n, y_n)$$

$$k_2 = \Delta t \cdot f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{k_1}{2}\right)$$

$$k_3 = \Delta t \cdot f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{k_2}{2}\right)$$

$$k_4 = \Delta t \cdot f(t_n + \Delta t, y_n + k_3)$$

Se ponderan los coeficientes para llegar al siguiente valor de la solución:

$$y|_{t+\Delta t} = y|_t + \frac{1}{6}(k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4)$$

Para aplicar el método RK4 a una ecuación diferencial vectorial de segundo orden, nuevamente hacemos uso de la descomposición en un sistema de ecuaciones diferenciales de primer orden:

$$\frac{d\vec{P}}{dt} = \vec{V}^* \quad \text{Ecuación (1)}$$

$$\frac{d}{dt}(\vec{V}^*) = \vec{a}_i \quad \text{Ecuación (2)}$$

Para la ecuación (1), se tiene, según el método RK4:

$$\overrightarrow{k_{1a}} = \Delta t \cdot \vec{V}^*(t, \vec{P})$$

$$\overrightarrow{k_{2a}} = \Delta t \cdot \overrightarrow{V^*} \left( t + \frac{\Delta t}{2}, \vec{P} + \frac{\overrightarrow{k_{1a}}}{2} \right)$$

$$\overrightarrow{k_{3a}} = \Delta t \cdot \overrightarrow{V^*} \left( t + \frac{\Delta t}{2}, \vec{P} + \frac{\overrightarrow{k_{2a}}}{2} \right)$$

$$\overrightarrow{k_{4a}} = \Delta t \cdot \overrightarrow{V^*} (t + \Delta t, \vec{P} + \overrightarrow{k_{3a}})$$

$$\vec{P}|_{t+\Delta t} = \vec{P}|_t + \frac{1}{6} (\overrightarrow{k_{1a}} + 2 \cdot \overrightarrow{k_{2a}} + 2 \cdot \overrightarrow{k_{3a}} + \overrightarrow{k_{4a}})$$

Como la velocidad  $\overrightarrow{V^*}$  no es una función del tiempo ni de la posición, se tiene:

$$\overrightarrow{k_{1a}} = \overrightarrow{k_{2a}} = \overrightarrow{k_{3a}} = \overrightarrow{k_{4a}} = \Delta t \cdot \overrightarrow{V^*}|_t$$

$$\vec{P}|_{t+\Delta t} = \vec{P}|_t + \frac{1}{6} (\Delta t \cdot \overrightarrow{V^*}|_t + 2 \cdot \Delta t \cdot \overrightarrow{V^*}|_t + 2 \cdot \Delta t \cdot \overrightarrow{V^*}|_t + \Delta t \cdot \overrightarrow{V^*}|_t)$$

$$\vec{P}|_{t+\Delta t} = \vec{P}|_t + \frac{1}{6} (6 \cdot \Delta t \cdot \overrightarrow{V^*}|_t)$$

$$\vec{P}|_{t+\Delta t} = \vec{P}|_t + \Delta t \cdot \overrightarrow{V^*}|_t$$

El resultado anterior, obtenido por la técnica RK4, equivale a una aproximación por el método Euler para la primera ecuación del sistema. Para la segunda, no es posible hacer esta simplificación pues la aceleración de cada cuerpo es función de su posición con relación a la de los demás:

$$\frac{d}{dt} (\overrightarrow{V^*}) = \overrightarrow{a}_l \quad \text{Ecuación (2)}$$

$$\overrightarrow{k_{1b}} = \Delta t \cdot a(t, \overrightarrow{V^*})$$

$$\overrightarrow{k_{2b}} = \Delta t \cdot a \left( t + \frac{\Delta t}{2}, \overrightarrow{V^*} + \frac{\overrightarrow{k_{1b}}}{2} \right)$$

$$\overrightarrow{k_{3b}} = \Delta t \cdot a \left( t + \frac{\Delta t}{2}, \overrightarrow{V^*} + \frac{\overrightarrow{k_{2b}}}{2} \right)$$

$$\overrightarrow{k_{4b}} = \Delta t \cdot a(t + \Delta t, \overrightarrow{V^*} + \overrightarrow{k_{3b}})$$

$$\overrightarrow{V^*}|_{t+\Delta t} = \overrightarrow{V^*}|_t + \frac{1}{6}(\overrightarrow{k_{1b}} + 2 \cdot \overrightarrow{k_{2b}} + 2 \cdot \overrightarrow{k_{3b}} + \overrightarrow{k_{4b}})$$

Fórmulas recurrentes para la integración por el método de Runge - Kutta 4:

$$\vec{P}|_{t+\Delta t} = \vec{P}|_t + \Delta t \cdot \overrightarrow{V^*}|_t$$

$$\overrightarrow{V^*}|_{t+\Delta t} = \overrightarrow{V^*}|_t + \frac{1}{6}(\overrightarrow{k_{1b}} + 2 \cdot \overrightarrow{k_{2b}} + 2 \cdot \overrightarrow{k_{3b}} + \overrightarrow{k_{4b}})$$

Se actualiza el campo de velocidad  $\vec{V}$  con base en el valor de la variable auxiliar  $\overrightarrow{V^*}$ :

$$\vec{V}|_{t+\Delta t} \leftarrow \overrightarrow{V^*}|_{t+\Delta t}$$

### **Resultados:**

Existen muchas formas de examinar la validez del método y del código que lo soporte.

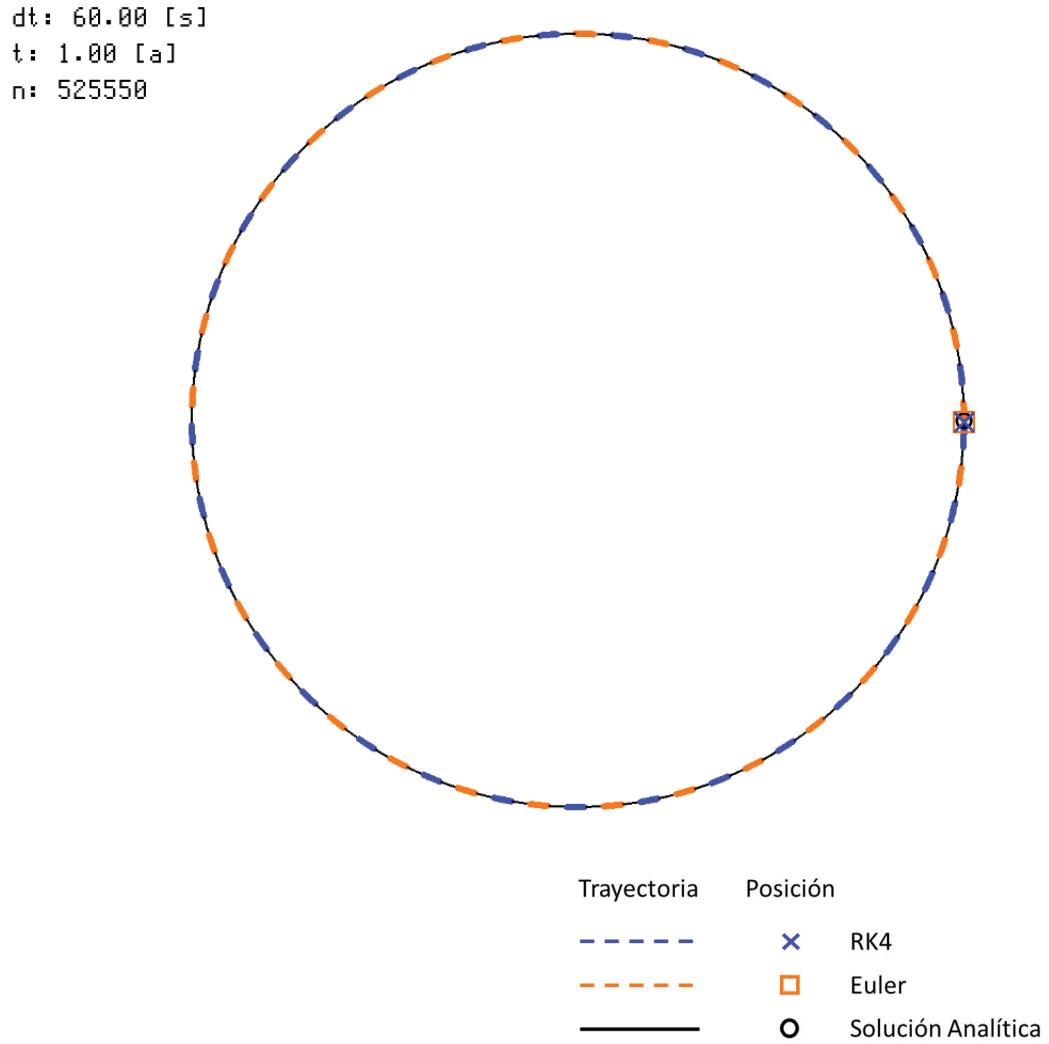
En su forma más elemental, podrían definirse únicamente dos objetos: uno con la masa de la Tierra y otro con una masa arbitraria pero mucho menor (por ejemplo, 1 Kg), separados una distancia relativamente pequeña (por ejemplo, 500 metros) y en condiciones de reposo. El sistema debe generar los resultados de posición y velocidad de un movimiento de caída libre, problema usualmente estudiado en los primeros cursos de física.

El método de validación seguido, sin embargo, consiste en definir dos objetos: el Sol y la Tierra, cada uno con sus masas reales y la distancia media entre ellos. Definidos estos datos, se provee a la Tierra de la velocidad que debería mantener una órbita circular:

$$v = \sqrt{\frac{G \cdot M_{Sol}}{d}}$$

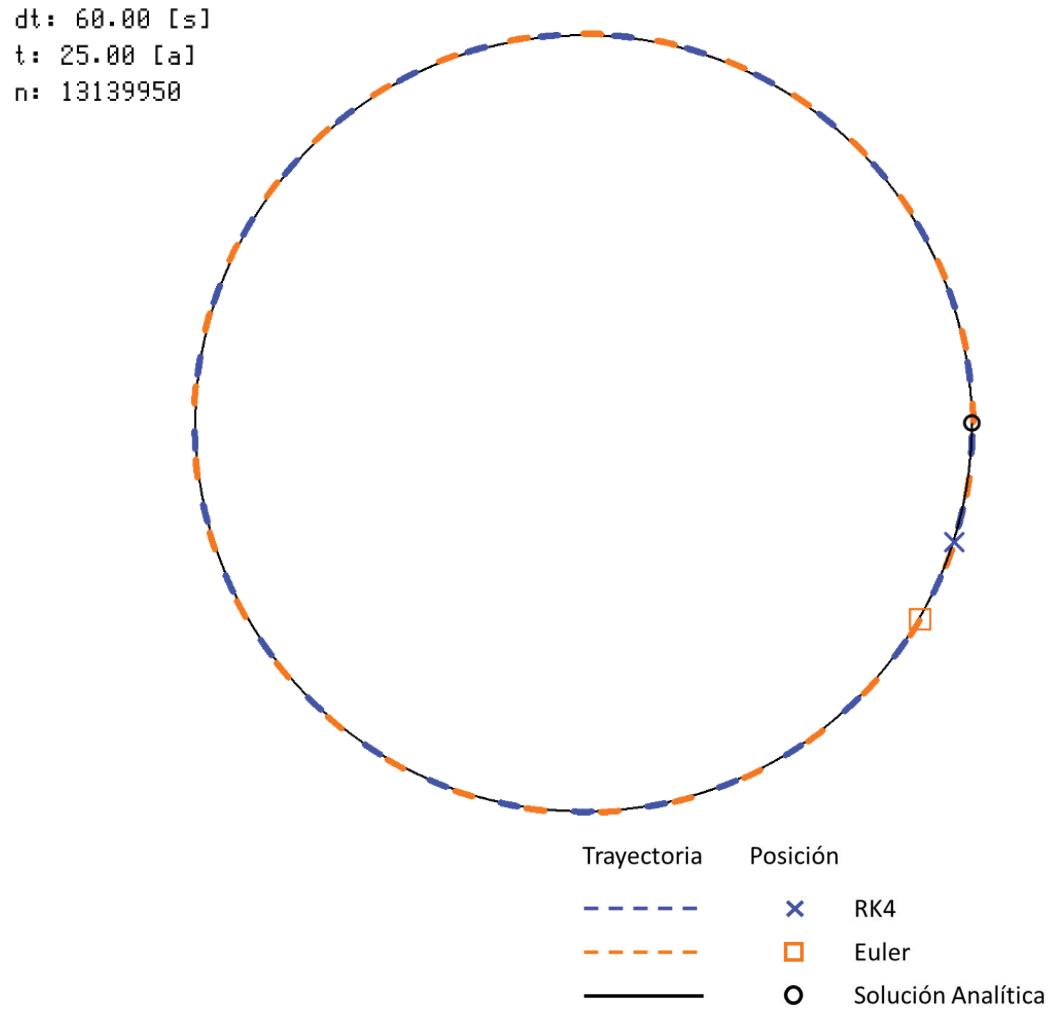
Como se conoce, una simulación de este tipo basada en métodos numéricos empieza a acumular errores que a la larga afectarán la órbita. Dado que el problema se ha resuelto tanto por el método de Euler como por el de RK4, es posible comparar su comportamiento.

Al cabo de la primera revolución (es decir, un año, o exactamente 525,960 pasos), para un sistema Sol-Tierra se puede apreciar que los resultados concuerdan entre sí, al menos al nivel de detalle planteado en la gráfica. Nota: posteriormente se discuten y cuantifican los errores.



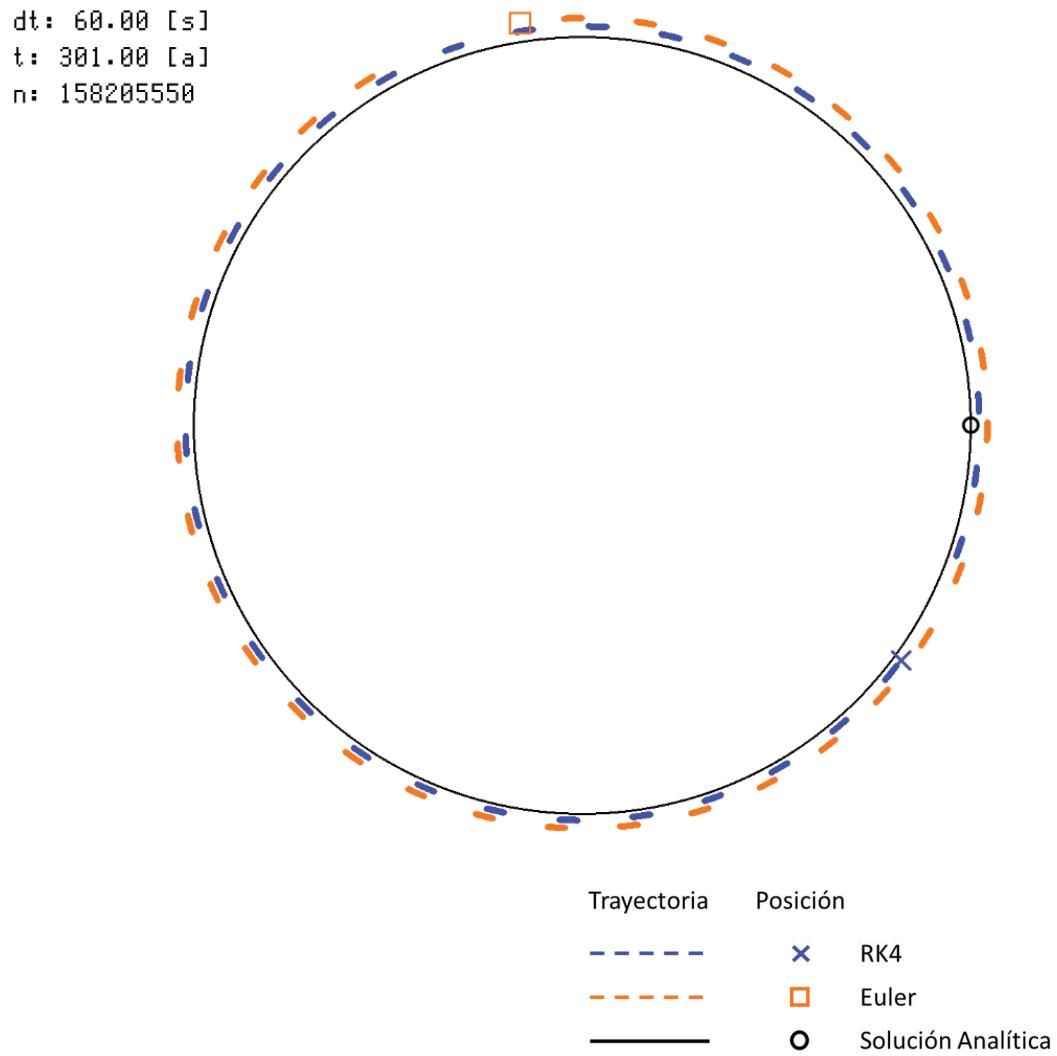
**Figura 7:** Resultados de la simulación al cabo de un tiempo de simulación de 1 año.  
[Recurso propio].

Al cabo de 25 períodos o “años”, si bien se conserva el radio orbital, se aprecia un rezago en las posiciones, el cual es mayor para el método de Euler que para RK4 (se sigue la convención antihoraria para la rotación).



**Figura 8:** Resultados al cabo de 25 años de la misma simulación. [Recurso propio].

Al cabo de 300 años, es apreciable la pérdida de energía en el objeto “satélite”, la cual se debe a factores tales como errores de truncamiento, limitaciones en la precisión de las funciones y de los tipos de datos, paso de integración demasiado grande para la escala de tiempo deseada, y otros. Para el caso de un objeto con una masa muy grande con relación a las demás, tal como el Sistema Solar, es posible efectuar mejoras de tipo físico basadas en la renormalización de la energía (es decir, ajustando la velocidad) en cada paso de integración. Otro factor de divergencia entre la solución analítica y las numéricas es el movimiento que la propia Tierra y otros cuerpos causan sobre el Sol.



**Figura 9:** Resultados de la simulación al cabo de 300 “años”. [Recurso propio].

### 5.1.3. Análisis del error y comparación entre métodos

Las técnicas numéricas para la solución de ecuaciones diferenciales tienen inherentemente errores, los cuales suelen aumentar en la medida que se avanzan los pasos de integración. Para el problema de análisis de órbita, el método RK4 presentó un mejor comportamiento que el método de Euler.

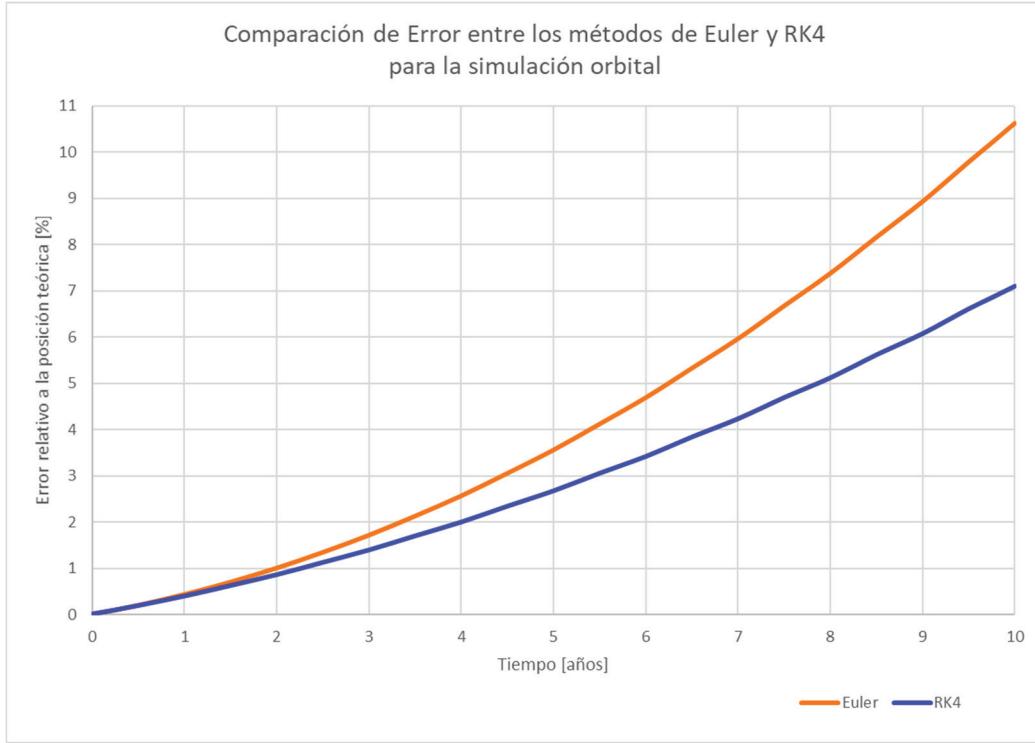
Se presentan a continuación los resultados de error obtenidos para la posición, con intervalos de media revolución o año, en los cuales se compara porcentualmente la norma (vectorial) de la diferencia entre la posición numérica y la analítica, con la norma del vector para la posición analítica. Este tipo de error entre “normas” asume siempre valores positivos.

$$e = \frac{\|\overrightarrow{P_{\text{numérica}}} - \overrightarrow{P_{\text{analítica}}}\|}{\|P_{\text{analítica}}\|} \cdot 100\%$$

Se aclara, sin embargo, que es necesario ser cuidadosos con las comparaciones basadas en distancia para sistemas en rotación, pues en la medida en que progresá la simulación, la solución obtenida mediante el método con menor pérdida de energía (RK4) alcanzará angularmente a la más rezagada (Euler), indicando una disminución aparente del error. Un ejemplo cotidiano de lo anterior se observa en las carreras de automóviles o de ciclismo, en las cuales los competidores aventajados se empiezan a acercar nuevamente a quienes van de últimos.

Tiempo (años)	% Error Euler	% Error RK4
0.00	0.00	0.00
0.50	0.19	0.19
1.00	0.43	0.39
1.50	0.69	0.62
2.00	1.00	0.86
2.50	1.34	1.12
3.00	1.71	1.39
3.50	2.12	1.69
4.00	2.56	2.00
4.50	3.05	2.34
5.00	3.55	2.67
5.50	4.12	3.05
6.00	4.69	3.42
6.50	5.33	3.84
7.00	5.96	4.23
7.50	6.67	4.69
8.00	7.38	5.12
8.50	8.16	5.62
9.00	8.93	6.08
9.50	9.79	6.62
10.00	10.63	7.11

**Tabla 1:** Comparación de error entre los métodos de Euler y RK4. [Recurso propio.]



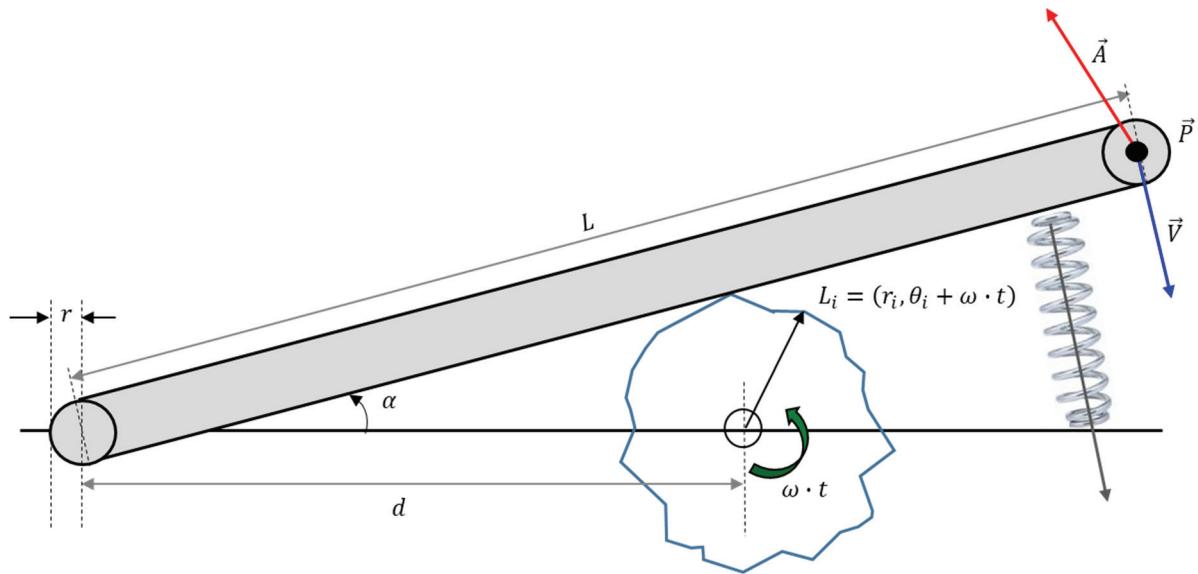
**Figura 10:** Comparación de error entre los métodos de Euler y RK4 para la simulación orbital. [Recurso propio].

## 5.2. Caso de uso: estudio dinámico de mecanismo de levas.

El código fuente para una solución a este problema se encuentra en el archivo de demostraciones `<demos.c>` en la sección `CASE_LEVER`.

En este caso se aplican varias características del sistema a la solución de un problema de física dinámica.

El mecanismo de levas presentado a continuación puede utilizarse para obtener, en forma repetitiva, algún movimiento  $\vec{P}(t)$ , velocidad  $\vec{V}(t)$ , o aceleración  $\vec{A}(t)$  deseados. La geometría de la leva, la cual rota con velocidad angular  $\omega$ , determinan el movimiento. Una aplicación de dicho mecanismo puede ser alcanzar cierto nivel de aceleración para efectuar pruebas de resistencia a algún producto o componente.



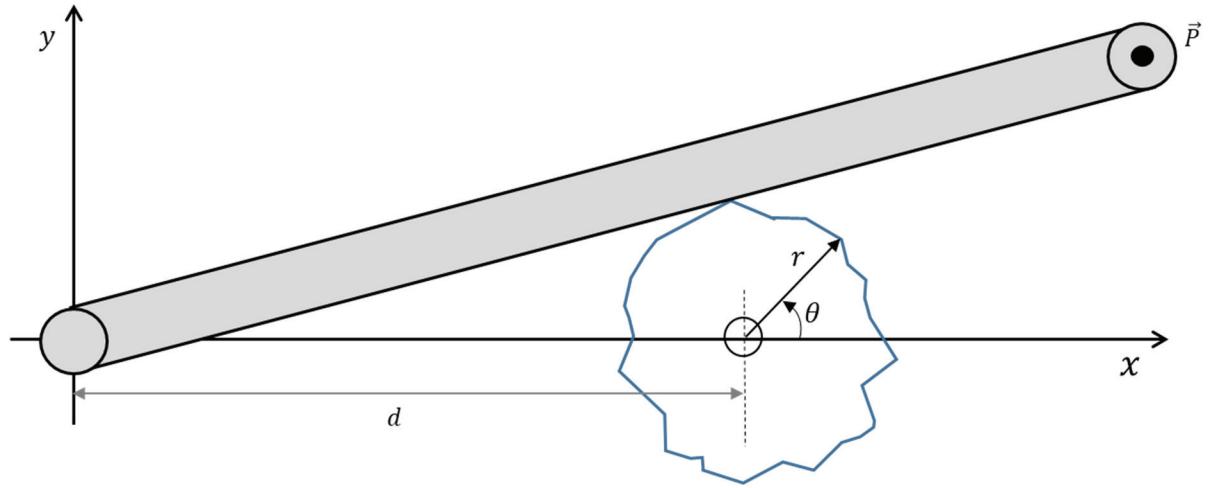
**Figura 11:** Mecanismo de leva cuya solución se aborda mediante el sistema propuesto.  
[Recurso propio].

Resolver analíticamente este sistema, es decir, la obtención de  $\vec{P}(t)$ ,  $\vec{V}(t)$ , y  $\vec{A}(t)$  para este sistema, es un trabajo arduo, sobre todo cuando la leva presenta formas complejas, discontinuas, o irregularidades. La solución por medio de una simulación resulta de interés por lo simple que resulta, y porque permite variar en forma práctica muchos tipos de parámetros y comparar los resultados.

Se asumen algunos supuestos para simplificar el problema:

- El brazo, de longitud  $L$ , es “de un material infinitamente rígido, resistente y ultraligero”.
- Se aplica, por medio de un resorte extendido, una “gran fuerza” al brazo, el cual lo mantiene en contacto con la leva. Adicionalmente, la velocidad angular de la leva es lo suficientemente baja como para ignorar la inercia del brazo (la cual es baja por tratarse de un material ultraligero).
- Se ignoran efectos de deformación, oscilación, y variación de longitud por efectos térmicos.

Para simular este sistema en una forma sencilla, es fundamental elegir adecuadamente los sistemas coordenados. Para este caso se definirán dos: uno rectangular, con el origen situado en el extremo fijo del brazo; y otro polar para la leva, centrado a una distancia  $d$  respecto al rectangular.



**Figura 12:** Elección de los sistemas coordinados para el mecanismo de leva. [Recurso propio].

La geometría de la leva, la cual es arbitraria, puede especificarse como un vector de pares radiales y angulares:

$$Leva = \begin{bmatrix} L_0 \\ L_1 \\ \vdots \\ L_n \end{bmatrix} = \begin{bmatrix} r_0 & \theta_0 \\ r_1 & \theta_1 \\ \vdots & \vdots \\ r_n & \theta_n \end{bmatrix}$$

Si se considerara que el brazo como “infinitamente delgado” (supuesto que no se seguirá), la posición  $\vec{P}$  para cada instante de tiempo podría obtenerse mediante el siguiente procedimiento:

Algoritmo para simulación del mecanismo de leva simplificado (brazo delgado):

Condiciones iniciales:

$$\begin{aligned} P_{x\_prev} &\leftarrow 0 \\ P_{y\_prev} &\leftarrow 0 \end{aligned}$$

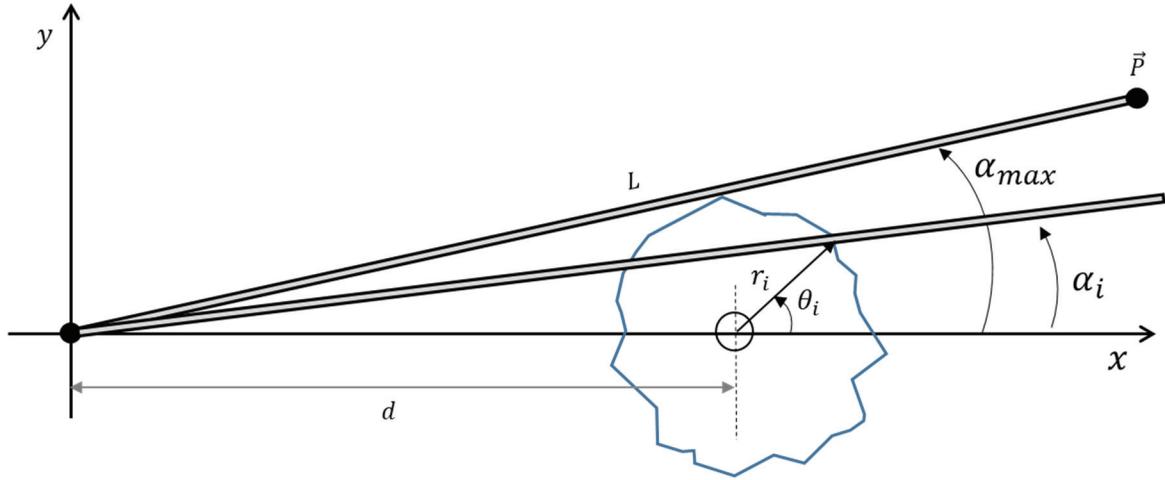
$$\begin{aligned} V_{x\_prev} &\leftarrow 0 \\ V_{y\_prev} &\leftarrow 0 \end{aligned}$$

Para  $t = 0$  hasta  $t = t_{max}$

- a) Para cada punto  $L_i$  de la leva, obtener el ángulo respecto al origen del sistema rectangular, y almacenar el mayor valor de los ángulos.

$$\alpha_i = \text{atan}\left(\frac{r_i \cdot \sin(\theta_i + \omega \cdot t)}{d + r_i \cdot \cos(\theta_i + \omega \cdot t)}\right)$$

Este  $\alpha_{max}$  determina el punto de contacto entre la leva y el brazo.



**Figura 13:** Mecanismo de leva simplificado (brazo delgado). [Recurso propio].

- b) Obtenido  $\alpha_{max}$ , el cómputo de la posición está dado simplemente por:

$$\vec{P} = \langle P_x, P_y \rangle = \langle L \cdot \cos(\alpha_{max}), L \cdot \sin(\alpha_{max}) \rangle$$

$$\vec{V} = \langle V_x, V_y \rangle = \left\langle \frac{P_x - P_{x\_prev}}{\Delta t}, \frac{P_y - P_{y\_prev}}{\Delta t} \right\rangle$$

$$\vec{A} = \left\langle \frac{V_x - V_{x\_prev}}{\Delta t}, \frac{V_y - V_{y\_prev}}{\Delta t} \right\rangle$$

- c) Almacenar los valores de velocidad en variables que reflejen los valores en el paso de tiempo inmediatamente anterior.

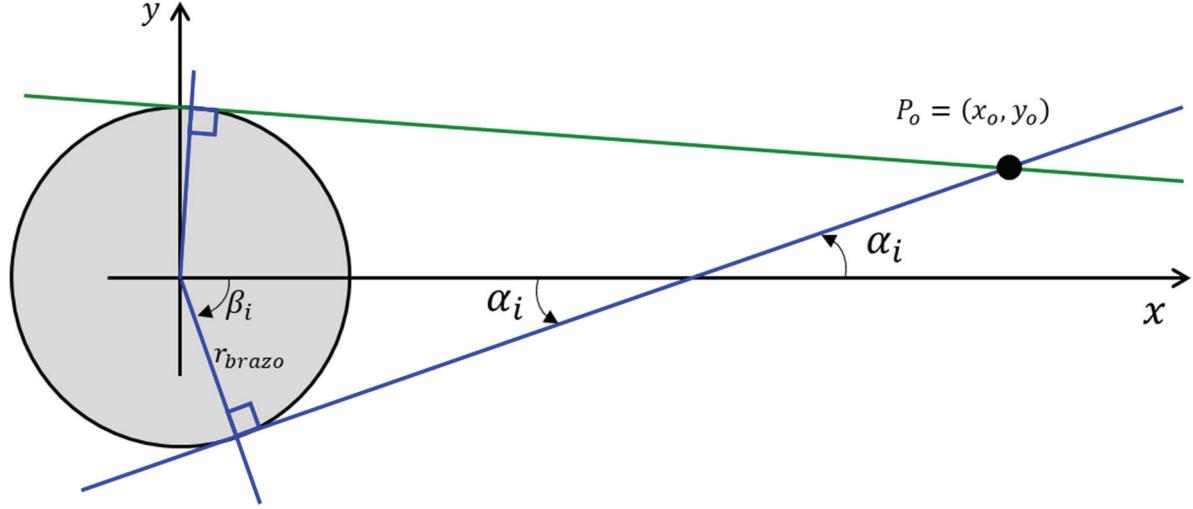
$$\begin{aligned} P_{x\_prev} &\leftarrow P_{y\_prev} \\ P_{y\_prev} &\leftarrow P_{y\_prev} \end{aligned}$$

$$\begin{aligned} V_{x\_prev} &\leftarrow V_{y\_prev} \\ V_{y\_prev} &\leftarrow V_{y\_prev} \end{aligned}$$

- d) Incrementar el tiempo:  $t \leftarrow t + \Delta t$

- e) Ir al paso a) y repetir el proceso.

El anterior es el esquema general de la simulación para el caso en el que el brazo es muy delgado. Cuando este tiene un ancho apreciable, el ángulo del brazo  $\alpha_i$  se determina con ayuda de la geometría:



**Figura 14:** Puntos de tangencia en una circunferencia dado un punto externo. [Recurso propio].

$$\alpha = \pi - \frac{\pi}{2} - \beta$$

$$\alpha = \frac{\pi}{2} - \beta$$

$\beta$  es el ángulo hasta el punto de tangencia de un círculo (con radio  $r$  y centrado en el origen), cuando la línea de tangente pasa por un punto  $P_o(x_o, y_o)$  exterior al círculo. Se puede demostrar que:

$$\beta = 2 \cdot \text{atan} \left( \frac{y_o \pm \sqrt{x_o^2 + y_o^2 - r^2}}{x_o + r} \right)$$

Nótese que la expresión ofrece dos resultados, dado que existen dos líneas tangentes a un círculo pasando por un punto. El ángulo de inclinación del brazo está dado por:

$$\alpha = \frac{\pi}{2} - 2 \cdot \text{atan} \left( \frac{y_o \pm \sqrt{x_o^2 + y_o^2 - r^2}}{x_o + r} \right)$$

$$\alpha_i = \frac{\pi}{2} - 2 \cdot \text{atan} \left( \frac{r_i \cdot \sin(\theta_i + \omega \cdot t) \pm \sqrt{[d + r_i \cdot \cos(\theta_i + \omega \cdot t)]^2 + [r_i \cdot \sin(\theta_i + \omega \cdot t)]^2 - r^2}}{d + r_i \cdot \cos(\theta_i + \omega \cdot t) + r} \right)$$

La expresión anterior modifica el paso a) del algoritmo, si se desea considerar el ancho del brazo en la simulación.

En la simulación se hace uso extensivo de varias funciones del sistema:

- Funciones gráficas para la creación (y cierre ordenado) de ventana, transformación de coordenadas reales a coordenadas de pantalla, funciones para la carga y despliegue de texto, dibujo de líneas, elipses, y vectores, con filtro anti-escalonamiento, y creación de ventanas separadas para el trazo independiente de las gráficas de velocidad y aceleración respecto al tiempo.
- Funciones para la creación en memoria dinámica de vectores.
- Función de espera de tiempo.

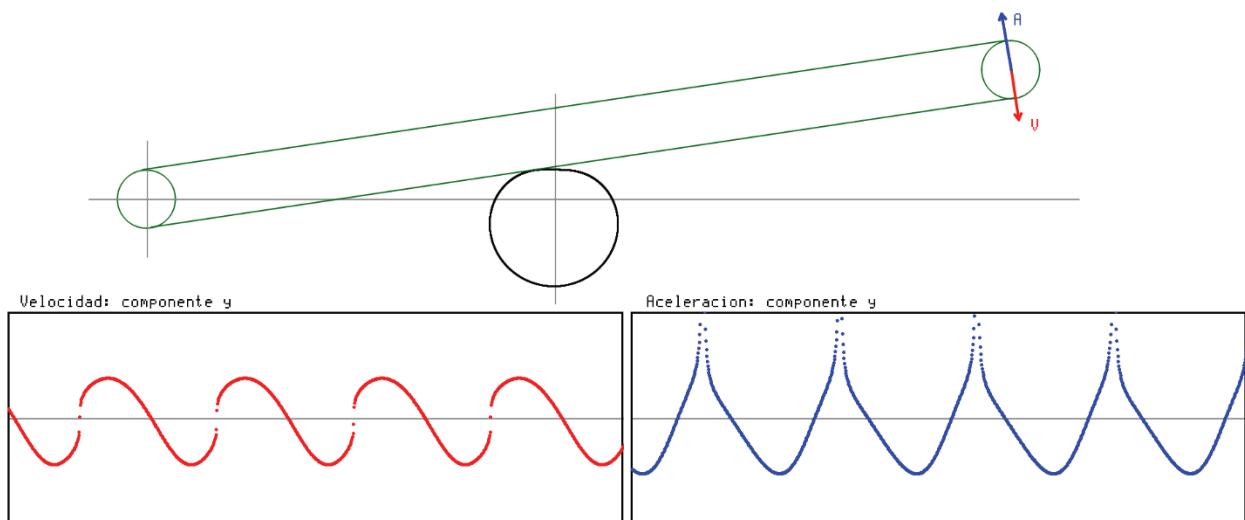
Resultados:

Al correr la simulación con una leva con forma:

$$r(\theta) = r_0 + k \cdot r_0 \cdot \sin(\theta)$$

$$k < 1$$

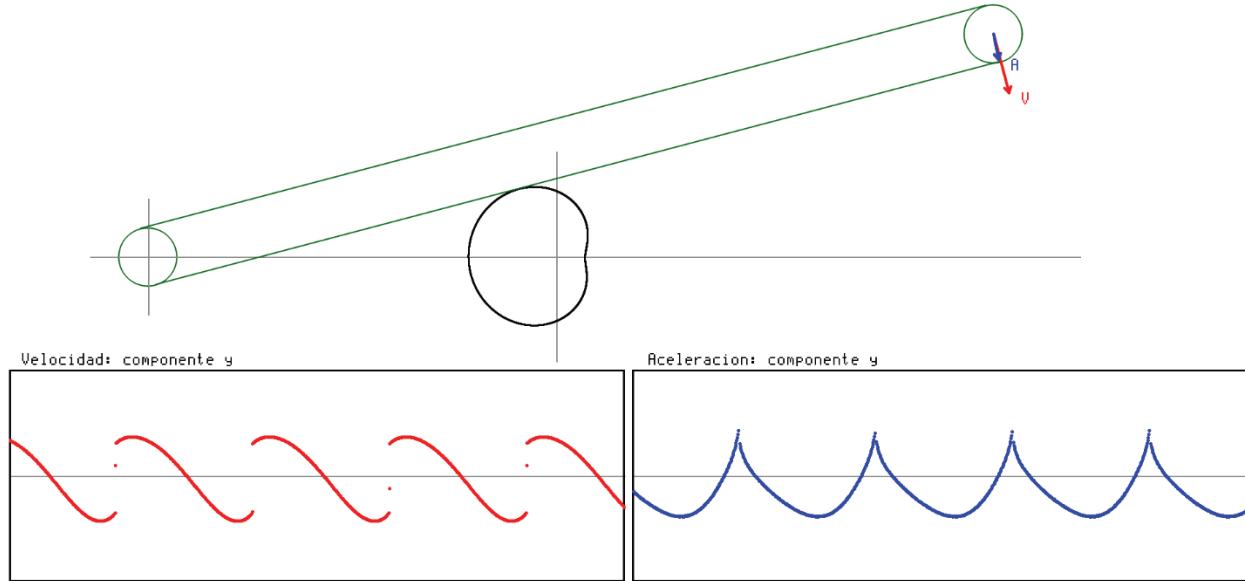
Esta leva es interesante desde el punto de vista numérico porque presenta un lado aproximadamente recto cuando  $|\theta + \omega \cdot t| \approx 0$ . De hecho, la simulación debe reflejar un salto súbito en la aceleración vertical al pasar cerca a esta sección ( $a_y \rightarrow \infty$  si el paso  $\Delta t \rightarrow 0$ ).



**Figura 15:** Resultados para una leva con forma  $r(\theta) = r_0 + k \cdot r_0 \cdot \sin(\theta)$ . [Recurso Propio].

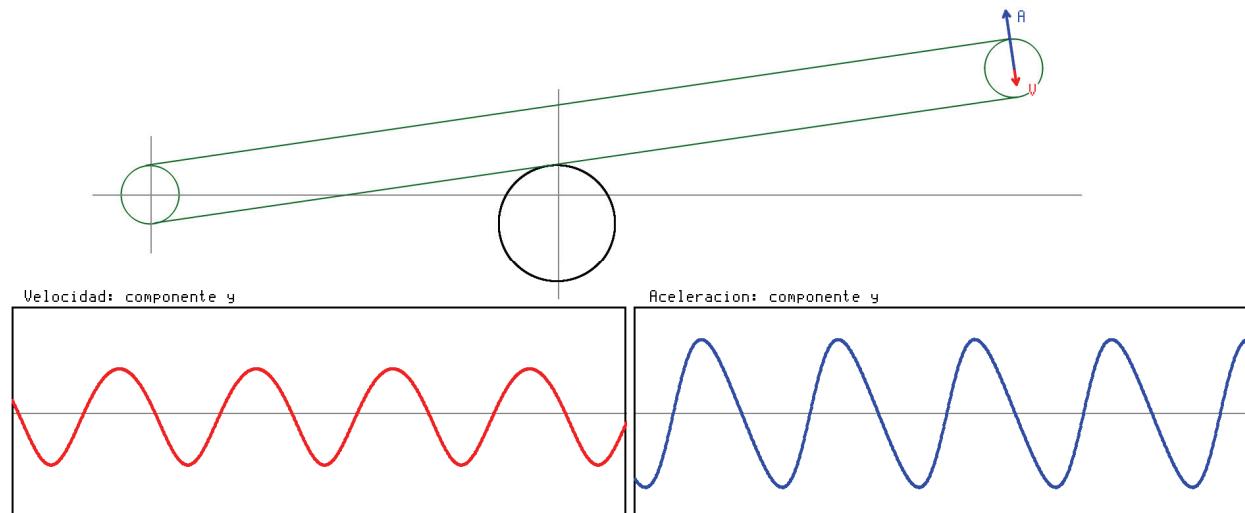
El método permite estudiar las características de un “golpe” deliberado, logrado con la forma de leva:

$$r = \sqrt{[k + r_0 \cdot \cos(\theta)]^2 + [r_0 \cdot \sin(\theta)]^2}$$



**Figura 16:** Resultados para una leva con concavidad (impacto). [Recurso propio].

Finalmente, se examina el caso de una leva circular con centro de rotación en  $x = \frac{r}{2}$



**Figura 17:** Resultados para una leva circular con centro desfasado. [Recurso propio].

En este caso no se presentan cambios súbitos de velocidad o aceleración, y se obtienen respuestas de posición, velocidad y aceleración aproximadamente senoidales.

### 5.3. Caso de uso: caracterización de modulaciones digitales arbitrarias utilizando análisis de Monte Carlo

El código fuente para una solución a este problema se encuentra en el archivo de demostraciones `<demos.c>` en la sección `CASE_DEMODULATION_MONTECARLO`.

Es posible que el asunto más importante en el estudio de las comunicaciones digitales sea determinar la probabilidad de que ocurra un error al transmitir un bit a través de un canal. Este tema se trata en detalle en libros y cursos de comunicaciones, en los cuales se presentan resultados analíticos para el desempeño de cada modulación. En breve, se trata de determinar la probabilidad de error  $P(e)$  en relación al cociente entre la energía por bit y la densidad espectral de ruido  $\frac{E_b}{N_o}$ . Tanto  $P(e)$  como  $\frac{E_b}{N_o}$  son adimensionales.

Para una modulación relativamente simple como 2-FSK, la probabilidad de error al transmitir un bit está dada por:

$$P(e)_{2FSK} = \frac{1}{2} \cdot erfc\left(\sqrt{\frac{E_b}{2 \cdot N_o}}\right)$$

La función error complementario está definido como:

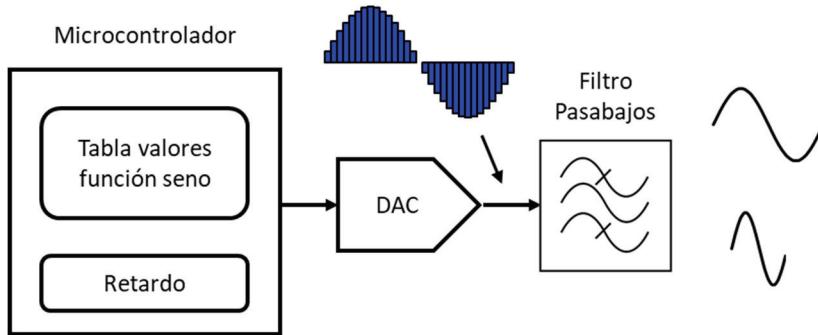
$$erfc(z) = \frac{2}{\pi} \int_z^{\infty} e^{-t^2} dt$$

El tratamiento matemático para llegar a  $P(e)$  es riguroso, y asume supuestos como una distribución gausiana del ruido (ancho de banda infinito), el cual en la práctica puede estar limitado a cierto ancho de banda.

Existen, sin embargo, métodos particulares o variaciones a las modulaciones convencionales para los cuales resulta difícil determinar analíticamente la probabilidad de error. A veces estas variaciones son consecuencia de restricciones en el diseño, de poca disponibilidad de componentes, o porque seguir una técnica no convencional simplifica el diseño.

Usualmente es más simple generar (modular) señales que decodificarlas (demodularlas). Por ejemplo, implementar un transmisor para un esquema 2-FSK (Frequency Shift Keying) en un sistema basado en microcontrolador es una tarea sencilla, pues se pueden tener una tabla de búsqueda con los valores de la función seno, los cuales se convierten en una señal análoga

haciendo uso de alguna técnica como PWM, una escalera de resistencias, o un conversor Digital-Análogo (DAC).



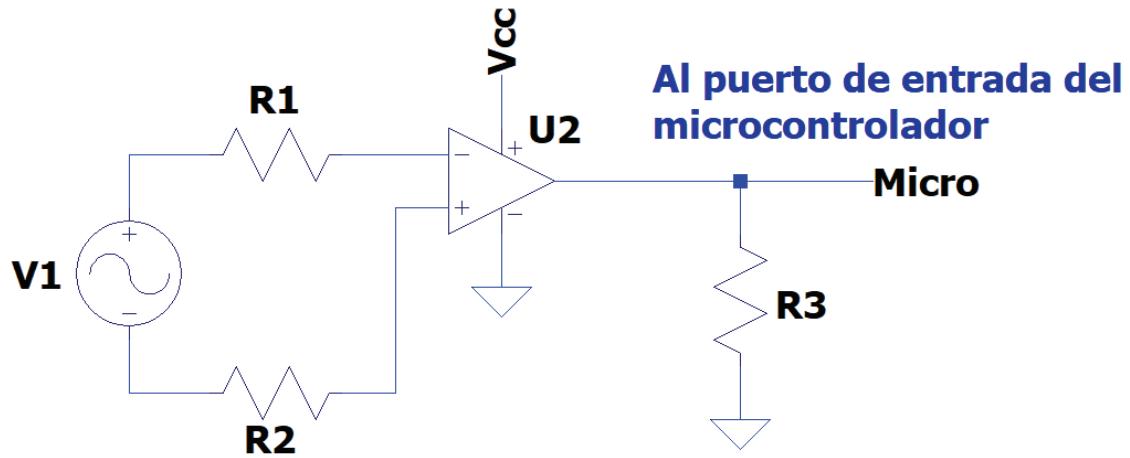
**Figura 18:** Generación simple de señales FSK en sistemas basados en microcontrolador.  
[Recurso propio].

Para cambiar de frecuencia, basta con variar el periodo de retardo entre los valores. Por ejemplo, el pseudo-código podría tener el siguiente estilo:

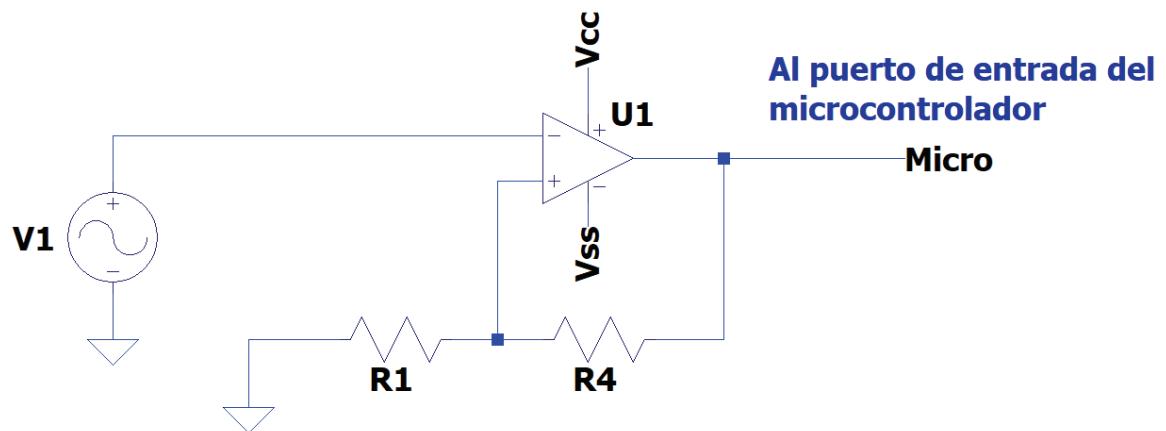
```
void send_bit(unsigned char bit)
{
    unsigned char i=0;
    unsigned char delay_value;
    if ((bit & 0x01)==0)
        delay_value=10;
    else
        delay_value=20;

    for (i=0;i<N_SINE_SAMPLES;i++)
    {
        Write_to_DAC(Sine_table[i]);
        delay(delay_value);
    }
}
```

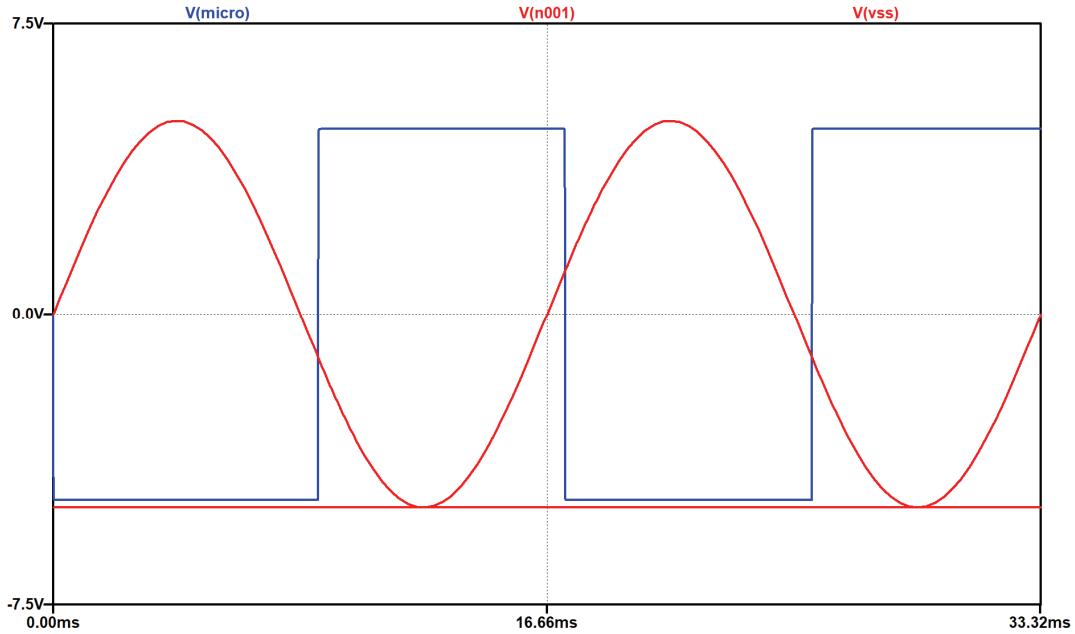
Para la demodulación, también existen varias técnicas, una de ellas consiste en crear filtros con alto  $Q$  (factor de calidad) sintonizados en las frecuencias de interés, método de alto costo y complejo por requerir componentes análogos de alta precisión, o ajustes a la medida. Otro método consiste en detectar los pasos por cero y medir el tiempo entre ellos, lo cual está muy bien (es simple de hacer digitalmente), excepto por el hecho de que, a fin de reducir falsos cruces por cero, es deseable incorporar cierta histéresis, factor que resulta muy difícil de incorporar en un cálculo analítico para la caracterización de la modulación.



**Figura 19:** Un circuito de detección de cruces por cero, sin histéresis. [Recurso propio].



**Figura 20:** Un circuito de detección de cruces por cero, con histéresis. [Recurso propio].



**Figura 21:** efecto de la adición de histéresis a un circuito de detección de cruces por cero.  
[Recurso propio].

### 5.3.1. Caracterización de la modulación por medio de una simulación

El procedimiento consiste en replicar “en software” el comportamiento del circuito de detección que se desea implementar, aplicando a este una señal de prueba con diferentes niveles de ruido. Para cada nivel de ruido se realizan “muchos experimentos”, y en cada uno de ellos se determina si una señal originalmente generada con un valor de bit es interpretada correctamente, como el bit opuesto (erróneo), o como un bit evidentemente erróneo (condición de detección de error).

De este modo, si el número de experimentos es alto, es posible estimar las probabilidades de error asociadas.

Más detalladamente:

- Se simulará una modulación tipo 2-FSK, con demodulación mediante detección de los cruces por cero, sin histéresis.
- La histéresis se omite a fin de poder comparar los resultados de la simulación con los analíticos, pero puede cuando un problema concreto así lo requiriera.

### Algoritmo:

Parámetros de entrada: frecuencias  $f_0$  y  $f_1$  y energía (o unidades relacionadas como voltaje, corriente, o potencia) de las mismas, las cuales se asumen iguales.

Condiciones iniciales: iniciar con un nivel de energía de ruido  $E$  “muy alto” por ejemplo, con la misma energía que la señal. En este tipo de simulación tipo Monte Carlo, basada en efectuar un número muy grande de experimentos (el cual podría ser  $10^6$  o mucho mayor como  $10^{10}$ ), es deseable iniciar con un nivel de ruido alto en el cual ocurran muchos errores, e irlo disminuyendo progresivamente hasta que no existan errores, deteniendo la simulación.

Definir el ancho de banda del ruido. El ancho de banda determina el tamaño del vector,  $E_n$ , el cual contendrá los coeficientes de los componentes del ruido en cada una de las frecuencias. El ancho de banda  $f_{max}$  se puede normalizar respecto a  $f_0$ ,  $f_1$ , o respecto al periodo de un ciclo de ambas  $T_{base} = \frac{1}{\frac{1}{f_0} + \frac{1}{f_1}}$  (se elige esta última).

$$N = \frac{1}{T_{base}} = \frac{f_{max}}{\left( \frac{1}{\frac{1}{f_0} + \frac{1}{f_1}} \right)}$$

A fin de aclarar este punto, un ejemplo numérico brinda mayor claridad. Si  $f_0 = 1200$  Hz,  $f_1 = 1800$  Hz, y si el ancho de banda del ruido  $f_{max} = 14400$  Hz,

$$N = \frac{14400}{\left( \frac{1}{\frac{1}{1200} + \frac{1}{1800}} \right)} = 20$$

La señal de ruido es, por consiguiente:

$$\begin{aligned} E(t) &= E_0 \cdot \sin\left(2 \cdot \pi \cdot \frac{0}{T_{base}} \cdot t\right) + E_1 \cdot \sin\left(2 \cdot \pi \cdot \frac{1}{T_{base}} \cdot t\right) + E_2 \cdot \sin\left(2 \cdot \pi \cdot \frac{2}{T_{base}} \cdot t\right) + \dots \\ &\quad + E_i \cdot \sin\left(2 \cdot \pi \cdot \frac{i}{T_{base}} \cdot t\right) + \dots + E_n \cdot \sin\left(2 \cdot \pi \cdot \frac{N}{T_{base}} \cdot t\right) \end{aligned}$$

$$E(t) = \sum_{i=0}^{N-1} E_i \cdot \sin\left(2\pi \frac{i}{T_{base}} \cdot t\right)$$

```

bandera_simular ← verdadero
E_exp ← Máxima energía de ruido

```

Mientras (bandera\_simular), hacer:

Para cada experimento  $k$  desde 0 hasta  $k_{max}$ :

- Generar un vector  $E_n$  y llenarlo con números aleatorios que sigan una distribución gaussiana, u otro tipo de distribución que se considere refleje el comportamiento del ruido en el sistema.
- El vector  $E_n$ , por contener coeficientes que fueron generados aleatoriamente, tiene una energía que difiere de  $E_{total}$ . Sin embargo,  $E_n$  puede normalizarse para que su energía de ruido corresponda al nivel deseado para cada serie de experimentos:

La energía (de ruido, en este caso) de una señal dada por un vector de coeficientes de señales senoidales, cada uno con amplitud  $E_i$ , evaluada desde  $t = 0$  hasta un instante  $T$ , está dada por:

$$E(t) = \sum_{i=0}^{N-1} \int_0^T \left| E_i \cdot \sin \left[ 2 \cdot \pi \left( \frac{i}{T_{base}} + \phi_{rand} \right) \right] \right|^2 dt$$

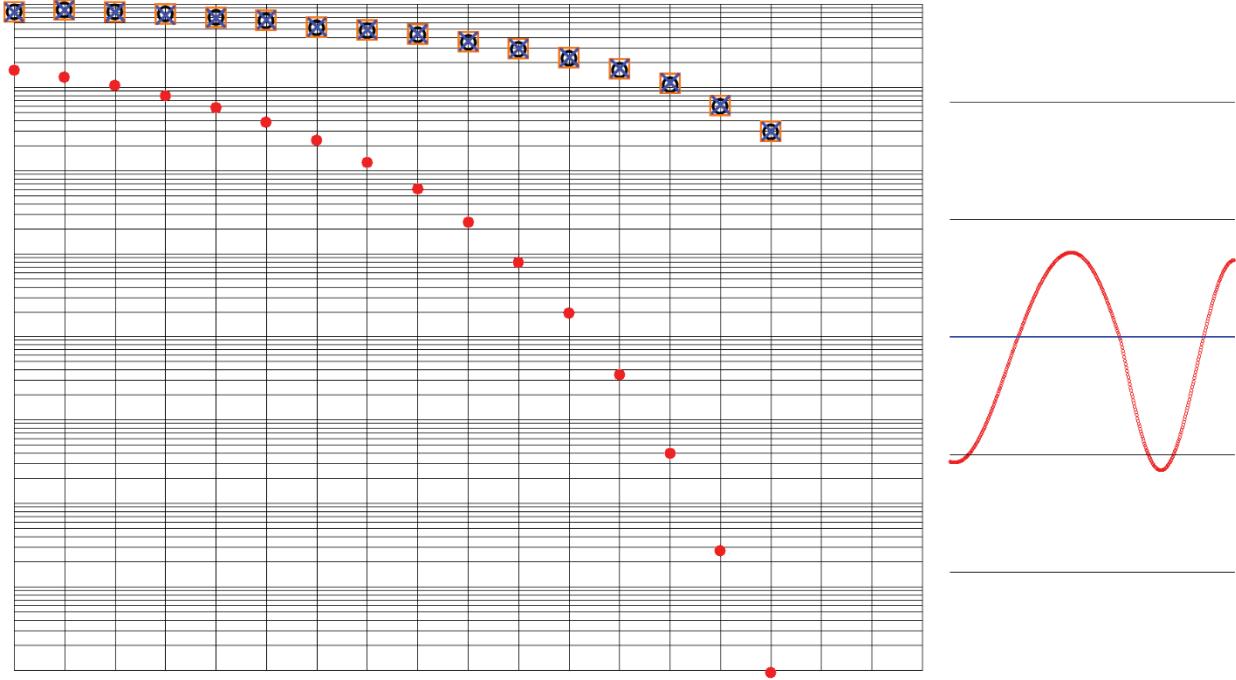
$\phi_{rand}$  es un número aleatorio entre 0.0 y 1.0 que determina la fase de la señal simulada.

Si el intervalo de integración es un múltiplo entero de los períodos fundamentales, la expresión puede simplificarse como:

$$E_{Total} = \frac{1}{2} \sum_{i=0}^{N-1} |E_i|^2$$

Cada  $E_i$  es normalizado, por consiguiente, como:

$$E_{i\_norm} = E_i \cdot \frac{E_{exp}}{E_{Total}}$$



Eje vertical:  $P(e)$  entre 1.0 y  $10^{-8}$ .  
 Eje horizontal:  $Eb/No$  entre 0.0 y -18 dB.

- Cruce por cero (2 bits, no histéresis)
- Cruce por cero (2 bits, 2.5% Vp histéresis)
- × Cruce por cero (2 bits, 5.0% Vp histéresis)
- 2-FSK (1 bit)

**Figura 22:** Curva de error obtenida para una modulación 2-FSK con detección mediante un circuito de cruces por cero. [Recurso propio].

### 5.3.2. Resultados:

Bajo las condiciones de ruido elegidas los resultados no presentan diferencias notables debidos a la adición de histéresis al detector. Bajo otras condiciones de ruido, se han encontrado diferencias que pueden sugerir la ventaja de incorporarla. Es importante aclarar que la probabilidad de éxito al transmitir dos bits es aproximadamente 3 dB inferior a la de un solo bit.

Solamente como referencia, los círculos rojos presentan los valores teóricos del éxito de transmitir un solo bit mediante una modulación 2-FSK convencional. La única razón para utilizar como demodulador un circuito de cruce por cero es la sencillez del hardware.

#### **5.4. Caso de uso: control para un sistema de pruebas de impacto para componentes electrónicos**

El desarrollo presentado a continuación no hizo parte del plan inicial para el proyecto. Durante el mes de octubre de 2019, se encomendó al grupo de diseño electrónico de la empresa Baker Hughes, donde labora el autor, la elaboración de un método o aparato que permitiera hacer pruebas de calidad a ciertos sensores que han presentado fallas, aparentemente debidas a condiciones de vibración o impacto.

Tras considerar algunas alternativas, se eligió una idea basada en un pistón activado neumáticamente, con paro súbito. Quedaba, pues, pendiente el diseño de un sistema computarizado que activara el pistón; que midiera la aceleración; que tuviera algunas facilidades como el paro de la prueba en caso de que exceda algún límite; y que ofreciera el soporte de modos de disparo manual o automático para repetir la prueba en forma continua.

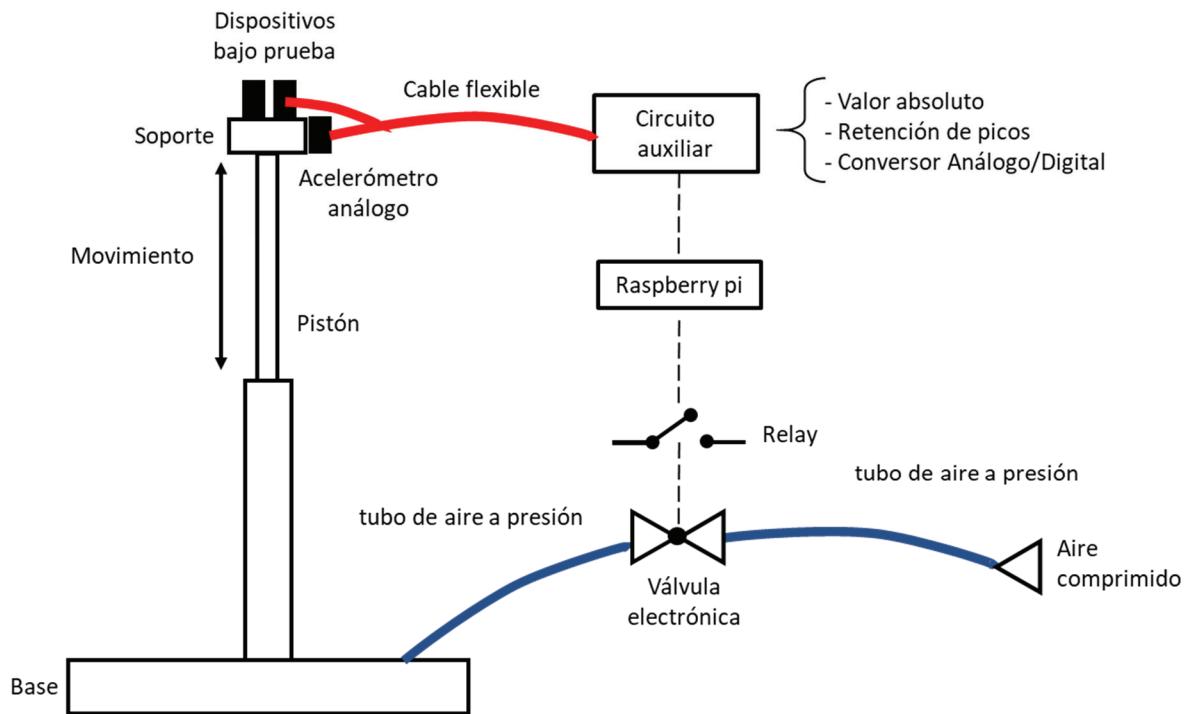
Todo lo anterior, por supuesto, es un tema laboral independiente del proyecto académico.

La ruta tradicional al interior de la empresa, seguida ya muchas veces cuando se necesita un sistema a la medida, consiste en utilizar uno de los mismos microcontroladores que se utilizan para los productos, diseñar y construir un circuito, escribir el código respectivo, y utilizar un computador convencional para la captura de datos, proyectos que típicamente toman entre cuatro y diez semanas en ser completados.

Dado el avance que se tenía en este proyecto académico a la fecha de este requerimiento, se consideró viable tratar de implementar el control del sistema neumático utilizando el software existente ejecutado en un sistema Raspberry Pi. En caso de operar adecuadamente, se ganaría información de primera mano sobre los resultados de aplicar el sistema a una necesidad concreta de industria. Para la empresa, en caso de funcionar adecuadamente, se podría reducir el tiempo de desarrollo. En caso de no alcanzar resultados óptimos para el control del sistema, se podría de todos modos volver (con algún retraso) al esquema habitual del microcontrolador y posiblemente, dentro de este proyecto, se haría una mención de los problemas encontrados y de los faltantes y trabajos adicionales necesarios para hacer aplicable el sistema a casos de industria.

Los resultados, sin embargo, fueron en general positivos, salvo un error en una rutina de temporización, el cual que se discute posteriormente, y a cierto nivel de ruido en la medida de la aceleración, debido principalmente a que la electrónica de soporte (la cual incluye circuitos análogos de valor absoluto, de retención de valor máximo, y el conversor análogo-digital) se energiza del bus digital de 3.3V, práctica que de antemano se sabe que es inadecuada pero que finalmente se aceptó.

El punto sobre el cual se desea hacer énfasis es que el sistema de software propuesto puede ser utilizado para la puesta a punto de soluciones de industria. El enfoque alternativo sería diseñar sistemas a la medida, los cuales involucran un mayor tiempo de desarrollo, o sistemas comerciales de alto costo tipo LabView o similares.



**Figura 23:** Diagrama general de un sistema neumático para pruebas de impacto. [Recurso propio].



**Figura 24:** Sistema neumático para pruebas de impacto controlado mediante el software del proyecto. [Recurso propio].

Se presenta a continuación la información general del método y la aplicación de las rutinas del proyecto. El código fuente detallado se encuentra disponible en GitHub (archivo `<demos.c>` en la sección `DEMO_GPIO`).

La operación del aparato puede observarse siguiendo el enlace: <https://youtu.be/kU1xPHZdBA0>

#### 5.4.1. Algoritmo simplificado para validación de operación ante impacto:

La idea es construir una tabla de este tipo:

No. de Impacto	Aceleración [ $g = 9.8 \frac{m}{s^2}$ ]	Condición después del impacto
1	400 $g$	Pasa
2	425 $g$	Pasa
...	...	...
n	402 $g$	Falla

Para cada impacto, se ejecutan los siguientes pasos:

1.  $a_{\max} \leftarrow 0$  (aceleración máxima)
2.  $t \leftarrow 0$  (tiempo)
3. Cambiar el estado del disparo (cambia dirección del pistón)
4. Mientras  $t < t_{\max}$  (Nota:  $t_{\max}$  debe permitir la extensión total del pistón)
  - a  $\leftarrow$  Leer ADC\_via\_SPI
  - Si ( $a > a_{\max}$ )
    - $a_{\max} \leftarrow a$  (retiene el valor máximo de la aceleración)
5. Imprimir en pantalla y almacenar en archivo el valor de la aceleración  $a_{\max}$ .
6. Establecer comunicación via SPI con el dispositivo bajo prueba para establecer condición de Falla por impacto o funcionamiento.
7. Ir a 1) en modo automático o esperar a un disparo de usuario en modo manual.

#### **5.4.2. Características del proyecto aplicables a la solución de problemas industriales:**

La elaboración de sistemas industriales se facilita enormemente cuando se dispone de las siguientes características:

- a) Disponibilidad de un puerto de datos de uso general, con pines fácilmente configurables como entradas o salidas.
- b) Funciones de medida de tiempo al nivel de milisegundos o microsegundos.
- c) Almacenamiento temporal rápido en RAM (usualmente en arreglos o tablas) de datos capturados.
- d) Almacenamiento masivo y permanente en memoria tipo FLASH.
- e) Conexión a redes de área local (LAN) o Internet para comunicación con otros dispositivos, servidores de archivos o para monitoreo.
- f) Funciones gráficas para despliegue de la información o de resultados, generación de histogramas, presentación de alarmas y otros.
- g) Soporte, mediante circuitos integrados de fácil adquisición, de puertos seriales a nivel TTL (Transistor-Transistor Logic), RS-232 (Telecommunications Industry Association, TIA-232-F), RS-422 (TIA/EIA-422) y RS-485 (ANSI/TIA/EIA-485).

#### **5.4.3. Otras aplicaciones:**

Con pocas modificaciones al circuito y al código, pueden implementarse instrumentos virtuales como voltímetros, amperímetros y medidores de resistencia. La adición de transductores permite el monitoreo de variables físicas como presión, temperatura, densidad, color, intensidad del sonido, con capacidades de salida o control. El soporte de gráficas 2D y 3D del sistema permite los refinamientos que se estimen convenientes. Además de la flexibilidad y del bajo costo, es posible utilizar monitores en desuso (comunes en las empresas y con un alto impacto ambiental al ser desechados) como pantallas dedicadas para instrumentación.

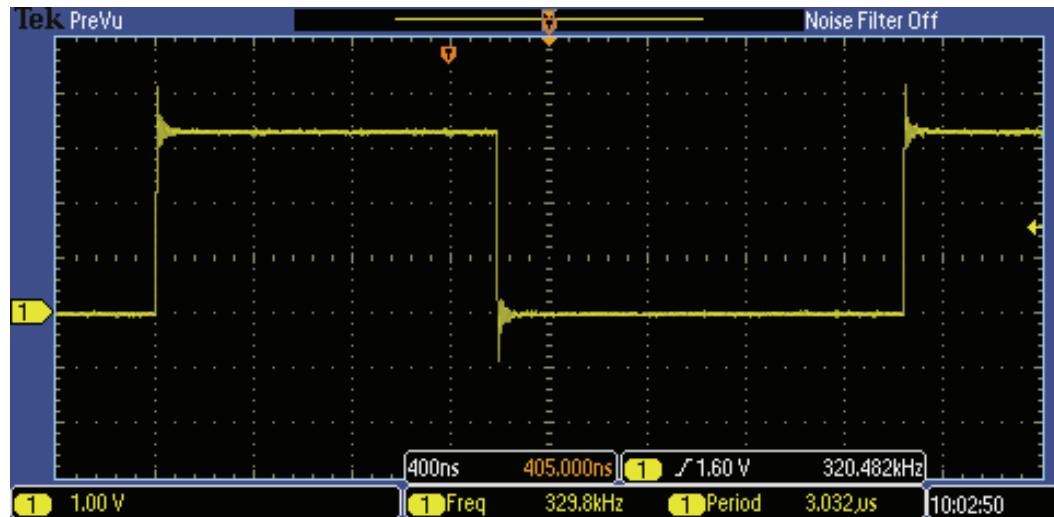
#### **5.4.4. Desempeño del GPIO:**

Comprobada la aplicabilidad de la librería del proyecto `<lb_gpio.c>` para el control de dispositivos electrónicos, surge la pregunta: ¿qué tal es su desempeño con relación a sistemas de amplia difusión como Python y su librería asociada `RPi.GPIO`?

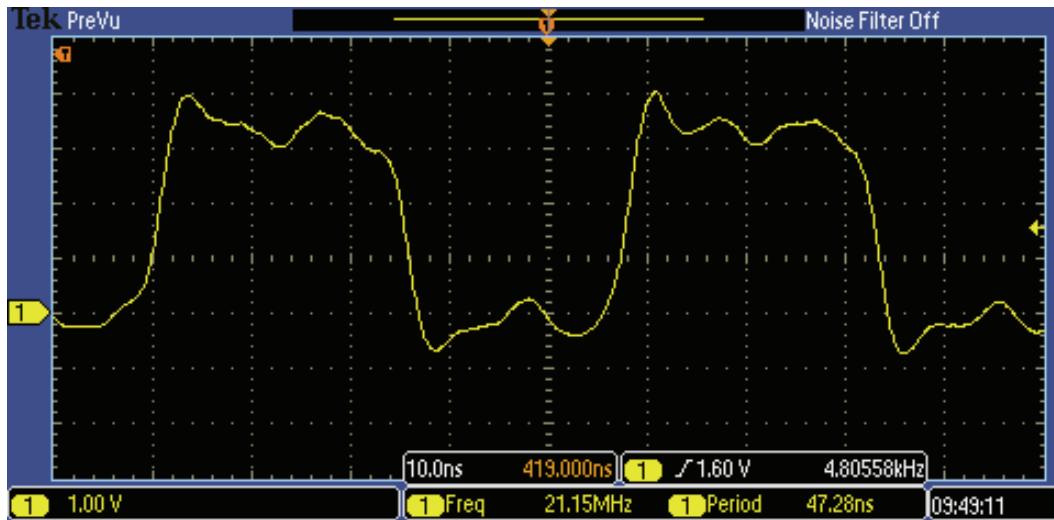
Para resolver esta duda se escribieron sendos programas bajo cada uno de estos lenguajes, ejecutados sobre el mismo sistema físico (el cual fue una tarjeta Raspberry Pi 3 B+), sin efectuar cambios a su configuración. Los programas, como se puede ver, son semejantes.

Librería Rpi.GPIO (Python)	Librería <code>&lt;lb_gpio.c&gt;</code> (Lenguaje C)
<b>Programa de Prueba:</b>  <pre>import RPi.GPIO as GPIO GPIO.setwarnings(False) GPIO.setmode(GPIO.BCM) GPIO.setup(22,GPIO.OUT,initial=GPIO.LOW) while True:     GPIO.output(22, GPIO.HIGH)     GPIO.output(22, GPIO.LOW)</pre>	<b>Programa de Prueba:</b>  <pre>#include &lt;lb_gpio.c&gt; void main(void) {     lb_gp_gpio_setup_pin(22,         GPIO_OUTPUT);     while (1)     {         lb_gp_gpio_wr(22, GPIO_HIGH);         lb_gp_gpio_wr(22, GPIO_LOW);     } }</pre>
<b>Desempeño:</b>  Señal cuadrada de 329.8 KHz	<b>Desempeño:</b>  Señal cuadrada de 21.15 MHz

Haciendo uso de la librería del proyecto, `<lb_gpio.c>`, se obtuvo un factor de desempeño  $k = \frac{21,150 \text{ KHz}}{329.8 \text{ KHz}} = 64.13$  para procesos de salida en comparación con la unidad de uso general `RPi.GPIO` de Python. El desempeño del GPIO medido en el sistema Raspberry Pi 3B+ es consistente con el determinado por HALFACREE (2018) [18].



**Figura 25:** Señal cuadrada generada haciendo uso de la librería `RPi.GPIO`. Se alcanzó una frecuencia de 329.8 KHz bajo un sistema Raspberry Pi 3 B+. [Recurso propio].



**Figura 26:** Señal cuadrada generada haciendo uso de la librería `<lb_gpio.c>`. Se alcanzó una frecuencia de 21.15 MHz bajo un sistema Raspberry Pi 3 B+. [Recurso propio].

Estos resultados indican la viabilidad del sistema propuesto para el desarrollo de aplicaciones industriales (por ejemplo, el Internet de las Cosas), como herramienta educativa, como recurso para experimentación, o para la elaboración rápida de prototipos.

#### 5.4.5. Problemas encontrados:

Una anomalía encontrada durante la operación fue el paro súbito tras transcurrir más de una hora de operación, en forma reproducible. La investigación de la falla determinó que el error ocurre por el regreso a cero de la variable interna de tiempo “ticks” consultada por la función `clock()` definida en `<time.h>`.

```
void lb_ti_delay_ms(UINT32_T delay_ms)
{
    clock_t time_a, time_b, time;
    time_a=clock();
    time_b=time_a+1000*delay_ms;
    do
    {
        time=clock();
    } while (time<time_b);
}
```

Efectivamente, el tamaño del tipo de datos `clock_t` depende de la plataforma. En las distribuciones en las cuales se escribió y probó esta función, este tipo de datos (y, por consiguiente, la variable “tick”)<sup>4</sup> son de 64 bits. En el sistema Raspberry Pi del sistema de prueba, el sistema operativo es versión `arm7l`, bajo el cual éste tipos de datos son de 32 bits.

Puede determinarse el tiempo de desbordamiento de la variable de tiempo:

$$\frac{2^{32} \text{ tictacs}}{10^6 \frac{\text{tictacs}}{\text{segundo}}} = 4,294.96 \text{ segundos} = 71 \text{ minutos}, 34 \text{ segundos}$$

En las versiones de 64 bits, el tiempo de desbordamiento es:

$$\frac{2^{64} \text{ tictacs}}{10^6 \frac{\text{tictacs}}{\text{segundo}}} = 584,452 \text{ años}$$

---

<sup>4</sup> Un “tick”, en el contexto de la electrónica ocurre cada vez que un contador asociado a un reloj digital se incrementa. El término es acuñado por analogía al sonido que produce un reloj mecánico. En español se utilizan las palabras “tictacs”, o “cuentas”.

La solución definida para evitar el error, incluso en los sistemas de 32 bits, fue la modificación de la función para que detecte si el contador ha disminuido respecto a una estampa inicial de tiempo. Si este fuera el caso, se determina (en una variable de 64 bits) el tiempo realmente transcurrido. La única restricción es que esta función debe llamarse al menos una vez cada 71 minutos, pero puede operar indefinidamente.

Una consecuencia, sin embargo, de corregir esta limitación, es que la función modificada involucra un número superior de operaciones que la inicial, y por consiguiente toma un poco más de tiempo de ejecución, lo cual reduce la resolución que de la misma. Por ejemplo, en sistemas Raspberry Pi 3 B+, pueden obtenerse resoluciones típicas de  $10 \mu\text{s}$  para la función modificada y de  $5 \mu\text{s}$  para la función escrita inicialmente (sujeta esta última, como se dijo anteriormente, a un error de desbordamiento tras transcurrir 71 minutos de la era de ejecución).

Otra forma de evitar este problema es asegurar que el sistema operativo instalado sea de 64 bits. En la actualidad se encuentra disponible una versión experimental de Ubuntu-Mate de 64 bits para procesadores ARMv8 como aquellos de los modelos Raspberry Pi B 3 y B 3+.

## 6. Tipos y Estructuras de Datos

Se presentan a continuación los principales tipos de datos definidos para el proyecto, comunes a todas las librerías.

### Tipos de datos y constantes matemáticos, vectoriales, y de álgebra lineal

Constantes booleanas:

```
#define TRUE 1
#define FALSE 0
```

Tipos de datos elementales:

```
#define REAL_T    double
#define UINT8_T    unsigned char
#define UINT16_T   unsigned short
#define UINT32_T   unsigned int
#define SINT8_T    char
#define SINT16_T   short
#define SINT32_T   long
```

Se hace uso de pseudónimos para los tipos de datos dado que los tipos de datos definidos por C pueden ser diferentes según el compilador y la arquitectura. Por ejemplo, los diversos estándares especifican que el tipo `int` tenga al menos 16 bits, sin que esto se garantice, lo cual podría causar problemas en tareas que requieran un número de bits determinado. Alternativamente, pudieron haberse utilizado los tipos de datos de tamaño fijo incorporados por el estándar C99 por la misma razón, como, por ejemplo, `int32_t`. El tipo de dato real puede definirse como `float` o como `double` según se requiera por razones de espacio (como en arreglos matriciales muy grandes) o de velocidad según el caso.

Tipo de datos para errores matemáticos:

Se define la enumeración `MATHERROR_T`, útil para que las funciones matemáticas puedan indicar condiciones de error, de invalidez de argumentos, o de resultados:

```
typedef enum { e_none=0, e_div_zero, e_too_big, e_complex, e_undefined, e_arg_smaller_one,
               e_arg_larger_one, e_arg_negative, e_arg_non_int, e_arg_non_even,
               e_arg_non_odd, e_arg_less_two, e_arg_too_big, e_arg_too_small,
               e_arg_inf_loop, e_arg_non_real, e_arg_not_var } MATHERROR_T;
```

Tipo de datos complejo:

```
typedef struct
{
    REAL_T r, i;
} COMPLEX_T;
```

También se define la constante compleja nula:

```
#define ZERO_C (COMPLEX_T){0.0, 0.0}
```

Máxima dimensión para arreglos multidimensionales:

```
#define ARRAY_MAX_DIM      5
```

Máximo número de elementos para arreglos multidimensionales:

```
#define ARRAY_MAX_N        65535
```

Los límites en los tamaños de los arreglos se establecen según el tamaño en bits de los índices del arreglo (por ejemplo, ver miembro `items` [el cual es de 16 bits sin signo] de la estructura de datos `VECTOR_SINT8_T` a continuación). Es posible redefinir el tipo de datos determinado para que soporte un número extraordinariamente grande de elementos, como podría requerirse, por ejemplo, para aplicaciones de monitoreo o “data logging”. La ruta a seguir es la siguiente:

- a) Redefinir el tipo de datos del miembro (por ejemplo, en la estructura `VECTOR_S_INT8_T` hacer `items` un `UINT32_T` en lugar de `UINT16_T`)
- b) Redefinir la constante asociada, por ejemplo, hacer `VECTOR_MAX_ITEMS 1000000` en lugar de `65535`.
- c) Si se llaman funciones que hagan uso del tipo de datos siendo redefinido, revisar en estas el tamaño de las variables índices.
- d) Efectuar pruebas de validación.

Máximo número de filas y columnas en una matriz:

```
#define MATRIX_MAX_ROWS   65535  
#define MATRIX_MAX_COLS   65535
```

Máximo número de elementos en un vector:

```
#define VECTOR_MAX_ITEMS 65535
```

Matrix de enteros de 8 bits con signo:

```
typedef struct  
{  
    SINT8_T *array;  
    UINT16_T items;  
} VECTOR_SINT8_T;
```

Vector de enteros de 8 bits con signo:

```
typedef struct
{
    SINT16_T      *array;
    UINT16_T      items;
} VECTOR_SINT16_T;
```

Vector de números reales:

```
typedef struct
{
    REAL_T        *array;
    UINT16_T      items;
} VECTOR_R_T;
```

Vector de números complejos:

```
typedef struct
{
    COMPLEX_T     *array;
    U_INT16_T     items;
} VECTOR_C_T;
```

Matriz de enteros de 8 bits con signo:

```
typedef struct
{
    SINT8_T      **array;
    UINT16_T      rows;
    UINT16_T      cols;
} MATRIX_SINT8_T;
```

Matriz de números reales:

```
typedef struct
{
    REAL_T        **array;
    UINT16_T      rows;
    UINT16_T      cols;
} MATRIX_R_T;
```

Matriz de números complejos:

```
typedef struct
{
    COMPLEX_T     **array;
    UINT16_T      rows;
    UINT16_T      cols;
} MATRIX_C_T;
```

Arreglo multidimensional (experimental):

```
typedef struct
{
    SINT8_T n;
    UINT16_T dim[5];
    REAL_T *a1;
    REAL_T **a2;
    REAL_T ***a3;
    REAL_T ****a4;
    REAL_T *****a5;
} ARRAY_R_T;
```

Punteros a funciones de una y dos variables, los cuales facilitan el paso de funciones como argumentos, lo cuál es útil o necesario, por ejemplo, para implementar métodos numéricos:

```
typedef FLOAT_T(*FUNCTION_X)(FLOAT_T, MATHERROR_T*);
typedef FLOAT_T(*FUNCTION_X_Y)(FLOAT_T, FLOAT_T, MATHERROR_T*);
```

### Tipos de datos y constantes para gráficas por computadora

Máximas dimensiones de pantalla vertical y horizontal:

```
#define PICTURE_MAX_HEIGHT 1080
#define PICTURE_MAX_WIDTH 1920
```

Estas definiciones buscan prevenir errores de desbordamiento en las variables internas utilizadas por las primitivas gráficas, y se eligieron buscando usar los tipos de datos más pequeños posibles y a mantener amigabilidad hacia el hardware de menor desempeño. La extensión a mayores resoluciones como 4K no es una tarea difícil, pues la validez de los algoritmos se mantiene, requiriendo únicamente cambios en los tipos de datos. La ruta a seguir consiste en validar cada primitiva bajo la resolución deseada y adecuar si fuera necesario las variables del caso, reemplazando, por ejemplo, `SINT8` por `SINT16`, o `SINT16` por `SINT32`).

Número de bits por canal de color para la estructura de datos `PICTURE_T`:

```
#define N_BITS_R 8 /* Number of bits for red channel */
#define N_BITS_G 8 /* Number of bits for green channel */
#define N_BITS_B 8 /* Number of bits for blue channel */
#define N_BITS_A 8 /* Number of bits for alpha channel */
```

La estructura `PICTURE_T` soporta un número de bits entre uno y ocho. Se hace esta previsión por dos razones: en primer lugar, durante el desarrollo de primitivas es útil utilizar un número menor de bits por canal, a fin de verificar la adecuada cuantización de la imagen. En segundo lugar, permite crear imágenes (o series de imágenes) que utilicen menos memoria en sistemas de menores recursos.

Como últimamente el despliegue de las imágenes (al menos en equipos de escritorio) se hace típicamente a través de una interfaz de 8 bits por canal, se definen factores de conversión asociados: (se muestran únicamente las definiciones asociadas al canal rojo. Las del verde, azul, y del canal auxiliar son similares)

```
#if N_BITS_R == 1
#define MAX_R 1
#define FACTOR_N_TO_8_R 255
#endif

#if N_BITS_R == 2
#define MAX_R 3
#define FACTOR_N_TO_8_R (255/3)
#endif

#if N_BITS_R == 3
#define MAX_R 7
#define FACTOR_N_TO_8_R (255/7)
#endif

#if N_BITS_R == 4
#define MAX_R 15
#define FACTOR_N_TO_8_R (255/15)
#endif

#if N_BITS_R == 5
#define MAX_R 31
#define FACTOR_N_TO_8_R (255/31)
#endif

#if N_BITS_R == 6
#define MAX_R 63
#define FACTOR_N_TO_8_R (255/63)
#endif

#if N_BITS_R == 7
#define MAX_R 127
#define FACTOR_N_TO_8_R (255/127)
#endif

#if N_BITS_R == 8
#define MAX_R 255
#define FACTOR_N_TO_8_R 1
#endif
```

Un error frecuente, incluso en ámbitos técnicos, consiste en asumir que la conversión de un valor codificado en cierto número de bits a otro, puede realizarse simplemente descartando los bits más bajos (para el cambio de mayor a menor número de bits) o desplazando bits y llenando los bits inferiores con ceros (para el cambio de menor a mayor número de bits). Dicha aproximación solamente es válida cuando el número de bits es alto.

Por ejemplo, el color rojo puro en un esquema con tres bits por canal se representa en binario como  $[rojo_3] = 0b111$ . Si se convierte a 8 bits, el impulso inicial es hacer  $[rojos] = 0b11100000$ . El resultado, sin embargo, difiere en  $0b00011111 / 0b11111111 = 12.16\%$ , lo cual es perceptible.

Un mejor factor de conversión, para el ejemplo anterior es (255/31) (alrededor de 8.22), cuyo resultado es `0b11111110` (254 en decimal) con aritmética truncada o `0b11111111` (255 en decimal) con redondeo al entero más cercano.

### Constantes de colores:

Se definen por conveniencia constantes de colores, las cuales son basadas en aquellas de X11, con algunas diferencias. En este proyecto se hace uso extensivo del modo de color de 12 bits, el cual es interesante porque permite representar el espacio de cada canal de color con un solo dígito hexadecimal, lo cual francamente simplifica la escritura de aplicaciones o el uso de máscaras. Además, el espacio total de colores es aún 4096, muchos más de los que es posible identificar para casi todos los usuarios en forma absoluta (en forma relativa sí es posible establecer diferencias, por ejemplo, entre 12 y 13 bits, si las muestras están lado a lado).

```
/* 32 most basic color definitions for the 12-bit color mode
#define PIXEL_BLACK      (PIXEL_T) { 0x0, 0x0, 0x0, 0xF } */

/* Transparency Attributes */
#define COLOR_SOLID          0xF000
#define COLOR_SEMI_SOLID       0x7000
#define COLOR_TRANSPARENT     0xFFFF

/* Common colors */
#define COLOR_RED            0x00F
#define COLOR_DARKRED         0x008
#define COLOR_ORANGE          0x0AF
#define COLOR_DARKORANGE      0x08F
#define COLOR_GOLD             0x0DF
#define COLOR_LIME             0x0F0
#define COLOR_GREEN            0x080
#define COLOR_OLIVE             0x088
#define COLOR_LIGHTGREEN        0x8E8
#define COLOR_LIMEGREEN         0x3C3
#define COLOR_DARKGREEN         0x060
#define COLOR_SPRINGGREEN       0x7F0
#define COLOR_BLUE              0xF00
#define COLOR_LIGHTBLUE         0xEDA
#define COLOR_SKYBLUE            0xEC8
#define COLOR_DARKBLUE           0x800

#define COLOR_YELLOW           0x0FF
#define COLOR_LIGHTYELLOW        0xDFF
#define COLOR_CYAN              0xFF0
#define COLOR_TEAL                0x880
#define COLOR_LIGHTCYAN          0xFFD
#define COLOR_MAGENTA             0xF0F
#define COLOR_PURPLE              0x808

/* Other common colors */
#define COLOR_PINK               0xCB
#define COLOR_LIGHTPINK            0xBB
```

```

#define COLOR_BROWN          0x22A
#define COLOR_KHAKI           0x8EE
#define COLOR_BEIGE           0xDEE
#define COLOR_VIOLET          0xE8E
#define COLOR_BLUEVIOLET      0xD38

/* Gray scale */
#define COLOR_BLACK            0x000
#define COLOR_DIMGRAY          0x666
#define COLOR_GRAY              0x888
#define COLOR_DARKGRAY          0xAAA
#define COLOR_SILVER             0xBBB
#define COLOR_LIGHTGRAY         0xCCC
#define COLOR_GAINSBORO        0xDDD
#define COLOR_WHITESMOKE       0xEEE
#define COLOR_WHITE              0xFFFF

/* Many other "designer's" colors */
#define COLOR_HOTPINK           0xB6F
#define COLOR_DEEPPINK          0x91F
#define COLOR_PALEVIOLETRED     0x97D
#define COLOR_MEDIUMVIOLETRED    0x81C
#define COLOR_LIGHTSALMON        0x79F
#define COLOR_SALMON              0x78F
#define COLOR_DARKSALMON         0x79E
#define COLOR_LIGHTCORAL         0x88E
#define COLOR_INDIANRED          0x55C
#define COLOR_CRIMSON             0x41D
#define COLOR_ORANGERED          0x04F
#define COLOR_TOMATO              0x46F
#define COLOR_CORAL                0x57F
#define COLOR_LEMONCHIFFON        0xCFF
#define COLOR_LIGHTGOLDENRODYELLOW 0xCFF
#define COLOR_MOCCASIN           0xBDF
#define COLOR_PALEGOLDENROD       0xAEE
#define COLOR_DARKKHAKI           0x6BB
#define COLOR_CORN SILK            0xDFF
#define COLOR_BLANCHEDALMOND      0xCEF
#define COLOR_BISQUE               0CDF
#define COLOR_NAVAJOWHITE         0xADF
#define COLOR_WHEAT                 0xBDE
#define COLOR_BURLYWOOD           0x8BD
#define COLOR_TAN                  0x8BC
#define COLOR_ROSYBROWN           0x88B
#define COLOR_SANDYBROWN          0x6AE
#define COLOR_GOLDENROD            0x2AD
#define COLOR_DARKGOLDENROD        0x18B
#define COLOR_PERU                  0x48C
#define COLOR_CHOCOLATE            0x26C
#define COLOR_SADDLEBROWN          0x148
#define COLOR_SIENNA                0x359
#define COLOR_DARKOLIVEGREEN       0x365
#define COLOR_OLIVEDRAB             0x286
#define COLOR_YELLOWGREEN          0x3C9
#define COLOR_LAWNGREEN             0x0F7
#define COLOR_CHARTREUSE            0x0F7
#define COLOR_GREENYELLOW          0x3FA
#define COLOR_MEDIUMSPRINGGREEN    0x9F0
#define COLOR_PALEGREEN              0x9F9
#define COLOR_DARKSEAGREEN          0x8B8
#define COLOR_MEDIUMSEAGREEN        0x7B4

```

```

#define COLOR_SEAGREEN          0x583
#define COLOR_FORESTGREEN       0x282
#define COLOR_MEDIUMAQUAMARINE   0xAC6
#define COLOR_PALETURQUOISE     0xEEA
#define COLOR_AQUAMARINE        0xCF7
#define COLOR_TURQUOISE         0xCD4
#define COLOR_MEDIUMTURQUOISE    0xCC4
#define COLOR_DARKTURQUOISE      0xCC0
#define COLOR_LIGHTSEAGREEN      0xAA2
#define COLOR_CADETBLUE          0x996
#define COLOR_LIGHTSTEELBLUE     0xDCA
#define COLOR_LIGHTSKYBLUE        0xFC8
#define COLOR_DEEPSKYBLUE        0xFB0
#define COLOR_DODGERBLUE         0xF82
#define COLOR_CORNFLOWERBLUE     0xE96
#define COLOR_STEELBLUE           0xB84
#define COLOR_ROYALBLUE          0x810
#define COLOR_MEDIUMBLUE          0xC00
#define COLOR_MIDNIGHTBLUE        0x711
#define COLOR_LAVENDER            0xFEE
#define COLOR_THISTLE              0xDBD
#define COLOR_PLUM                 0xD9D
#define COLOR_ORCHID               0xD7D
#define COLOR_MEDIUMORCHID         0xC5B
#define COLOR_MEDIUMPURPLE        0xD79
#define COLOR_DARKORCHID           0xC39
#define COLOR_INDIGO                0x804
#define COLOR_DARKSLATEBLUE        0x844
#define COLOR_REBECCAPURPLE        0x936
#define COLOR_SLATEBLUE             0xC56
#define COLOR_MEDIUMSLATEBLUE      0xE67
#define COLOR_HONEYDEW              0xEFE
#define COLOR_AZURE                  0xFFE
#define COLOR_SEASHELL                0xEFF
#define COLOR_FLORALWHITE           0xEFF
#define COLOR_IVORY                  0xEFF
#define COLOR_ANTIQUEWHITE           0xDEF
#define COLOR_MISTYROSE                0xDDF
#define COLOR_LIGHTSLATEGRAY        0x987
#define COLOR_SLATEGRAY                0x887
#define COLOR_DARKSLATEGRAY          0x553

```

```

/* Redundant colors */
#define COLOR_MAROON                0x008
#define COLOR_NAVY                  0x800
#define COLOR_AQUA                  0xFF0
#define COLOR_FUCHSIA                0xF0F
#define COLOR_DARKMAGENTA             0x808
#define COLOR_DARKCYAN                0x880
#define COLOR_DARKVIOLET                0xC09
#define COLOR_GHOSTWHITE                0xFFF
#define COLOR_BROWN                  0x22A
#define COLOR_POWDERBLUE                0xEDA
#define COLOR_SNOW                    0xFFFF
#define COLOR_OLDLACE                  0xEEF
#define COLOR_PEACHPUFF                0xBDF
#define COLOR_LINEN                  0xEEF
#define COLOR_LAVENDERBLUSH             0xEEF
#define COLOR_MINTCREAM                0FFE
#define COLOR_ALICEBLUE                0FFE
#define COLOR_PAPAYAWHIP                0xDEF

```

**Máximo número de puntos en una línea**, representada como un vector de pares ordenados enteros o reales (existen versiones independientes):

```
#define LINE_MAX_ITEMS 65535
```

**Modos de copia**, los cuales definen la operación a desempeñar entre el pixel a escribir y aquel existente en el destino. Se soportan los modos de copia rápida, operaciones binarias digitales, copia aditiva, subtractiva, multiplicativa, de cociente, y los operadores de combinación con difusión Porter-Duff [13]:

```
typedef enum { COPYMODE_FRAMEBUFFER=0,
    COPYMODE_COPY,
    COPYMODE_AND,
    COPYMODE_OR,
    COPYMODE_XOR,
    COPYMODE_NOT_SRC,
    COPYMODE_NOT_DST,
    COPYMODE_ADD,
    COPYMODE_SRC_MINUS_DST,
    COPYMODE_DST_MINUS_SRC,
    COPYMODE_AVG,
    COPYMODE_BLEND,
    COPYMODE_MULTIPLY,
    COPYMODE_SRC_DIV_DST,
    COPYMODE_DST_DIV_SRC,
    COPYMODE_PORTRDUFF_CLEAR_DST,
    COPYMODE_PORTRDUFF_COPY_SRC,
    COPYMODE_PORTRDUFF_LEAVE_DST,
    COPYMODE_PORTRDUFF_SRC_OVER_DST,
    COPYMODE_PORTRDUFF_DST_OVER_SRC,
    COPYMODE_PORTRDUFF_SRC_IN_DST,
    COPYMODE_PORTRDUFF_DST_IN_SRC,
    COPYMODE_PORTRDUFF_SRC_OUT_DST,
    COPYMODE_PORTRDUFF_DST_OUT_SRC,
    COPYMODE_PORTRDUFF_SRC_ATOP_DST,
    COPYMODE_PORTRDUFF_DST_ATOP_SRC,
    COPYMODE_PORTRDUFF_XOR } COPYMODE_T;
```

Las siguientes constantes y macros se utilizan para definir opciones para el despliegue de una imagen (**PICTURE\_T**) en pantalla. Por ejemplo, se soporta el cambio de escala, la definición de un color de fondo, y otros. Su uso se muestra mejor a través de los ejemplos del proyecto. Las máscaras se declaran en sistema binario porque así se evitan conversiones, se identifican fácilmente los campos, y en la práctica se evitan errores.

```
#define RENDERMODE_T U_INT32_T

/* This section deals with the attributes passed through the "copymode" parameter. They're
   listed from lower to higher order.
   For example: value = (value >> THAT_VALUE_SHIFT ) && THAT_VALUE_MASK */

#define RENDERMODE_PIXELBG_R_MASK      0b00000000000000000000000000000001111
#define RENDERMODE_PIXELBG_G_MASK      0b000000000000000000000000000000011110000
```

```

#define RENDERMODE_PIXELBG_B_MASK      0b00000000000000000000000011100000000
#define RENDERMODE_PIXELMODE_MASK       0b0000000000000000000000001110000000000
#define RENDERMODE_SCALE_X_MASK        0b00000000001111100000000000000000
#define RENDERMODE_SCALE_Y_MASK        0b00001111100000000000000000000000

#define RENDERMODE_PIXELMODE_0          0b0000000000000000000000000000000000000000
#define RENDERMODE_PIXELMODE_1          0b000000000000000000000000000000001000000000000
#define RENDERMODE_PIXELMODE_2          0b000000000000000000000000000000001000000000000
#define RENDERMODE_PIXELMODE_3          0b000000000000000000000000000000001100000000000

#define RENDERMODE_PIXELMODE_SHIFT     12
#define RENDERMODE_PIXELBG_R_SHIFT     0
#define RENDERMODE_PIXELBG_G_SHIFT     4
#define RENDERMODE_PIXELBG_B_SHIFT     8
#define RENDERMODE_SCALE_X_SHIFT       16
#define RENDERMODE_SCALE_Y_SHIFT       22

/* These macros help the user setting an arbitrary "pixel size" */

#define RENDERMODE_SCALE_X(X) (((X-1)<<RENDERMODE_SCALE_X_SHIFT) &
RENDERMODE_SCALE_X_MASK)

#define RENDERMODE_SCALE_Y(X) (((X-1)<<RENDERMODE_SCALE_Y_SHIFT) &
RENDERMODE_SCALE_Y_MASK)

#define RENDERMODE_BGCOLOR(X) (X & (RENDERMODE_PIXELBG_R_MASK |
RENDERMODE_PIXELBG_G_MASK | RENDERMODE_PIXELBG_B_MASK) )

```

La siguiente enumeración define los posibles **modos de trazo de una línea**: puntos, puntos con filtro anti-escalonamiento, línea sólida o línea con filtro anti-escalonamiento)

```

typedef enum
{
    LINEMODE_DOTS_SOLID,
    LINEMODE_DOTS_FILTERED,
    LINEMODE_SOLID, /* Draws solid pixels */
    LINEMODE_FILTERED /* Uses anti-aliasing */
} LINEMODE_T;

```

**Punto en dos dimensiones** en variable signada entera de 16 bits. La bandera puede usarse como se deseé, por ejemplo, para indicar si el punto debe o no ser presentado en pantalla:

```

typedef struct
{
    S_INT16_T x;
    S_INT16_T y;
    S_INT8_T flag;
} POINT_2D_SINT16_T;

```

Punto en dos dimensiones en variable real:

```
typedef struct
{
    REAL_T x;
    REAL_T y;
    S_INT8_T flag;
} POINT_2D_REAL_T;
```

Punto en tres dimensiones en variable real:

```
typedef struct
{
    REAL_T x;
    REAL_T y;
    REAL_T z;
    S_INT8_T flag;
} POINT_3D_REAL_T;
```

Línea en dos dimensiones compuesta por enteros de 16 bits signados:

```
typedef struct
{
    POINT_2D_SINT16_T *array;
    UINT16_T items;
} LINE_2D_SINT16_T;
```

Línea en dos dimensiones compuesta por reales:

```
typedef struct
{
    POINT_2D_REAL_T *array;
    UINT16_T items;
} LINE_2D_REAL_T;
```

Línea en tres dimensiones compuesta por reales:

```
typedef struct
{
    POINT_3D_REAL_T *array;
    UINT16_T items;
} LINE_3D_REAL_T;
```

Matriz de puntos bidimensionales de variable real:

```
typedef struct
{
    POINT_2D_REAL_T **array;
    UINT16_T rows;
    UINT16_T cols;
} MATRIX_POINT_2D_REAL_T;
```

**Matriz de puntos tridimensionales en variable real:**

```
typedef struct
{
    POINT_3D_REAL_T **array;
    UINT16_T rows;
    UINT16_T cols;
} MATRIX_POINT_3D_REAL_T;
```

**Estructura de Ventana o “Viewport”.** Define la transformación entre coordenadas reales cartesianas en el plano y coordenadas de pantalla:

```
typedef struct
{
    SINT16_T      xp_min;
    SINT16_T      yp_min;
    SINT16_T      xp_max;
    SINT16_T      yp_max;
    REAL_T        xr_min;
    REAL_T        xr_max;
    REAL_T        yr_min;
    REAL_T        yr_max;
} VIEWPORT_2D_T;
```

**Estructura de Ventana o “Viewport” tridimensional.** Define los parámetros de posición de la cámara y de la proyección de la imagen:

```
typedef struct
{
    SINT16_T      xp_min;
    SINT16_T      yp_min;
    SINT16_T      xp_max;
    SINT16_T      yp_max;
    REAL_T        scale;      /* Zoom */
    REAL_T        cam_d;      /* Stereoscopic */
    REAL_T        cam_h;      /* Depth */
    POINT_3D_REAL_T cam;      /* Camera's location */
} VIEWPORT_3D_T;
```

**Elemento de imagen o “pixel”.** A cada canal se le asigna un número arbitrario de bits.

```
typedef struct
{
    UINT8_T r:N_BITS_R;
    UINT8_T g:N_BITS_G;
    UINT8_T b:N_BITS_B;
    UINT8_T a:N_BITS_A;
} PIXEL_T;
```

**Estructura de datos de Imagen “PICTURE\_T”.** Sobre el buffer apuntado por `pic` se efectúan las operaciones gráficas, en RAM, en forma independiente del hardware.

```
typedef struct
{
    PIXEL_T **pic;
    UINT16_T w;
    UINT16_T h;
} PICTURE_T;
```

**Estructura de datos de Pantalla “SCREEN\_T”.** Es similar a `PICTURE_T`, pero se define en forma que permita la copia directa al dispositivo de video, por lo cual es dependiente del método de despliegue que se elija, por ejemplo, copia a una textura de Simple DirectMedia Layer.

```
typedef struct
{
    char *dat;
    UINT16_T w;
    UINT16_T h;
} SCREEN_T;
```

**Estructura de datos para fuentes de tipo fijo.** Comprende parámetros como el número de columnas, de filas, la escala, la orientación o ángulo, banderas de despliegue del color principal y de fondo, y los valores de los mismos.

```
typedef struct
{
    UINT8_T cols;
    UINT8_T rows;
    UINT8_T range_a;
    UINT8_T range_b;
    unsigned char* data;
    UINT8_T scale_x;
    UINT8_T scale_y;
    UINT8_T gap_x;
    UINT8_T gap_y;
    UINT8_T max_x;
    SINT16_T angle;
    UINT8_T flag_fg;
    UINT8_T flag_bg;
    PIXEL_T color_fg;
    PIXEL_T color_bg;
} FONT_T;
```

**Estructura de datos de cada elemento de texto para consola virtual.** Las consolas virtuales permiten presentar almacenar texto a través de un buffer rotativo tipo FIFO (First Input, First Output). Esta estructura es muy simple, y consta solamente de un carácter, y de sus atributos, como son el color de despliegue y el color de fondo.

```

typedef struct
{
    char c;
    PIXEL_T color_fg;
    PIXEL_T color_bg;
} CONSOLE_CHAR_T;

```

### Estructura de Consola Virtual:

```

typedef struct
{
    /* 2D circular buffer */
    CONSOLE_CHAR_T **screen;           /* 2D circular buffer for visualization, scrollable */
    SINT16_T MAX_ROWS;                /* Maximum (total) number of rows stored in memory */
    SINT16_T MAX_ROWS_V;              /* Visible number of rows*/
    SINT16_T MAX_COLS;                /* SINT16_T n_row;          Number of "rows" which have been
                                         received*/
    SINT16_T a_row;                   /* Index of the oldest row that was written */
    SINT16_T b_row;                   /* Index of the row of the next cell to be written */
    SINT16_T w_col;                  /* Index of the column of the next cell to be written */
    SINT16_T s_row;                  /* Index of the row of the next cell to be "seen"(cursor) */
    SINT16_T s_col;                  /* Index of the column of the next cell to be "seen"
                                         (cursor) */
    SINT16_T n_back;                 /* Number of rows "backwards" counted from the next cell
                                         to be written */
    SINT16_T n_back_adj;             /* Adjusted number of rows "backwards", to add hysteresis */
    PIXEL_T color_fg;                /* Current foreground color */
    PIXEL_T color_bg;                /* Current background color */
} CONSOLE_T;

```

Las siguientes constantes se utilizan para el paso de opciones a las rutinas de trazo de gráficas XY, por ejemplo, la inclusión o no de los ejes coordenados, el trazo de flechas, trazo de retícula, de etiquetas, o escala logarítmica.

```

#define AXIS_DRAW_X          0b0000000000000001
#define AXIS_DRAW_X_ARROWS   0b0000000000000010
#define AXIS_DRAW_X_GRID     0b00000000000000100
#define AXIS_DRAW_X_GRID_LOG 0b000000000000001000
#define AXIS_DRAW_X_LABEL    0b0000000000000010000
#define AXIS_DRAW_Y          0b00000000000000100000
#define AXIS_DRAW_Y_ARROWS   0b000000000000001000000
#define AXIS_DRAW_Y_GRID     0b0000000000000010000000
#define AXIS_DRAW_Y_GRID_LOG 0b00000000000000100000000
#define AXIS_DRAW_Z_GRID     0b00000000000000100000000

#define AXIS_DRAW_COLORVALUES_X_1 0b0000001000000000 /* Degrade */
#define AXIS_DRAW_COLORVALUES_X_2 0b0000100000000000 /* Color code */
#define AXIS_DRAW_COLORVALUES_Y_1 0b0001000000000000 /* Degrade */
#define AXIS_DRAW_COLORVALUES_Y_2 0b0010000000000000 /* Color code */

#define AXIS_POLAR_R_GRID    0b000000001
#define AXIS_POLAR_T_GRID    0b000000010

```

**Variables globales:** Framebuffer (compatible con SDL):

```
SCREEN_T ty_screen;
```

**Bandera de estado:** para comprobar si SDL ha sido inicializado

```
SINT8_T ty	SDL_initialized;
```

## Constantes ASCII

Se definen las siguientes constantes correspondientes al código ASCII simplemente por conveniencia, pues es de uso frecuente en sistemas embebidos.

```
#define ASCII_NUL      0x00    /* NULL */
#define ASCII_SOH       0x01    /* Start of heading */
#define ASCII_STX       0x02    /* Start of text */
#define ASCII_ETX       0x03    /* End of text */
#define ASCII_EOT       0x04    /* End of transmission */
#define ASCII_ENQ       0x05    /* Enquiry */
#define ASCII_ACK       0x06    /* Acknowledge */
#define ASCII_BEL       0x07    /* Bell */
#define ASCII_BS        0x08    /* Backspace */
#define ASCII_TAB        0x09    /* Horizontal tab */
#define ASCII_LF        0x0A    /* New line feed, new line */
#define ASCII_VT        0x0B    /* Vertical tab */
#define ASCII_FF        0x0C    /* New page form feed, new page */
#define ASCII_CR        0x0D    /* Carriage return */
#define ASCII_SO        0x0E    /* Shift out */
#define ASCII_SI        0x0F    /* Shift in */

#define ASCII_DLE        0x10    /* Data link escape */
#define ASCII_DC1        0x11    /* Device control 1 */
#define ASCII_DC2        0x12    /* Device control 2 */
#define ASCII_DC3        0x13    /* Device control 3 */
#define ASCII_DC4        0x14    /* Device control 4 */
#define ASCII_NAK        0x15    /* Negative acknowledge */
#define ASCII_SYN        0x16    /* Synchronous idle */
#define ASCII_ETB        0x17    /* End of transmission block */
#define ASCII_CAN        0x18    /* Cancel */
#define ASCII_EM         0x19    /* End of Medium */
#define ASCII_SUB        0x1A    /* Substitute */
#define ASCII_ESC        0x1B    /* Escape */
#define ASCII_FS         0x1C    /* File separator */
#define ASCII_GS         0x1D    /* Group separator */
#define ASCII_RS         0x1E    /* Record separator */
#define ASCII_US         0x1F    /* Unit separator */

#define ASCII_SPACE      0x20    /*   */
#define ASCII_EXCLAMATION 0x21    /* ! */
#define ASCII_QUOTATION   0x22    /* " */
#define ASCII_NUMBER      0x23    /* # */
#define ASCII_DOLLAR      0x24    /* $ */
#define ASCII_PERCENT     0x25    /* % */
#define ASCII_AMPERSAND   0x26    /* & */
```

```

#define ASCII_APOSTOPHE      0x27      /* ' */
#define ASCII_PARENTHESSES_L  0x28      /* ( */
#define ASCII_PARENTHESSES_R  0x29      /* ) */
#define ASCII_ASTERISK        0x2A      /* * */
#define ASCII_PLUS             0x2B      /* + */
#define ASCII_COMMA            0x2C      /* , */
#define ASCII_MINUS             0x2D      /* - */
#define ASCII_POINT             0x2E      /* . Formal name is full stop */
#define ASCII_SLASH             0x2F      /* / */

#define ASCII_0                 0x30
#define ASCII_1                 0x31
#define ASCII_2                 0x32
#define ASCII_3                 0x33
#define ASCII_4                 0x34
#define ASCII_5                 0x35
#define ASCII_6                 0x36
#define ASCII_7                 0x37
#define ASCII_8                 0x38
#define ASCII_9                 0x39
#define ASCII_COLON              0x3A      /* : */
#define ASCII_SEMICOLON          0x3B      /* ; */
#define ASCII_LESS_THAN           0x3C      /* < */
#define ASCII_EQUALS              0x3D      /* = */
#define ASCII_GREATER_THAN         0x3E      /* > */
#define ASCII_QUESTION             0x3F      /* ? */

#define ASCII_AT                  0x40      /* @ */
#define ASCII_A                  0x41
#define ASCII_B                  0x42
#define ASCII_C                  0x43
#define ASCII_D                  0x44
#define ASCII_E                  0x45
#define ASCII_F                  0x46
#define ASCII_G                  0x47
#define ASCII_H                  0x48
#define ASCII_I                  0x49
#define ASCII_J                  0x4A
#define ASCII_K                  0x4B
#define ASCII_L                  0x4C
#define ASCII_M                  0x4D
#define ASCII_N                  0x4E
#define ASCII_O                  0x4F

#define ASCII_P                  0x50
#define ASCII_Q                  0x51
#define ASCII_R                  0x52
#define ASCII_S                  0x53
#define ASCII_T                  0x54
#define ASCII_U                  0x55
#define ASCII_V                  0x56
#define ASCII_W                  0x57
#define ASCII_X                  0x58
#define ASCII_Y                  0x59
#define ASCII_Z                  0x5A
#define ASCII_SQUARE_BRACKET_L   0x5B
#define ASCII_BACKSLASH           0x5C
#define ASCII_SQUARE_BRACKET_R   0x5D
#define ASCII_CARET               0x5E
#define ASCII_UNDERSCORE           0x5F

```

```

#define ASCII_GRAVE_ACCENT      0x60      /* ` */
#define ASCII_a                  0x61
#define ASCII_b                  0x62
#define ASCII_c                  0x63
#define ASCII_d                  0x64
#define ASCII_e                  0x65
#define ASCII_f                  0x66
#define ASCII_g                  0x67
#define ASCII_h                  0x68
#define ASCII_i                  0x69
#define ASCII_j                  0x6A
#define ASCII_k                  0x6B
#define ASCII_l                  0x6C
#define ASCII_m                  0x6D
#define ASCII_n                  0x6E
#define ASCII_o                  0x6F

#define ASCII_p                  0x70
#define ASCII_q                  0x71
#define ASCII_r                  0x72
#define ASCII_s                  0x73
#define ASCII_t                  0x74
#define ASCII_u                  0x75
#define ASCII_v                  0x76
#define ASCII_w                  0x77
#define ASCII_x                  0x78
#define ASCII_y                  0x79
#define ASCII_z                  0x7A
#define ASCII_CURLY_BRACKET_L    0x7B
#define ASCII_VERTICAL_BAR        0x7C
#define ASCII_CURLY_BRACKET_R    0x7D
#define ASCII_TILDE               0x7E
#define ASCII_DEL                 0x7F

```

## Constantes de Teclado

Las siguientes constantes se asignan haciendo uso los valores correspondientes a caracteres ASCII no imprimibles. Estos códigos son útiles para la detección de teclas especiales durante la elaboración de programas simples de consola.

```

#define PCKEY_UP                0x01
#define PCKEY_DOWN               0x02
#define PCKEY_RIGHT              0x03
#define PCKEY_LEFT               0x04
#define PCKEY_PAGE_UP            0x05
#define PCKEY_PAGE_DOWN           0x06
#define PCKEY_INSERT              0x07
#define PCKEY_BS                 ASCII_BS /* 0x08 for compatibility with Backspace*/
#define PCKEY_TAB                ASCII_TAB /* 0x09 for compatibility with Tab */
#define PCKEY_ENTER               ASCII_LF /* 0x0A for compatibility with New Line(Enter) */
#define PCKEY_F1                  0x0B
#define PCKEY_F2                  0x0C
#define PCKEY_F3                  0x0D
#define PCKEY_F4                  0x0E
#define PCKEY_F5                  0x0F
#define PCKEY_F6                  0x10

```

```
#define PCKEY_F7      0x11
#define PCKEY_F8      0x12
#define PCKEY_F9      0x13
#define PCKEY_F10     0x14
#define PCKEY_F11     0x15
#define PCKEY_F12     0x16
#define PCKEY_HOME    0x17
#define PCKEY_PAUSE   0x18
#define PCKEY_ALT     0x19      /* Warning: cannot be read through console */
#define PCKEY_SCROLL_LOCK 0x1A      /* Warning: cannot be read through console */
#define PCKEY_ESC      ASCII_ESC /* 0x1B Reserved for compatibility with Escape */
#define PCKEY_CAPS_LOCK 0x1C      /* Warning: cannot be read through console */
#define PCKEY_SHIFT    0x1D      /* Warning: cannot be read through console */
#define PCKEY_CONTROL  0x1E      /* Warning: cannot be read through console */
#define PCKEY_NUM_LOCK 0x1F      /* Warning: cannot be read through console */
#define PCKEY_DEL      ASCII_DEL /* 0x7F Reserved for compatibility with Delete */
#define PCKEY_END      0x80
```

## 7. Librería Algebraica “lb\_algebra.c”

Esta librería contiene funciones esenciales del álgebra de vectores y de matrices. No se pretende hacer un cubrimiento general del tema, sino hacer una selección de las funciones más comunes. La expansión se deja a cargo de interesados o para desarrollos futuros.

**Funciones de comprobación de integridad:** estas funciones comprueban la existencia de errores en la definición de los arreglos, como por ejemplo dimensiones cero o negativas. Existen diferentes versiones de acuerdo al tipo de datos.

### Funciones de comprobación de validez dimensional

```
SINT8_T    lb_al_assert_dimensions_array_r(ARRAY_R_T *M);
SINT8_T    lb_al_assert_dimensions_matrix_p2d(MATRIX_POINT_2D_REAL_T *S);
SINT8_T    lb_al_assert_dimensions_matrix_p3d(MATRIX_POINT_3D_REAL_T *S);
SINT8_T    lb_al_assert_dimensions_matrix_r(MATRIX_R_T *M);
SINT8_T    lb_al_assert_dimensions_matrix_si8(MATRIX_SINT8_T *M);
SINT8_T    lb_al_assert_dimensions_vector_c(VECTOR_C_T *V);
SINT8_T    lb_al_assert_dimensions_vector_r(VECTOR_R_T *V);
SINT8_T    lb_al_assert_dimensions_vector_si8(VECTOR_SINT8_T *V);
SINT8_T    lb_al_assert_dimensions_vector_si16(VECTOR_SINT16_T *V);
```

### Funciones de adición o substracción de vectores reales o complejos

```
void    lb_al_add_vector_r(VECTOR_R_T *A, VECTOR_R_T *B, VECTOR_R_T *C);
void    lb_al_add_vector_c(VECTOR_C_T *A, VECTOR_C_T *B, VECTOR_C_T *C);
void    lb_al_subtract_vector_c(VECTOR_C_T *A, VECTOR_C_T *B, VECTOR_C_T *C);
void    lb_al_subtract_vector_r(VECTOR_R_T *A, VECTOR_R_T *B, VECTOR_R_T *C);
```

### Suma de los elementos de vectores reales, complejos, y de matrices

```
REAL_T    lb_al_sum_vector_r(VECTOR_R_T *V);
COMPLEX_T lb_al_sum_vector_c(VECTOR_C_T *V);
REAL_T    lb_al_sum_matrix_r(MATRIX_R_T *M);
```

### Suma del valor absoluto de los elementos de vectores y matrices

```
REAL_T    lb_al_sum_abs_matrix_r(MATRIX_R_T *M);
```

```

REAL_T    lb_al_sum_abs_vector_c(VECTOR_C_T *V, MATHERROR_T *error);
REAL_T    lb_al_sum_abs_vector_r(VECTOR_R_T *V);

```

### Funciones de copia de una columna (o una fila) de una matriz a un vector

```

void    lb_al_copy_matrix_r_col_to_vector_r(MATRIX_R_T *M, VECTOR_R_T *V, UINT16_T j);
void    lb_al_copy_matrix_r_row_to_vector_r(MATRIX_R_T *M, VECTOR_R_T *V, UINT16_T i);

```

### Funciones de copia de una matriz (o de una matriz de 3x3) hacia un destino

```

void    lb_al_copy_matrix_r(MATRIX_R_T *M1, MATRIX_R_T *M2);
void    lb_al_copy_matrix33_r(REAL_T M1[3][3], REAL_T M2[3][3]);

```

### Función de copia del menor de una matriz

```

void    lb_al_copy_minor_matrix_r(MATRIX_R_T *M1, MATRIX_R_T *M2, UINT16_T io, UINT16_T jo);

```

### Funciones de copia de un vector real o complejo

```

void    lb_al_copy_vector_c(VECTOR_C_T *V1, VECTOR_C_T *V2);
void    lb_al_copy_vector_r(VECTOR_R_T *V1, VECTOR_R_T *V2);

```

### Funciones de creación y destrucción de arreglos en memoria dinámica

Las dimensiones se especifican como parámetros de cada estructura.

```

void    lb_al_create_array_r(ARRAY_R_T *M);
void    lb_al_create_matrix_p2d(MATRIX_POINT_2D_REAL_T *S);
void    lb_al_create_matrix_p3d(MATRIX_POINT_3D_REAL_T *S);
void    lb_al_create_matrix_r(MATRIX_R_T *M);
void    lb_al_create_matrix_si8(MATRIX_SINT8_T *M);
void    lb_al_create_vector_c(VECTOR_C_T *V);
void    lb_al_create_vector_r(VECTOR_R_T *V);
void    lb_al_create_vector_si8(VECTOR_SINT8_T *V);
void    lb_al_create_vector_si16(VECTOR_SINT16_T *V);
void    lb_al_release_array_r(ARRAY_R_T *M);
void    lb_al_release_matrix_p2d(MATRIX_POINT_2D_REAL_T *S);

```

```

void lb_al_release_matrix_p3d(MATRIX_POINT_3D_REAL_T *S);
void lb_al_release_matrix_r(MATRIX_R_T *M);
void lb_al_release_matrix_si8(MATRIX_SINT8_T *M);
void lb_al_release_vector_c(VECTOR_C_T *V);
void lb_al_release_vector_r(VECTOR_R_T *V);
void lb_al_release_vector_si8(VECTOR_SINT8_T *V);
void lb_al_release_vector_si16(VECTOR_SINT16_T *V);

```

### Funciones de producto externo o cruz entre vectores

```

void lb_al_cross_product_vector_r(VECTOR_R_T *V1, VECTOR_R_T *V2, VECTOR_R_T *V3);
void lb_al_cross_product_vector3_r(REAL_T V1[3], REAL_T V2[3], REAL_T V3[3]);

```

### Funciones de eliminación o inserción de elemento(s) de un vector real

```

void lb_al_delete_item_vector_r(VECTOR_R_T *V, UINT16_T pos);
void lb_al_delete_n_items_vector_r(VECTOR_R_T *V, UINT16_T pos, UINT16_T n);
void lb_al_insert_item_vector_r(VECTOR_R_T *V, REAL_T x, UINT16_T pos);

```

### Funciones de cálculo de determinantes

Se proveen versiones recursivas y no-recursivas por el método de Gauss-Jordan, para matrices de tamaño arbitrario, y solución simplificada para matrices 3x3:

```

REAL_T lb_al_determinant_matrix_r(MATRIX_R_T *M);
REAL_T lb_al_determinant_matrix_r_gauss(MATRIX_R_T *M);
REAL_T lb_al_determinant_matrix22_r(REAL_T M[2][2]);
REAL_T lb_al_determinant_matrix33_r(REAL_T M[3][3]);

```

### Producto punto entre vectores reales

```
REAL_T lb_al_dot_product_vector_r(VECTOR_R_T *V1, VECTOR_R_T *V2);
```

### Función de llenado de tipos especiales de matrices

Función de llenado de una matriz identidad:

```
void lb_al_fill_identity_matrix_r(MATRIX_R_T *M);
```

Función de llenado de una matriz de ceros:

```
void lb_al_fill_zeros_matrix_r(MATRIX_R_T *M);
```

### Generación de matrices de rotación

Generación de una matriz de rotación con base en los ángulos Tait-Bryan o aeronáuticos:

```
void lb_al_fill_rotation_matrix_tait_bryan_ZYX(MATRIX_R_T *R, REAL_T yaw, REAL_T pitch,  
REAL_T roll);  
  
void lb_al_fill_rotation_matrix33_tait_bryan_ZYX(REAL_T R[3][3], REAL_T yaw, REAL_T  
pitch, REAL_T roll);
```

Generación de una matriz de rotación que permita hacer giros de un ángulo alrededor de un vector (x, y, z):

```
void lb_al_fill_rotation_matrix_vector(MATRIX_R_T *R, REAL_T x, REAL_T y, REAL_T z,  
REAL_T angle);
```

Generación de una matriz de rotación que permita hacer giros de un ángulo alrededor de los ejes X, Y, y Z:

```
void lb_al_fill_rotation_matrix_X(MATRIX_R_T *R, REAL_T angle_x);  
void lb_al_fill_rotation_matrix_Y(MATRIX_R_T *R, REAL_T angle_y);  
void lb_al_fill_rotation_matrix_Z(MATRIX_R_T *R, REAL_T angle_z);  
  
void lb_al_fill_rotation_matrix33_X(REAL_T R[3][3], REAL_T angle_x);  
void lb_al_fill_rotation_matrix33_Y(REAL_T R[3][3], REAL_T angle_y);  
void lb_al_fill_rotation_matrix33_Z(REAL_T R[3][3], REAL_T angle_z);
```

### Funciones de interpolación

Interpolación lineal de datos con base en datos de una matriz:

```
void lb_al_inter_matrix(MATRIX_R_T *M, REAL_T ir, REAL_T jr, REAL_T *Q);
```

### Multiplicación matricial

Multiplicación matricial, con versión de almacenamiento condicional a C diferente de A y B; y arbitrario (hay diferencias de desempeño). Estas funciones podrían integrarse en una sola versión:

```
void lb_al_multiply_matrix_r(MATRIX_R_T *A, MATRIX_R_T *B, MATRIX_R_T *C);
```

```
void lb_al_multiply_matrix_r_copy(MATRIX_R_T *A, MATRIX_R_T *B, MATRIX_R_T *C);
```

Multiplicación para matrices de 3x3, con versión de almacenamiento condicional a C diferente de A y B; y arbitrario (hay diferencias de desempeño). Estas funciones podrían integrarse en una sola versión:

```
void lb_al_multiply_matrix33_r(REAL_T A[3][3], REAL_T B[3][3], REAL_T C[3][3]);  
void lb_al_multiply_matrix33_r_copy(REAL_T A[3][3], REAL_T B[3][3], REAL_T C[3][3]);
```

Multiplicación de matriz por un vector, con almacenamiento en vector (sujeta a compatibilidad de las dimensiones):

```
void lb_al_multiply_matrix_r_vector_r(MATRIX_R_T *A, VECTOR_R_T *B, VECTOR_R_T *C);  
void lb_al_multiply_matrix33_r_vector_r(REAL_T A[3][3], REAL_T B[3], REAL_T C[3]);
```

Multiplicación de un vectores reales o complejos por un número real:

```
void lb_al_multiply_vector_c_real(VECTOR_C_T *V1, REAL_T k);  
void lb_al_multiply_vector_c_real_copy(VECTOR_C_T *V1, REAL_T k, VECTOR_C_T *V2);  
void lb_al_multiply_vector_r_real(VECTOR_R_T *V1, REAL_T k);  
void lb_al_multiply_vector_r_real_copy(VECTOR_R_T *V1, REAL_T k, VECTOR_R_T *V2);
```

### Cálculo de norma y normalización

```
REAL_T lb_al_norm_matrix_r(MATRIX_R_T *M, MATHERROR_T *error);  
REAL_T lb_al_norm_vector_r(VECTOR_R_T *V, MATHERROR_T *error);  
void lb_al_normalize_matrix_r(MATRIX_R_T *M, MATRIX_R_T *Mnorm, MATHERROR_T *error);  
void lb_al_normalize_vector_r(VECTOR_R_T *V, VECTOR_R_T *Vnorm, MATHERROR_T *error);
```

### Funciones de impresión de vectores y matrices

```
void lb_al_print_array_r(ARRAY_R_T *M, const char *text, SINT8_T len, SINT8_T dec);  
void lb_al_print_matrix_r(MATRIX_R_T *M, char *text, const char *format);  
void lb_al_print_matrix33_r(REAL_T M[3][3], char *text, const char *format);  
void lb_al_print_vector_c(VECTOR_C_T *V, char *text, const char *format);  
void lb_al_print_vector_r(VECTOR_R_T *V, char *text, const char *format);  
void lb_al_print_vector_si8(VECTOR_SINT8_T *V, char *text);  
void lb_al_print_vector_si16(VECTOR_SINT16_T *V, char *text);
```

### Funciones de redimensionamiento de vectores

```
void lb_al_resize_vector_r(VECTOR_R_T *V, UINT16_T new_size);
void lb_al_resize_vector_si16(VECTOR_SINT16_T *V, UINT16_T new_size);
```

### Reversión de orden de un vector real

```
void lb_al_reverse_order_vector_r(VECTOR_R_T *V1, VECTOR_R_T *V2);
```

### Solución de sistemas lineales :

```
void lb_al_solve_linear_system_r(MATRIX_R_T *M, VECTOR_R_T *X, VECTOR_R_T *S);
```

### Métodos de ordenación

Se incluyen funciones con los métodos de “burbuja” y “quicksort”:

```
void lb_al_sort_bubble_vector_r(VECTOR_R_T *V1, VECTOR_R_T *V2);
void lb_al_sort_bubble_vector_r_unreg(REAL_T *V, UINT16_T n);
void _lb_al_sort_quicksort_vector_r(VECTOR_R_T *V, UINT16_T a, UINT16_T b);
void lb_al_sort_quicksort_vector_r(VECTOR_R_T *V1, VECTOR_R_T *V2);
```

### Intercambio de contenido entre vectores reales

```
void lb_al_swap_vector_r(VECTOR_R_T *A, VECTOR_R_T *B);
```

### Transposición de matrices reales

```
void lb_al_transpose_matrix_r(MATRIX_R_T *M1, MATRIX_R_T *M2);
```

## 8. Librería de Análisis de Fourier “lb\_fourier.c”

Esta librería es concisa, pero de gran utilidad, y cubre las transformadas discretas y rápidas, tanto en

### Transformadas directas

```
void lb_fo_DFT_C(VECTOR_C_T *T, VECTOR_C_T *F, SINT8_T sign, MATHERROR_T *error);  
void lb_fo_DFT_R(VECTOR_R_T *T, VECTOR_C_T *F);
```

### Transformadas rápidas

```
void lb_fo_FFT_C_recursive(VECTOR_C_T *T, VECTOR_C_T *F, SINT8_T sign);  
void lb_fo_calculate_W(VECTOR_C_T *W, SINT8_T sign);  
void lb_fo_FFT_C(VECTOR_C_T *T, VECTOR_C_T *F, VECTOR_C_T *W, SINT8_T sign);  
void lb_fo_FFT_R(VECTOR_R_T *T, VECTOR_C_T *F, VECTOR_C_T *W);  
void lb_fo_IFFT_R(VECTOR_C_T *T, VECTOR_C_T *F, VECTOR_C_T *W);
```

## 9. Librería de funciones en variable real “lb\_real.c”

Este módulo provee funciones generales en variable real.

### División, potencia, cuadrado, raíz, raíz-n

```
REAL_T lb_re_divide(REAL_T x, REAL_T y, MATHERROR_T *error);  
REAL_T lb_re_pow(REAL_T x, REAL_T y, MATHERROR_T *error);  
REAL_T lb_re_sqr(REAL_T x, MATHERROR_T *error);  
REAL_T lb_re_sqrt(REAL_T x, MATHERROR_T *error);  
REAL_T lb_re_xroot(REAL_T x, REAL_T y, MATHERROR_T *error);
```

### Exponencial, logaritmo natural, logaritmo decimal, logaritmo en base 2

```
REAL_T lb_re_exp(REAL_T x, MATHERROR_T *error);  
REAL_T lb_re_ln(REAL_T x, MATHERROR_T *error);  
REAL_T lb_re_log(REAL_T x, MATHERROR_T *error);  
REAL_T lb_re_log2(REAL_T x, MATHERROR_T *error);
```

### Funciones trigonométricas circulares

```
REAL_T lb_re_tan(REAL_T x, MATHERROR_T *error);  
REAL_T lb_re_asin(REAL_T x, MATHERROR_T *error);  
REAL_T lb_re_acos(REAL_T x, MATHERROR_T *error);  
REAL_T lb_re_atan2(REAL_T y, REAL_T x);
```

### Funciones trigonométricas hiperbólicas

```
REAL_T lb_re_sinh(REAL_T x, MATHERROR_T *error);  
REAL_T lb_re_cosh(REAL_T x, MATHERROR_T *error);  
REAL_T lb_re_acosh(REAL_T x, MATHERROR_T *error);  
REAL_T lb_re_atanh(REAL_T x, MATHERROR_T *error);
```

### Comparación

**Igualdad** (basada en una constante de tolerancia interna que aplica a todo el sistema):

```
SINT8_T lb_re_equal(REAL_T a, REAL_T b);
```

**Verificación si es número entero:**

```
SINT8_T lb_re_is_int(REAL_T x);
```

**Verificación “mayor o igual” y “menor igual” (basadas en constante de tolerancia):**

```
SINT8_T lb_re_larger_or_equal(REAL_T a, REAL_T b);  
SINT8_T lb_re_smaller_or_equal(REAL_T a, REAL_T b);
```

**Máximo y mínimo:**

```
REAL_T lb_re_max(REAL_T x, REAL_T y);  
REAL_T lb_re_min(REAL_T x, REAL_T y);
```

**Pertenencia a un intervalo cerrado y abierto:**

```
REAL_T lb_re_test_interval_closed(REAL_T a, REAL_T b, REAL_T x);  
REAL_T lb_re_test_interval_open(REAL_T a, REAL_T b, REAL_T x);
```

### Funciones singulares

```
REAL_T lb_re_frac(REAL_T x);  
SINT8_T lb_re_ispos(REAL_T x);  
REAL_T lb_re_ramp(REAL_T x);  
SINT8_T lb_re_sign(REAL_T x);  
REAL_T lb_re_pulse(REAL_T a, REAL_T b, REAL_T x);  
REAL_T lb_re_pulse_triangle(REAL_T a, REAL_T b, REAL_T x);  
REAL_T lb_re_step(REAL_T x);  
REAL_T lb_re_step_practical(REAL_T t, REAL_T x);
```

### Otras funciones

**Número mínimo de dígitos que se requieren para representar un número:**

```
SINT16_T lb_re_ndigits(REAL_T x);
```

**Factorial:**

```
REAL_T lb_re_factorial(REAL_T n, MATHERROR_T *error);
```

**Exponentes y significando (mantisa) en forma normada de un número:**

```
REAL_T lb_re_normed_exponent(REAL_T x);  
REAL_T lb_re_normed_significand(REAL_T x);
```

## Funciones Financieras

Conversiones de tasa APR (“Annual Percentage Rate, half yearly, not in advanced”, o “Tasa anual equivalente sin reinversión de intereses basada en periodos semestrales”). Es una forma capciosa de presentar las tasas de interés a los clientes por los bancos canadienses.

```
REAL_T lb_re_APR_to_monthly(REAL_T apr, MATHERROR_T *error);  
REAL_T lb_re_monthly_to_APR(REAL_T monthly, MATHERROR_T *error);
```

**Conversión de tasa nominal a efectiva:**

```
REAL_T lb_re_monthly_to_effective(REAL_T monthly, MATHERROR_T *error);
```

**Interpolación:**

```
void lb_re_inter_linear(REAL_T x0, REAL_T Q0, REAL_T x1, REAL_T Q1, REAL_T x, REAL_T *Q,  
MATHERROR_T *error);  
void lb_re_inter_bilinear(REAL_T x0, REAL_T y0, REAL_T x1, REAL_T y1, REAL_T Q00, REAL_T  
Q01, REAL_T Q10, REAL_T Q11, REAL_T x, REAL_T y, REAL_T *Q, MATHERROR_T *error);
```

**Rotación:**

```
void lb_re_rotate(REAL_T x, REAL_T y, REAL_T angle, REAL_T *xr, REAL_T *yr);
```

## 10. Librería de Funciones Estadísticas “lb\_statistics.c”

Este módulo incorpora algunas funciones elementales de tipo estadístico. Se hace énfasis en las distribuciones de tipo gaussiano, pero la estructura y estilo pueden seguirse para otras distribuciones.

### Funciones estadísticas

**Función de error.** Calcula una aproximación a la función de error definida como:

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

```
REAL_T lb_st_erf(REAL_T x);
```

**Función de error complementario.** Calcula una aproximación a la función de error definida como:

$$erfc(x) = 1 - erf(x)$$

```
REAL_T lb_st_erfc(REAL_T x);
```

**Generación de un número aleatorio.** Generación de un número real aleatorio entre 0.0 y 1.0, siguiendo una distribución uniforme.

```
REAL_T lb_st_frand(void);
```

**Generación de dos números aleatorios gaussianos.** Usa el método polar de Marsaglia para generar dos números polares aleatorios independientes r1 y r2 con media 0.0 y varianza  $\sigma^2$  especificada por ‘variance’.

```
void lb_st_marsaglia_polar2(REAL_T variance, REAL_T *r1, REAL_T *r2);
```

**Generación de un número aleatorios gaussiano.** Se define por conveniencia esta función, la cual simplemente descarta uno de los números aleatorios generados por la función anterior, retornándolo por valor en lugar de referencia.

```
REAL_T lb_st_marsaglia_polar(REAL_T variance);
```

**Cálculo de la aproximación del área bajo la curva en una distribución normal.** Esta es quizás una de las operaciones más frecuentes en estadística aplicada, pues determina el área acumulada bajo la curva.

Se utiliza una aproximación polinomial de amplia difusión (la cual aparece con algunos cambios en una variedad de proyectos y ejemplos), la cual tiene la ventaja de ser rápida y fácil de computar, pero que debe usarse con cuidado, pues tiene un error máximo dado por  $|\epsilon(x)| \leq 1.5 \cdot 10^{-7}$ . La fuente original de esta práctica aproximación es el manual de fórmulas de ABRAMOWITZ, M. y STEGUN, I. [14].

```
REAL_T lb_st_gauss_area(REAL_T x);
```

Aproximación:

$$erf(x) = 1 - (a_1 \cdot t + a_2 \cdot t^2 + a_3 \cdot t^3 + a_4 \cdot t^4 + a_5 \cdot t^5)e^{-x^2} + \epsilon(x)$$

Donde:

$$t = \frac{1}{1 + p \cdot x}$$

$$p = 0.32759$$

$$a_1 = 0.254829592$$

$$a_2 = -0.284496736$$

$$a_3 = 1.421413741$$

$$a_4 = -1.453152027$$

$$a_5 = 1.061405429$$

Otra forma (no implementada) de aproximar esta solución, es el uso de la serie de MacLaurin:

$$erfc(x) = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{n! (2 \cdot n + 1)}$$

**Cálculo de la media aritmética.** Obtiene la media aritmética de los elementos almacenados en el vector Data.

```
REAL_T lb_st_average(VECTOR_R_T *Data);
```

**Cálculo de la desviación estándar.** Obtiene la varianza  $\sigma^2$  de los elementos almacenados en el vector Data con base en la media ‘mu’.

```
REAL_T lb_st_stddev2(VECTOR_R_T *Data, REAL_T mu);
```

**Generación de un histograma.** Toma los elementos de un vector de datos de tamaño arbitrario y obtiene su distribución o “histograma” en intervalos igualmente espaciados entre el intervalo cerrado [a ; b], utilizando un número de recipientes igual al número de elementos del vector ‘Bins’ (contenedores). Se retornan, como parte del análisis, la media aritmética y la varianza. Esta función se encarga únicamente del análisis de los datos; para su representación gráfica se puede utilizar `lb_gr_draw_histogram()` o cualquier otro método.

```
void lb_st_histogram(VECTOR_R_T *Data, VECTOR_R_T *Bins, REAL_T a, REAL_T b, REAL_T *mu,
REAL_T *sigma2 );
```

## 11. Funciones de variable compleja “lb\_complex.c”

Esta librería incluye funciones de uso común en variable compleja.

### Operadores básicos en variable compleja

```
COMPLEX_T lb_cp_add(COMPLEX_T a, COMPLEX_T b);
COMPLEX_T lb_cp_subtract(COMPLEX_T a, COMPLEX_T b);
COMPLEX_T lb_cp_neg(COMPLEX_T a);
COMPLEX_T lb_cp_multiply(COMPLEX_T a, COMPLEX_T b);
COMPLEX_T lb_cp_multiply_real(COMPLEX_T c, REAL_T r);
COMPLEX_T lb_cp_divide_real(COMPLEX_T c, REAL_T r, MATHERROR_T *error);
COMPLEX_T lb_cp_divide(COMPLEX_T a, COMPLEX_T b, MATHERROR_T *error);
COMPLEX_T lb_cp_conjugate(COMPLEX_T a);
```

### Operadores de potencia en variable compleja

```
COMPLEX_T lb_cp_sqr(COMPLEX_T z, MATHERROR_T *error);
COMPLEX_T lb_cp_sqrt(COMPLEX_T z, MATHERROR_T *error);
COMPLEX_T lb_cp_power(COMPLEX_T z1, COMPLEX_T z2, MATHERROR_T *error);
```

### Operadores exponenciales y logarítmicos en variable compleja

```
COMPLEX_T lb_cp_exp(COMPLEX_T z, MATHERROR_T *error);
COMPLEX_T lb_cp_ln(COMPLEX_T z, MATHERROR_T *error);
COMPLEX_T lb_cp_log(COMPLEX_T z, MATHERROR_T *error);
COMPLEX_T lb_cp_log2(COMPLEX_T z, MATHERROR_T *error);
COMPLEX_T lb_cp_logb(COMPLEX_T b, COMPLEX_T z, MATHERROR_T *error);
```

### Funciones trigonométricas circulares en variable compleja

```
COMPLEX_T lb_cp_sin(COMPLEX_T z, MATHERROR_T *error);
COMPLEX_T lb_cp_cos(COMPLEX_T z, MATHERROR_T *error);
COMPLEX_T lb_cp_tan(COMPLEX_T z, MATHERROR_T *error);
COMPLEX_T lb_cp_asin(COMPLEX_T z, MATHERROR_T *error);
COMPLEX_T lb_cp_acos(COMPLEX_T z, MATHERROR_T *error);
```

```
COMPLEX_T lb_cp_atan(COMPLEX_T z, MATHERROR_T *error);
```

### Funciones trigonométricas hiperbólicas en variable compleja

```
COMPLEX_T lb_cp_sinh(COMPLEX_T z, MATHERROR_T *error);
COMPLEX_T lb_cp_cosh(COMPLEX_T z, MATHERROR_T *error);
COMPLEX_T lb_cp_tanh(COMPLEX_T z, MATHERROR_T *error);
COMPLEX_T lb_cp_asinh(COMPLEX_T z, MATHERROR_T *error);
COMPLEX_T lb_cp_acosh(COMPLEX_T z, MATHERROR_T *error);
COMPLEX_T lb_cp_atanh(COMPLEX_T z, MATHERROR_T *error);
```

### Conversión de formato

```
COMPLEX_T lb_cp_complex(REAL_T r, REAL_T i);
COMPLEX_T lb_cp_polar_to_rect(REAL_T mag, REAL_T angle);
COMPLEX_T lb_cp_rect_to_polar(REAL_T x, REAL_T y);
```

### Parte magnitud, argumento, parte real y parte imaginaria

```
REAL_T lb_cp_abs(COMPLEX_T z);
REAL_T lb_cp_arg(COMPLEX_T z);
REAL_T lb_cp_re(COMPLEX_T z);
REAL_T lb_cp_im(COMPLEX_T z);
```

### Comparación de igualdad en números complejos

```
SINT8_T lb_cp_equal(COMPLEX_T z1, COMPLEX_T z2);
```

### Impresión

```
void lb_cp_print_c(COMPLEX_T z, const char *format);
```

## 12. Librería geométrica “lb\_geometry.c”

Esta unidad incluye algunas funciones que pueden ser útiles para simulaciones mecánicas o como funciones auxiliares para gráficas por computadora. Es una librería pequeña que se espera crezca según se necesite.

### Fórmulas de uso frecuente en simulaciones de mecanismos

Cálculo de la distancia de un punto a una línea:

```
REAL_T lb_ge_distance_point_to_line_coordinates(REAL_T x1, REAL_T y1, REAL_T x2, REAL_T  
y2, REAL_T x, REAL_T y, MATHERROR_T *error);
```

Cálculo de los ángulos tendidos a los puntos de tangencia dados un círculo y un punto exterior a éste:

```
void lb_ge_tangents_to_circle_point(REAL_T xc, REAL_T yc, REAL_T r, REAL_T xo, REAL_T  
yo, REAL_T *tetha1, REAL_T *tetha2);
```

### Expresiones aproximadas para la longitud y ángulos de elipses

```
REAL_T lb_ge_angle_from_length_ellipse_q1(REAL_T a, REAL_T b, REAL_T length, REAL_T R);  
REAL_T lb_ge_arclength_ellipse(REAL_T a, REAL_T b);  
REAL_T lb_ge_arclength_ellipse_angle(REAL_T a, REAL_T b, REAL_T t);  
REAL_T lb_ge_arclength_ellipse_angle_q1(REAL_T a, REAL_T b, REAL_T t, REAL_T R);  
REAL_T lb_ge_arclength_ellipse_angles(REAL_T a, REAL_T b, REAL_T t1, REAL_T t2);
```

### **13. Librería para uso del GPIO (General Purpose Input and Output) Puerto de Propósito General de Entrada y Salida “lb\_gpio.c”**

Esta librería abre las puertas a la interconexión del sistema a dispositivos físicos para elaborar aplicaciones de control, medida, o interactivas. Soporta a la fecha el modelo Raspberry Pi 3, y con algunos cambios, puede ajustarse a modelos anteriores.

#### **Constantes**

Constantes para el acceso al puerto y para el mapeo (solamente pueden requerir cambios para otras versiones de hardware):

```
#define PHYSICAL_ADDRESS 0x3F000000  
#define GPIO_OFFSET      0x00200000  
#define BLOCK_SIZE        (4*1024)
```

Definición de valores para configuración de pines como “entrada” o “salida” u para estados “alto” y “bajo”:

```
#define GPIO_INPUT      1  
#define GPIO_OUTPUT     0  
#define GPIO_HIGH       1  
#define GPIO_LOW        0
```

#### **Variable**

Variable global que contiene la dirección mapeada al puerto GPIO:

```
extern volatile unsigned int *lb_gp_gpio;
```

#### **Macros de configuración**

Estas macros permiten definir pines específicos del GPIO como entradas, salidas, o configura la función alterna. (Este código es provisional y en parte procede de un ejemplo de uso. Se han hecho mejoras, pero tiene deficiencias como baja legibilidad, tales como paréntesis innecesarios).

```
#define GPIO_SET_INP(g)    *(lb_gp_gpio + ((g)/10)) &= ~(7<<(((g)%10)*3))  
#define GPIO_SET_OUT(g)   *(lb_gp_gpio + ((g)/10)) |= (1<<(((g)%10)*3))  
#define GPIO_SET_ALT(g,a) *(lb_gp_gpio + (((g)/10))) |= (((a)<=3?(a) +  
4:(a)==4?3:2)<<(((g)%10)*3))
```

#### **Macros de escritura y lectura**

```
#define GPIO_SET *(lb_gp_gpio + 7)      /* Sets bits which are 1. Leaves unchanged those  
                                         which are 0 */
```

```

#define GPIO_CLR *(lb_gp_gpio + 10)      /* Clears bits which are 1. Leaves unchanged those
                                         which are 0 */

#define GPIO_READ(g) *(lb_gp_gpio + 13)

#define GPIO_PULL *(lb_gp_gpio + 37)

#define GPIO_PULLCLK0 *(lb_gp_gpio + 38)

```

## Estructura de puerto virtual SPI

Esa estructura define un puerto virtual SPI. Cada miembro indica el pin a ser utilizado, por ejemplo, 5 en MOSI indica que el pin 8 del GPIO es utilizado para escritura del Maestro al Esclavo. Esta definición provee gran flexibilidad, pues pueden tantos puertos como se deseen. El retardo de reloj es configurable, así como un retardo adicional entre bytes sucesivos.

```

typedef struct
{
    unsigned char MOSI;
    unsigned char CLK;
    unsigned char MISO;
    unsigned char CPOL;
    unsigned char CPHA;
    unsigned int delay_clk;
    unsigned int delay_byte;
} SPI_PORT_T;

```

## Funciones de inicialización

**Función de inicialización.** Prepara el puerto de propósito general de entrada y salida GPIO para su uso, mapeando internamente las direcciones de acceso al puerto con un área en memoria.

```
int      lb_gp_gpio_open();
```

**Función de cierre del GPIO.** Esta función debe llamarse durante el cierre del programa o cuando no se vaya a utilizar más el puerto de propósito general de entrada y salida GPIO.

```
void      lb_gp_gpio_close();
```

## Funciones de configuración

**Configuración de un pin individual.** Configura un pin específico del GPIO como entrada o salida. El valor ‘1’ lo configura como entrada, mientras que el valor ‘0’ lo hace como salida. El siguiente mnemónico es común entre los desarrolladores de sistemas embebidos: 1nput, 0output.

```
void      lb_gp_gpio_setup_pin(UINT8_T pin_number, UINT8_T mode);
```

## Funciones de lectura y escritura

**Lectura de un pin específico.** Lee el estado de uno de los pines específicos del GPIO.

```
UINT8_T lb_gp_gpio_rd(UINT8_T pin_number);
```

**Lectura del estado de todos los pines.** Lee, al mismo tiempo, el estado de todos los pines del GPIO.

```
UINT32_T lb_gp_gpio_rd_all(void);
```

**Escritura de un pin específico.** Escribe el bit más bajo de ‘value’ en el pin especificado del GPIO.

```
void lb_gp_gpio_wr(UINT8_T pin_number, UINT8_T value);
```

## Funciones de puerto SPI

**Función de intercambio SPI (Serial Peripheral Interface) a nivel de 1 byte.** Ejecuta una transacción SPI de 1 byte utilizando el puerto virtual definido previamente en la estructura ‘port’, cuyos pines deben haber sido configurados previamente como entradas y salida, para de lectura y escritura. ‘value’ es el dato a ser escrito; el valor retornado por la función es el valor leído del dispositivo. Al tratarse de un puerto virtual, lo más común es que el puerto sea el ‘Maestro’, y el dispositivo siendo leído sea el ‘Esclavo’.

```
UINT8_T lb_gp_gpio_SPI_rw(SPI_PORT_T *port, UINT8_T byte_out);
```

**Función de intercambio SPI (Serial Peripheral Interface) a nivel de n-bytes.**

Simplemente realiza una serie de transacciones, haciendo uso internamente de la función anterior, escribiendo los primeros n-bytes apuntados por ‘buffer\_out’, almacenando el resultado de cada transacción en ‘buffer\_in’. Ambos arreglos deben ser haber sido declarados estáticamente o asignados dinámicamente.

```
void lb_gp_gpio_SPI_rw_buffer(SPI_PORT_T *port, UINT8_T *buffer_out, UINT8_T *buffer_in, UINT8_T n_bytes);
```

**Función de intercambio SPI (Serial Peripheral Interface) a nivel de n-bits.** Realiza una transacción SPI a nivel de n-bits:

```
UINT16_T lb_gp_gpio_SPI_rw_nbites(SPI_PORT_T *port, UINT16_T int16_out, UINT8_T n_bits);
```

## Funciones de impresión o depuración

Impresión de un número de n-bits (n menor o igual a 32) en formato binario. Esta función imprime los `n_bits` más bajos de la variable no signada ‘value’.

```
void    lb_gp_print_u32_as_binary(UINT32_T value, UINT8_T n_bits);
```

## 14. Librería gráfica “lb\_graphics.c”

Esta librería permite la creación de bloques pictóricos en memoria RAM (tantos como se deseen o permitan los recursos del sistema), con un alto nivel de control en manos del usuario y con mínimos niveles de abstracción. Incluye un juego básico de primitivas que se extiende en la medida que surgen nuevas ideas o necesidades.

### Funciones de inicialización y de cierre

Estas funciones facilitan la inicialización y el cierre del controlador por defecto, el cual es SDL (Simple DirectMedia Layer). Es posible utilizar otros métodos de despliegue de las imágenes, o incluso no utilizar ninguno, exportando por ejemplo el resultado a un archivo tipo BMP o JPG.

```
void lb_gr	SDL_init(const char *title, Uint32 flags, SINT16_T width, SINT16_T height,  
UINT8_T r, UINT8_T g, UINT8_T b);  
  
void lb_gr	SDL_close();
```

Hay dos grupos de funciones de primitivas gráficas: aquellas que operan en forma independiente del dispositivo, y otras que se han optimizado para su uso con el driver por defecto, el cual es el framebuffer de SDL (Simple DirectMedia Layer). La única razón para duplicar algunas funciones es simplemente alcanzar un poco más de velocidad, y el deseo de experimentación.

### Funciones “fb” o “framebuffer”

**Línea horizontal.** Dibuja una línea horizontal entre (x0, y0), y (x1, y0), con colores r, g, b en 8 bits, con copia directa (sin realizar ninguna operación con el fondo)

```
void lb_gr_fb_line_h(SCREEN_T *screen, SINT16_T y0, SINT16_T x0, SINT16_T x1,  
UINT8_T r, UINT8_T g, UINT8_T b);
```

**Línea horizontal con modo de copia.** Dibuja una línea horizontal entre (x0, y0), y (x1, y0), con colores r, g, b en 8 bits, con opción de operación de combinación con el fondo.

```
void lb_gr_fb_line_h_copymode(SCREEN_T *screen, SINT16_T y0, SINT16_T x0, SINT16_T x1,  
UINT8_T r, UINT8_T g, UINT8_T b, UINT8_T a, COPYMODE_T copymode);
```

**Línea vertical.** Dibuja una línea vertical entre (x0, y0), y (x0, y1), con colores r, g, b en 8 bits, con copia directa (sin realizar ninguna operación con el fondo)

```
void lb_gr_fb_line_v(SCREEN_T *screen, SINT16_T x0, SINT16_T y0, SINT16_T y1,  
UINT8_T r, UINT8_T g, UINT8_T b);
```

**Línea vertical con modo de copia.** Dibuja una línea vertical entre (x0, y0), y (x0, y1), con colores r, g, b en 8 bits, con opción de operación de combinación con el fondo.

```
void lb_gr_fb_line_v_copymode(SCREEN_T *screen, SINT16_T x0, SINT16_T y0, SINT16_T y1,  
    UINT8_T r, UINT8_T g, UINT8_T b, UINT8_T a, COPYMODE_T copymode);
```

**Rectángulo.** Traza un rectángulo entre (x0, y0) y (x1, y1) incluyendo los segmentos x=x0, x=x1, y=y0, y y=y1.

```
void lb_gr_fb_rectangle(SCREEN_T *screen, SINT16_T x0, SINT16_T y0, SINT16_T x1, SINT16_T  
    y1, UINT8_T r, UINT8_T g, UINT8_T b);
```

**Rectángulo con opción de operación de combinación con el destino.** Traza un rectángulo, con opción de operación de combinación con el fondo, entre (x0, y0) y (x1, y1) incluyendo los segmentos x=x0, x=x1, y=y0, y y=y1.

```
void lb_gr_fb_rectangle_copymode(SCREEN_T *screen, SINT16_T x0, SINT16_T y0, SINT16_T x1,  
    SINT16_T y1, UINT8_T r, UINT8_T g, UINT8_T b, UINT8_T a, COPYMODE_T copymode);
```

**Función de escritura (no validada) de pixel en formato ARGB (canal auxiliar, rojo, verde y azul).** Escribe un pixel (x,y) con color A, R, G y B, en el buffer de destino sin verificar si excede o no los límites del mismo. Si se usa incorrectamente, causará corrupción de memoria, inestabilidad del sistema o errores catastróficos. Es altamente eficiente.

```
void _lb_gr_fb_setpixel_ARGB(SCREEN_T *screen, SINT16_T x, SINT16_T y,  
    UINT8_T r, UINT8_T g, UINT8_T b, UINT8_T a);
```

**Función de escritura validada de pixel en formato ARGB.** Comprueba que la posición de destino no exceda los límites del buffer, y, en caso de que la posición deseada sea válida, escribe un pixel con valores ARGB.

```
void lb_gr_fb_setpixel_ARGB(SCREEN_T *screen, SINT16_T x, SINT16_T y, UINT8_T r, UINT8_T  
    g, UINT8_T b, UINT8_T a);
```

**Función de escritura validada de pixel en formato ARGB, con opción de operación de combinación con el fondo.** Comprueba que la posición de destino no exceda los límites del buffer, y, en caso de que la posición deseada sea válida, escribe un pixel con valores ARGB.

```
void lb_gr_fb_setpixel_ARGB_copymode(SCREEN_T *screen, SINT16_T x, SINT16_T y, UINT8_T  
    src_r, UINT8_T src_g, UINT8_T src_b, UINT8_T src_a, COPYMODE_T copymode);
```

**Función de escritura (no validada) de pixel en formato XRGB.** No copia el valor del canal auxiliar o “alpha”. Debe usarse con precaución dado que no efectúa comprobación de límites.

```
void _lb_gr_fb_setpixel_XRGB(SCREEN_T *screen, SINT16_T x, SINT16_T y, UINT8_T r, UINT8_T  
    g, UINT8_T b);
```

**Función de escritura validada de pixel en formato XRGB.** Deja intacto el posible valor de “alpha” o canal auxiliar que contenga el framebuffer.

```
void lb_gr_fb_setpixel_XRGB(SCREEN_T *screen, SINT16_T x, SINT16_T y, UINT8_T r, UINT8_T g, UINT8_T b);
```

**Conversión del formato de color de 12 bits al modo actual.** Convierte un color en formato de 12 bits (a:4, b:4, g:4, r:4), de común uso en el sistema, al número de bits de cada canal requerido por `PIXEL_T`. Por ejemplo, 0x000F indica rojo. Si `PIXEL_T` se define con 8 bits por canal, `lb_gr_12RGB()` retornará un color cercano a 0x000000FF.

```
PIXEL_T lb_gr_12RGB(UINT16_T number);
```

### Funciones de comprobación de integridad dimensional

Verifican que las estructuras respectivas no excedan los límites establecidos:

```
SINT8_T lb_gr_assert_dimensions_line2d_i(LINE_2D_SINT16_T *L);
SINT8_T lb_gr_assert_dimensions_line2d_f(LINE_2D_REAL_T *L);
SINT8_T lb_gr_assert_dimensions_line3d_f(LINE_3D_REAL_T *L);
SINT8_T lb_gr_assert_dimensions_picture(PICTURE_T *Pic);
void lb_gr_refresh();
```

### Primitivas de dibujo de líneas

**Función de línea genérica.** Es una función “empaquetadora”, la cual, con base en las opciones definidas en los parámetros, llama a otras funciones de línea eligiendo aquellas que sean más eficientes. Soporta un grosor arbitrario.

```
void lb_gr_draw_line(PICTURE_T *Pic, REAL_T x0, REAL_T y0, REAL_T x1, REAL_T y1, REAL_T w,
PIXEL_T color, COPYMODE_T copymode, LINEMODE_T linemode);
```

**Función de trazo de línea escalonada con grosor unitario.** Traza una línea siguiendo el algoritmo de Bresenham, con opción de selección de operación de combinación con el fondo.

```
void lb_gr_draw_line1(PICTURE_T *Pic, SINT16_T x0, SINT16_T y0, SINT16_T x1, SINT16_T y1,
PIXEL_T color, COPYMODE_T copymode);
```

**Función de trazo de línea escalonada con grosor unitario, en variable real, siguiendo la ecuación de la línea recta.** Traza una línea recta en variable real, siguiendo la ecuación que define la línea recta.

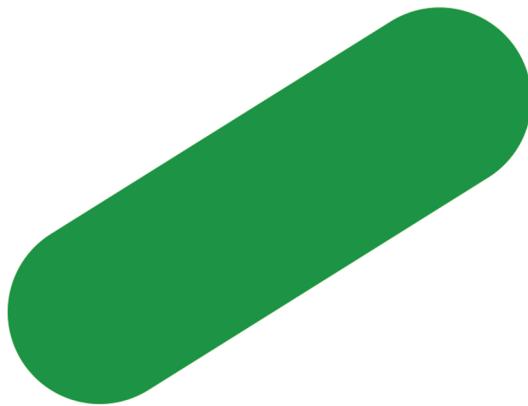
```
void lb_gr_draw_line1_r(PICTURE_T *Pic, REAL_T x0, REAL_T y0, REAL_T x1, REAL_T y1,
PIXEL_T color, COPYMODE_T copymode);
```

**Funciones de trazo de línea escalonada con grosor de dos y tres píxeles, en variable entera.** Trazan líneas rectas de grosor 2 y 3 píxeles, respectivamente, siguiendo una versión modificada del algoritmo de líneas de Bresenham. Son funciones de alta eficiencia, pero pueden brindar una apariencia escalonada.

```
void lb_gr_draw_line2(PICTURE_T *Pic, SINT16_T _x0, SINT16_T _y0, SINT16_T _x1, SINT16_T _y1, PIXEL_T color, COPYMODE_T copymode);  
void lb_gr_draw_line3(PICTURE_T *Pic, SINT16_T _x0, SINT16_T _y0, SINT16_T _x1, SINT16_T _y1, PIXEL_T color, COPYMODE_T copymode);
```

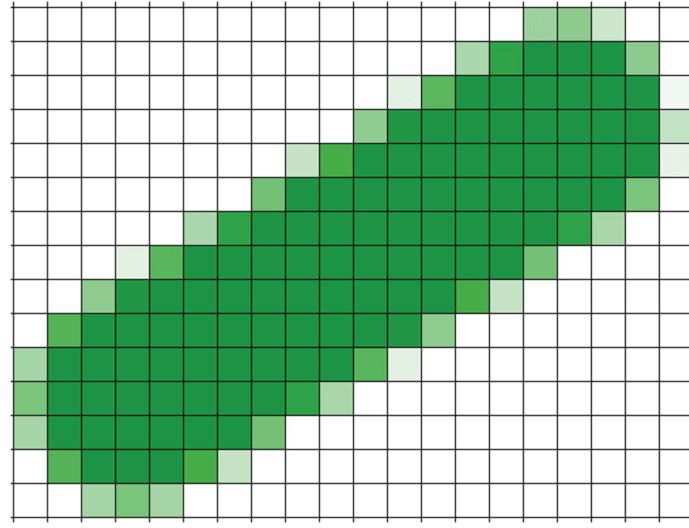
**Función avanzada de dibujo de línea con filtro anti-escalonamiento.** Es una función para el trazo refinado de líneas, la cual brinda una apariencia acabada y presenta un desempeño aceptable. Efectúa un redondeo en los extremos, soporta ancho arbitrario entero o fraccional, y permite efectuar una operación de combinación con el fondo.

```
void lb_gr_draw_line_antialiasing(PICTURE_T *Pic, REAL_T _xr0, REAL_T _yr0, REAL_T _xr1, REAL_T _yr1, REAL_T w, PIXEL_T color);
```



**Figura 27.** Representación de una línea con grosor arbitrario y filtro anti-escalonamiento.  
[Recurso propio].

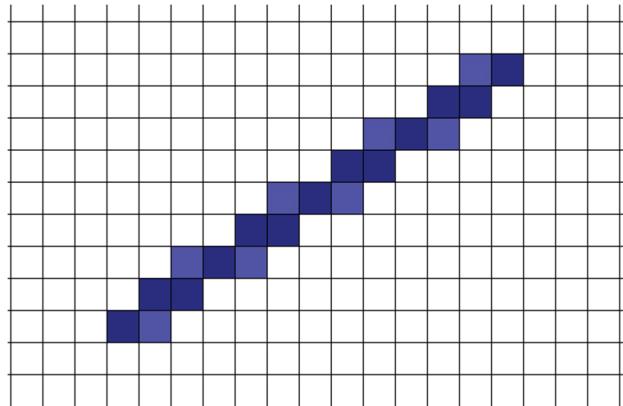
Las técnicas empleadas permiten lograr una apariencia de mayor calidad en dispositivos de baja resolución, como se muestra en la figura 28.



**Figura 28.** Detalle en baja resolución de una línea con grosor arbitrario y filtro anti-escalonamiento. [Recurso propio].

**Función de línea en variable entera con filtro anti-escalonamiento rápido y grosor de 2 pixeles.** Esta función permite el trazo de una línea con filtro anti-escalonamiento siguiendo una versión modificada del algoritmo de línea de Bressenham:

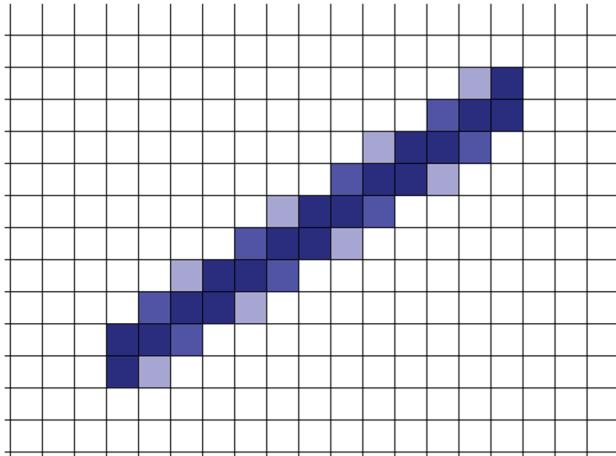
```
void lb_gr_draw_line_antialiasing2(PICTURE_T *Pic, SINT16_T _x0, SINT16_T _y0, SINT16_T
_x1, SINT16_T _y1, PIXEL_T color);
```



**Figura 29.** Detalle en baja resolución de una línea con grosor de dos pixeles y filtro rápido anti-escalonamiento. [Recurso propio].

**Función de línea en variable entera con filtro anti-escalonamiento rápido y grosor de 3 pixeles.** Esta función permite el trazo de una línea con filtro anti-escalonamiento siguiendo una versión modificada del algoritmo de línea de Bressenham:

```
void lb_gr_draw_line_antialiasing3(PICTURE_T *Pic, SINT16_T _x0, SINT16_T _y0, SINT16_T
_x1, SINT16_T _y1, PIXEL_T color);
```



**Figura 30.** Detalle en baja resolución de una línea con grosor de tres pixeles y filtro rápido anti-escalonamiento. [Recurso propio].

**Versión en variable real de la función de trazo de línea, filtro anti-escalonamiento, y grosor de 2 pixeles.**

```
void lb_gr_draw_line_antialiasing2_r(PICTURE_T *Pic, REAL_T _x0, REAL_T _y0, REAL_T _x1,
REAL_T _y1, PIXEL_T color);
```

**Versión en variable real de la función de trazo de línea, filtro anti-escalonamiento, y grosor de 3 pixeles.**

```
void lb_gr_draw_line_antialiasing3_f(PICTURE_T *Pic, REAL_T _x0, REAL_T _y0, REAL_T _x1,
REAL_T _y1, PIXEL_T color);
```

**Borrado de una imagen.** Copia el color representado por ‘`default_color`’ en la imagen de destino, sin realizar ninguna operación de combinación con los datos existentes.

```
void lb_gr_clear_picture(PICTURE_T *Pic, PIXEL_T default_color);
```

**Escritura de un pixel.** Esta función escribe un pixel con coordenadas (x, y), realizando una operación de combinación con el destino especificado por el parámetro ‘`copymode`’.

```
void lb_gr_draw_pixel(PICTURE_T *Pic, SINT16_T x, SINT16_T y, PIXEL_T pixel, COPYMODE_T
copymode);
```

**Dibujo de un rectángulo.** Llena el área rectangular definida por los puntos en variable entera (x0, y0) y (x1, y1), incluyendo los segmentos extremos, realizando una operación de combinación con el fondo especificada por el parámetro ‘`copymode`’ .

```
void lb_gr_draw_rectangle(PICTURE_T *Pic, SINT16_T x0, SINT16_T y0, SINT16_T x1, SINT16_T y1, PIXEL_T color, COPYMODE_T copymode);
```

**Dibujo de un rectángulo sin opción de combinación con el destino.** Es una función optimizada para llenar un área rectangular en modo de copia directa, es decir, sin considerar el contenido del destino y sin permitir operaciones de combinación con el mismo.

```
void lb_gr_draw_rectangle_solid(PICTURE_T *Pic, SINT16_T x0, SINT16_T y0, SINT16_T x1, SINT16_T y1, PIXEL_T color);
```

**Dibujo de una línea rectangular.** Dibuja una línea rectangular definida por los puntos en variable entera ( $x_0, y_0$ ) y ( $x_1, y_1$ ), realizando una operación de combinación con el fondo especificada por el parámetro `copymode`.

```
void lb_gr_draw_rectangle_line(PICTURE_T *Pic, SINT16_T x0, SINT16_T y0, SINT16_T x1, SINT16_T y1, SINT16_T w, PIXEL_T color, COPYMODE_T copymode);
```

**Dibujo de una línea rectangular con relleno o “barra”.** Dibuja una línea rectangular definida por los puntos ( $x_0, y_0$ ) y ( $x_1, y_1$ ) con el color de línea especificado por ‘`color_line`’. El interior del área, sin incluir la línea, se llena con el color especificado por ‘`color_fill`’.

```
void lb_gr_draw_rectangle_bar(PICTURE_T *Pic, SINT16_T x0, SINT16_T y0, SINT16_T x1, SINT16_T y1, SINT16_T w, PIXEL_T color_line, PIXEL_T color_background, COPYMODE_T copymode);
```

**Dibujo de un triángulo lleno.** Llena un triángulo definido por los vértices  $P_0(x_0,y_0)$ ,  $P_1(x_1,y_1)$ , y  $P_2(x_2,y_2)$  en variable entera, con el color especificado por ‘`color`’ y la operación de combinación con el destino definida por ‘`copymode`’.

```
void lb_gr_draw_triangle_fill_i(PICTURE_T *Pic, POINT_2D_SINT16_T P0, POINT_2D_SINT16_T P1, POINT_2D_SINT16_T P2, PIXEL_T color, COPYMODE_T copymode);
```

**Dibujo de un polígono.** Traza un polígono, el cual puede ser abierto o cerrado, definido en un arreglo de puntos bidimensionales en variable entera. Puede especificarse el tipo de operación de combinación con el fondo y el estilo de línea.

```
void lb_gr_draw_polygon_i(PICTURE_T *Pic, LINE_2D_SINT16_T *L, REAL_T w, PIXEL_T color, COPYMODE_T copymode, LINEMODE_T linemode);
```

**Dibujo de un polígono con filtro anti-escalonamiento.** Es una función de tipo preliminar que permite trazar polígonos con filtro anti-escalonamiento. Por el momento, examina la distancia entre cada uno de los puntos de la imagen a cada uno de los segmentos de recta que componen el polígono, lo cual no es eficiente. Sin embargo, ofrece un resultado de calidad y se deja provisionalmente como sujeta a mejoras.

```
void lb_gr_draw_polygon_antialiasing(PICTURE_T *Pic, LINE_2D_REAL_T *L, REAL_T w, PIXEL_T color);
```

**Dibujo de un polígono en variable real.** Traza un polígono, el cual puede ser abierto o cerrado, definido en un arreglo de puntos bidimensionales en variable real. Puede especificarse el tipo de operación de combinación con el fondo y el estilo de línea.

```
void lb_gr_draw_polygon_r(PICTURE_T *Pic, LINE_2D_REAL_T *L, REAL_T w, PIXEL_T color,  
COPYMODE_T copymode, LINEMODE_T linemode);
```

**Dibujo de un polígono lleno.** Traza un polígono cerrado, definido en un arreglo de puntos bidimensionales en variable entera. Las coordenadas del último punto del arreglo deben ser las mismas que el primero. Puede especificarse el tipo de operación de combinación con el fondo y el estilo de línea. Desarrollar una versión en variable entera de esta versión puede constituir un desafío.

```
void lb_gr_draw_polygon_fill_i(PICTURE_T *Pic, LINE_2D_SINT16_T *L, PIXEL_T color,  
COPYMODE_T copymode);
```

**Comprobación de verificación de pertenencia a un área poligonal.** Las siguientes funciones verifican si un punto  $P(x,y)$  se encuentra en el interior de un polígono cerrado y no auto-intersectado.

```
SINT16_T lb_gr_is_in_polygon_i(LINE_2D_SINT16_T *L, POINT_2D_SINT16_T P);  
SINT16_T lb_gr_is_in_polygon_f(LINE_2D_REAL_T *L, POINT_2D_REAL_T P);
```

**Dibujo de un vector.** Traza un vector con apariencia de flecha, apuntando desde  $(x_0, y_0)$  hacia  $(x_1, y_1)$

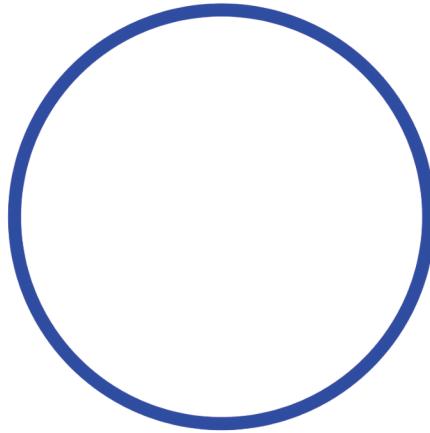
```
void lb_gr_draw_arrow(PICTURE_T *Pic, REAL_T x0, REAL_T y0, REAL_T x1, REAL_T y1,  
REAL_T w, REAL_T arrow_size, PIXEL_T color, COPYMODE_T copymode, LINEMODE_T  
linemode);
```

**Trazo de una línea circular simple.** Dibuja un círculo centrado en  $(xc, yc)$  con radio ‘radius’ y con grosor de un pixel, siguiendo el algoritmo de círculo Bresenham, con apariencia escalonada. Es la función de dibujo de círculos con mejor desempeño:

```
void lb_gr_draw_circle(PICTURE_T *Pic, SINT16_T xc, SINT16_T yc, SINT16_T radius, PIXEL_T  
color, COPYMODE_T copymode);
```

**Trazo de un círculo de alta calidad con filtro anti-escalonamiento.** Dibuja un círculo centrado en  $(xc, yc)$ , con radio ‘radius’, con grosor de línea “w” (puede tener parte fraccional).

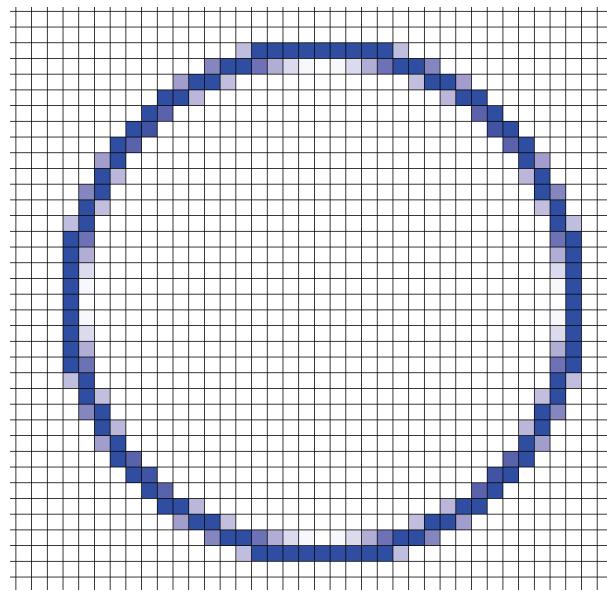
```
void lb_gr_draw_circle_antialiasing(PICTURE_T *Pic, REAL_T xc, REAL_T yc, REAL_T radius,  
REAL_T w, PIXEL_T color);
```



**Figura 31.** Representación de un círculo con grosor arbitrario y filtro anti-escalonamiento.  
[Recurso propio].

**Trazo de un círculo con filtro anti-escalonamiento, con grosor de línea igual a 2 pixeles.** Dibuja un círculo centrado en  $(xc, yc)$ , con radio ‘radius’ y con grosor de línea fijo en 2 pixeles. Esta es una versión modificada del algoritmo de círculo de Bresenham, la cual incorpora el efecto de anti-escalonamiento sin perjuicio excesivo en el desempeño.

```
void lb_gr_draw_circle_antialiasing2(PICTURE_T *Pic, SINT16_T xc, SINT16_T yc,  
SINT16_T radius, PIXEL_T color);
```



**Figura 32.** Detalle en baja resolución de un círculo con grosor de dos pixeles y filtro rápido anti-escalonamiento. [Recurso propio].

**Trazo de un círculo con filtro anti-escalonamiento, con grosor de línea igual a 3 pixeles.** Dibuja un círculo centrado en  $(xc, yc)$ , con radio “radius” y con grosor de línea fijo en 3 pixeles. Esta es una versión modificada del algoritmo de círculo de Bresenham, la cual incorpora el efecto de anti-escalonamiento sin perjuicio excesivo en el desempeño.

```
void lb_gr_draw_circle_antialiasing3(PICTURE_T *Pic, SINT16_T xc, SINT16_T yc, SINT16_T radius, PIXEL_T color);
```

**Función experimental para trazos de círculos con grosor arbitrario.** De interés solamente para estudio, pues muestra la técnica básica para lograr filtro anti-escalonamiento. Podría modificarse para otras aplicaciones.

```
void lb_gr_draw_circle_antialiasing_simple(PICTURE_T *Pic, REAL_T xc, REAL_T yc, REAL_T radius, REAL_T w, PIXEL_T color);
```

**Trazo de un segmento circular.** Trazo de un segmento circular o arco con base en un círculo centrado en  $(xc, yc)$ , radio ‘radius’, tendido entre los ángulos  $a1$  y  $a2$  en radianes.

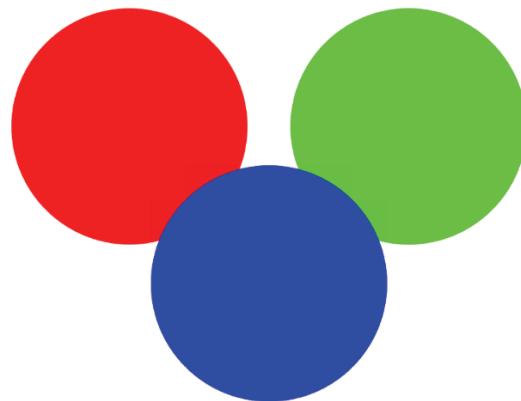
```
void lb_gr_draw_circle_arc(PICTURE_T *Pic, SINT16_T xc, SINT16_T yc, SINT16_T radius, REAL_T a1, REAL_T a2, PIXEL_T color, COPYMODE_T mode);
```

**Trazo de un círculo lleno.** Dibuja un círculo, incluyendo su interior, centrado en  $(xc, yc)$ , con radio “radius” y apariencia escalonada:

```
void lb_gr_draw_circle_filled(PICTURE_T *Pic, SINT16_T xc, SINT16_T yc, SINT16_T radius, PIXEL_T color);
```

**Trazo de un círculo lleno, con filtro anti-escalonamiento.** Dibuja un círculo, incluyendo su interior, centrado en  $(xc, yc)$ , con radio “radius” y filtro anti-escalonamiento.

```
void lb_gr_draw_circle_filled_antialiasing(PICTURE_T *Pic, SINT16_T xc, SINT16_T yc, SINT16_T radius, PIXEL_T color);
```



**Figura 33.** Representación de círculos llenos con grosor arbitrario y filtro anti-escalonamiento.  
[Recurso propio].

**Trazo de un círculo lleno, con filtro anti-escalonamiento (versión en punto flotante).** Dibuja un círculo, incluyendo su interior, centrado en  $(xc, yc)$ , con radio ‘`radius`’ y filtro anti-escalonamiento. Esta función está sujeta a comparaciones adicionales de desempeño con la versión entera y podría eliminarse.

```
void lb_gr_draw_circle_filled_antialiasing_r(PICTURE_T *Pic, REAL_T xc, REAL_T yc, REAL_T radius, PIXEL_T color);
```

**Trazo de un círculo lleno** (versión “lenta”), solo para estudio.

```
void lb_gr_draw_circle_filled_slow(PICTURE_T *Pic, SINT16_T xc, SINT16_T yc, SINT16_T radius, PIXEL_T color, COPYMODE_T copymode);
```

**Rotación de una imagen en un ángulo arbitrario:**

```
void lb_gr_bitmap_rotate(PICTURE_T *pic_src, PICTURE_T *pic_dst, REAL_T angle, PIXEL_T default_color);
```

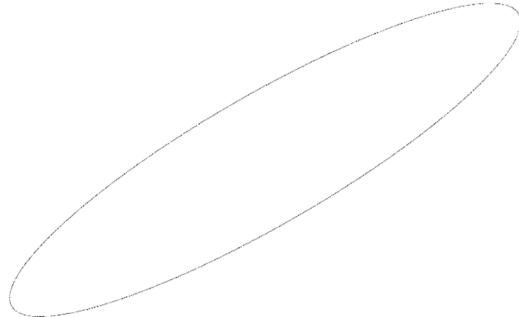
```
void lb_gr_bitmap_rotate_sampling(PICTURE_T *pic_src, PICTURE_T *pic_dst, REAL_T angle, UINT8_T n_samples, PIXEL_T default_color);
```

**Dibujo de una elipse, con apariencia escalonada.** Traza una elipse, basada en el método de Bresenham, con centro en  $(xc, yc)$ , y semiejes a y b. El ancho de la línea es un pixel y presenta escalonamiento.

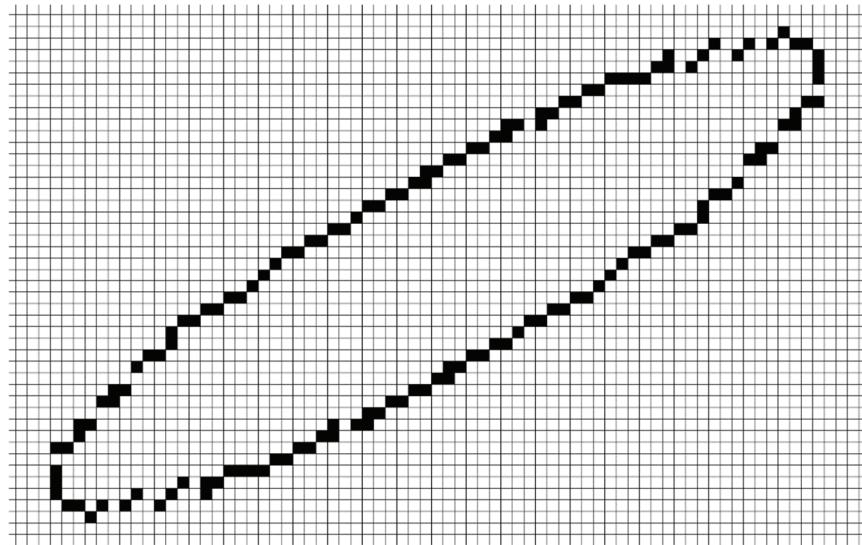
```
void lb_gr_draw_ellipse(PICTURE_T *Pic, SINT32_T xc, SINT32_T yc, SINT32_T a, SINT32_T b, PIXEL_T color, COPYMODE_T copymode);
```

**Dibujo de una elipse, rotada, con apariencia escalonada.** Traza una elipse rotada un ángulo “angle” con centro en  $(xc, yc)$ , y semiejes a y b. Es una función altamente eficiente, pero puede mejorarse para reducir su escalonamiento y discontinuidades, las cuales son notables a bajas resoluciones.

```
void lb_gr_draw_ellipse_rotated(PICTURE_T *Pic, SINT16_T xc, SINT16_T yc, SINT16_T a, SINT16_T b, REAL_T angle, PIXEL_T color, COPYMODE_T copymode);
```



**Figura 34.** Elipse rápida con rotación arbitraria. [Recurso propio].



**Figura 35.** Detalle de errores (notables en baja resolución), de la técnica trazo rápido de elipse.  
[Recurso propio].

**Dibujo de una elipse, rotada, con filtro anti-escalonamiento.** Traza una elipse rotada un ángulo ‘angle’ con centro en (xc, yc), y semiejes a y b. Aproxima la elipse mediante ‘n\_q1’ segmentos de recta por cuadrante, cada uno de ancho ‘w’.

```
void lb_gr_draw_ellipse_rotated_aspolygon (PICTURE_T *Pic, REAL_T xc, REAL_T yc, REAL_T
a, REAL_T b, REAL_T angle, REAL_T w, SINT16_T n_q1, PIXEL_T color, COPYMODE_T
copymode, LINEMODE_T linemode);
```

**Dibujo de una elipse, con centro en (xc, yc) y semiejes a y b, con filtro anti-escalonamiento y con grosor de 2 pixeles.** Es una función altamente eficiente basada en el algoritmo de Bressenham, pero modificada para efectuar un filtro básico con buen desempeño.

```
void lb_gr_draw_ellipse_antialiasing2(PICTURE_T *Pic, SINT16_T xc, SINT16_T yc,
SINT16_T a, SINT16_T b, PIXEL_T color, COPYMODE_T copymode);
```

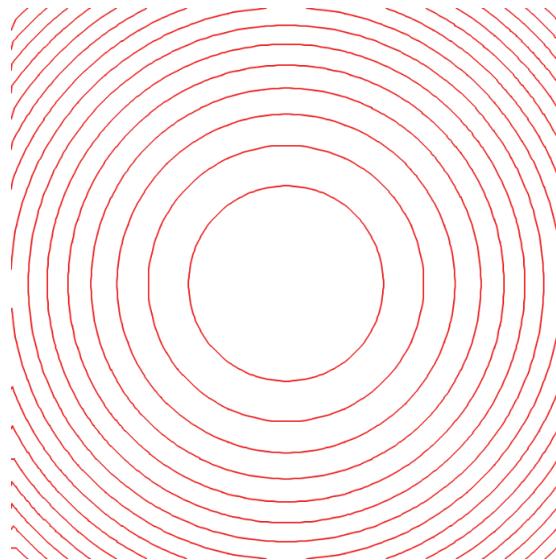
**Dibujo de una elipse, con centro en (xc, yc) y semiejes a y b, con filtro anti-escalonamiento y con grosor de 3 pixeles.** Es una función altamente eficiente basada en el algoritmo de Bressenham, pero modificada para efectuar un filtro básico con buen desempeño.

```
void lb_gr_draw_ellipse_antialiasing3(PICTURE_T *Pic, SINT16_T xc, SINT16_T yc,
SINT16_T a, SINT16_T b, PIXEL_T color, COPYMODE_T copymode);
```

**Dibujo de una función en forma implícita.** Traza en dos dimensiones la curva de nivel correspondiente a  $z=0$ , para una función  $f(x,y)$  cuya evaluación se ha previamente almacenado en una matriz M. La columna  $j=0$  corresponde a `vp2d.xr_min`; la columna  $j=m-1$  corresponde a `vp2d.xr_max`; la fila  $i=0$  corresponde a `vp2d.yr_min`; la fila  $i=n-1$  corresponde a `vp2d.yr_max`. Las líneas de la curva de nivel tienen un grosor 2, un color definido por ‘pixel’ el cual realiza una

operación de combinación con el fondo definida por ‘`copymode`’ y soporta estilos sólidos, filtrados, o solamente de puntos extremos según el parámetro ‘`linemode`’.

```
void lb_gr_implicit_2d(PICTURE_T *Pic, VIEWPORT_2D_T vp2d, MATRIX_R_T *M, REAL_T w,
PIXEL_T pixel, COPYMODE_T copymode, LINEMODE_T linemode);
```



**Figura 36.** Gráfica de los ceros correspondientes a la función  $z = \sin(x^2 + y^2)$  con  $x$  entre -5.0 y 5.0;  $y$  entre -5.0 y 5.0, obtenida mediante la función de trazo de funciones en forma implícita.  
[Recurso propio].

### Funciones de carga y almacenamiento de archivos

Obtener las dimensiones en píxeles de la imagen almacenada en un archivo BMP:

```
void lb_gr_BMPfile_getsize(const char *filename, SINT16_T *width, SINT16_T *height);
```

Carga de una imagen almacenada en un archivo BMP a un buffer PICTURE\_T:

```
void lb_gr_BMPfile_load_to_pic(const char *filename, PICTURE_T *Pic, UINT8_T alpha);
```

```
void lb_gr_BMPfile_load_to_matrix(const char *filename, MATRIX_R_T *R, MATRIX_R_T *G,
MATRIX_R_T *B);
```

Carga de la imagen almacenada en un archivo BMP en una matriz en escala de grises:

```
SINT8_T lb_gr_BMPfile_load_to_matrix_gs(const char *filename, MATRIX_R_T *P);
```

Almacenamiento de una imagen en un archivo BMP:

```
void lb_gr_BMPfile_save(const char *filename, PICTURE_T *Pic);
```

**Almacenamiento de una imagen en un archivo JPG:** (requiere la inclusión de la librería jpeglib)

```
SINT8_T lb_gr_JPGfile_save(const char *filename, PICTURE_T *Pic, UINT8_T quality);
```

### Funciones de creación y destrucción de objetos gráficos

Creación y destrucción de una línea bidimensional con coordenadas en variable entera:

```
void lb_gr_create_line2d_i(LINE_2D_SINT16_T *L);
void lb_gr_release_line2d_i(LINE_2D_SINT16_T *L);
```

Creación y destrucción de una línea bidimensional con coordenadas en variable real:

```
void lb_gr_create_line2d_r(LINE_2D_REAL_T *L);
void lb_gr_release_line2d_r(LINE_2D_REAL_T *L);
```

Creación y destrucción de una línea tridimensional con coordenadas en variable real:

```
void lb_gr_create_line3d_f(LINE_3D_REAL_T *L);
void lb_gr_release_line3d_f(LINE_3D_REAL_T *L);
```

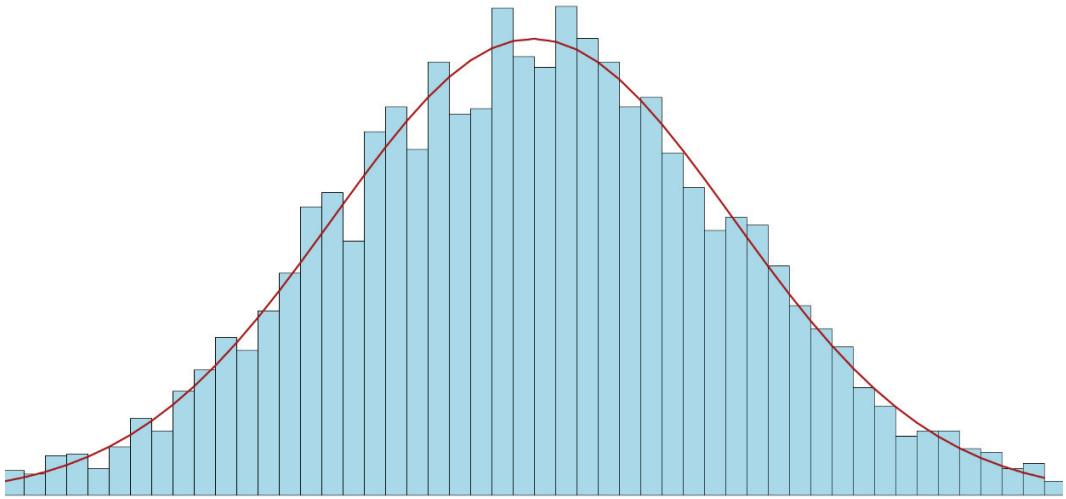
**Creación y destrucción de una imagen.** Asigna en memoria un arreglo sobre el cual se pueden elaborar, procesar y presentar imágenes, con ancho ‘w’ y altura ‘h’ especificados como miembros de Pic. La imagen es inicializada con el color `default_color`.

```
void lb_gr_create_picture(PICTURE_T *Pic, PIXEL_T default_color);
void lb_gr_release_picture(volatile PICTURE_T *Pic);
```

### Gráficas Bidimensionales tipo XY, Logarítmicas y Estadísticas

**Trazo de un histograma.** Dibuja un diagrama de barras asociado al vector ‘`bins`’ (contenedores), con base en los parámetros de escala y coordenadas de pantalla definidos por la estructura `VIEWPORT_2D_T vp2d`. A diferencia de los diagramas de barras de los productos comerciales, este simple diagrama es estricto estadísticamente y refleja la esencia de una una distribución de probabilidad. Por ejemplo, las barras son adyacentes, sin separación entre ellas y se interpretan correctamente los intervalos de cada contenedor. Obtiene la varianza  $\sigma^2$  y la media  $\mu$ , y se escala adecuadamente la curva normal para que sus áreas sean equivalentes.

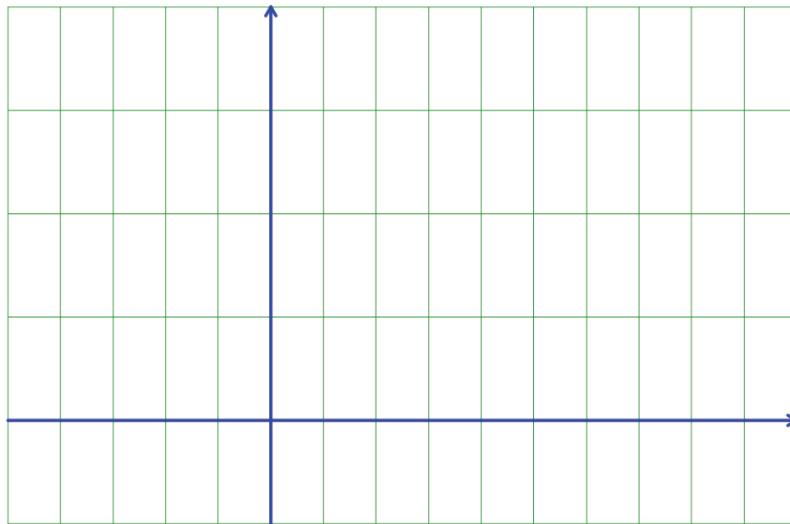
```
void lb_gr_draw_histogram(PICTURE_T *Pic, VIEWPORT_2D_T vp2d, VECTOR_R_T *bins,
PIXEL_T color_border, PIXEL_T color_background, PIXEL_T color_bar_border, PIXEL_T
color_bar);
```



**Figura 37:** Histograma de datos simulados siguiendo una distribución gaussiana con 50 contenedores y su “campana” asociada, obtenido mediante la función `lb_gr_draw_histogram()`. [Recurso propio].

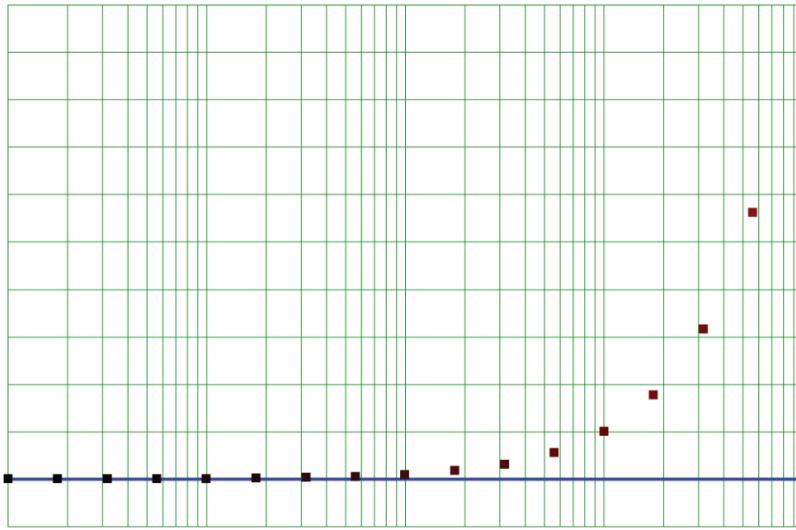
**Función de trazo de ejes coordinados.** Esta función ayuda a construir sistemas coordenados que faciliten la representación de funciones u otros tipos de información con opciones como ejes, retículas, flechas que indiquen el sentido de los ejes, y soporte de escalas logarítmicas independientes en cada eje.

```
void lg_gr_draw_axis_2d(PICTURE_T *Pic, VIEWPORT_2D_T vp2d, FONT_T *font,
PIXEL_T color_axis, REAL_T w_axis, REAL_T arrow_size,
PIXEL_T color_grid_x, REAL_T delta_grid_x,
PIXEL_T color_grid_y, REAL_T delta_grid_y, REAL_T w_grid,
UINT16_T options, COPYMODE_T copymode, LINEMODE_T linemode);
```



**Figura 38:** Trazo de los ejes de un sistema coordenado dirigido. [Recurso propio].

Se soportan opciones para el trazo de ejes logarítmicos:



**Figura 39:** Trazo de un sistema de ejes semi-logarítmico. [Recurso propio].

**Función para dibujo de ejes polares.** Esta función es provisional, pues la retícula angular (círculos) solamente se puede trazar en forma escalonada, y es deseable alcanzar el soporte de líneas con filtros anti-escalamiento.

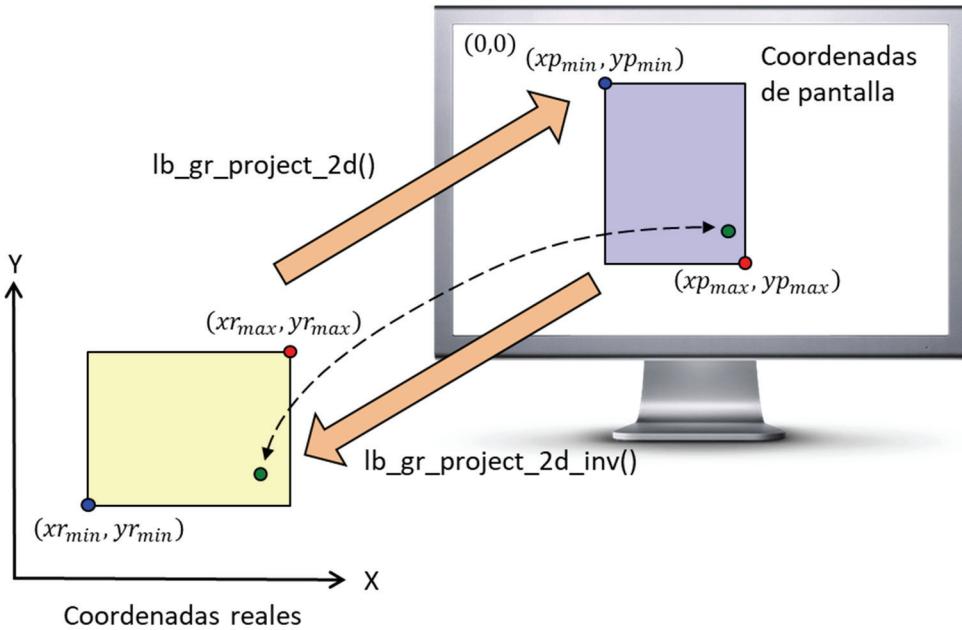
```
void lg_gr_draw_axis_2d_polar(PICTURE_T *Pic, VIEWPORT_2D_T vp2d, FONT_T *font,
    REAL_T r0, REAL_T r1, REAL_T delta_r, PIXEL_T color_r,
    REAL_T t0, REAL_T t1, REAL_T delta_t, PIXEL_T color_t,
    UINT16_T options, COPYMODE_T copymode);
```

**Función de proyección bidimensional.** Transforma las coordenadas reales ( $x_r$ ,  $y_r$ ) en las coordenadas de pantalla ( $x_p$ ,  $y_p$ ). Se proveen funciones independientes para conversión en cada eje. La estructura  $vp2d$  ('viewport') define los límites de la transformación.

```
void lb_gr_project_2d(VIEWPORT_2D_T vp2d, REAL_T xr, REAL_T yr, REAL_T *xp, REAL_T *yp);
void lb_gr_project_2d_x(VIEWPORT_2D_T vp2d, REAL_T xr, REAL_T *xp);
void lb_gr_project_2d_y(VIEWPORT_2D_T vp2d, REAL_T yr, REAL_T *yp);
```

**Función de proyección inversa.** Dadas las coordenadas de pantalla, obtiene las coordenadas reales respectivas.

```
void lb_gr_project_2d_inv(VIEWPORT_2D_T vp2d, REAL_T xp, REAL_T yp, REAL_T *xr, REAL_T *yr);
```



**Figura 40:** Relación entre coordenadas reales y de pantalla y concepto del “viewport”, [Recurso propio].

**Función de proyección en escala logarítmica.** Convierte un par de coordenadas para  $(xr, yr)$  haciendo una transformación en escala logarítmica en ambos ejes. Tanto  $xr$  como  $yr$  deben ser valores mayores o iguales a cero. Se proveen funciones para hacer transformaciones independientes en cada eje, útiles, por ejemplo, para elaborar gráficas semilogarítmicas.

```

void lb_gr_project_2d_log(VIEWPORT_2D_T vp2d, REAL_T xr, REAL_T yr, REAL_T *xp, REAL_T *yp);

void lb_gr_project_2d_x_log(VIEWPORT_2D_T vp2d, REAL_T xr, REAL_T *xp);

void lb_gr_project_2d_y_log(VIEWPORT_2D_T vp2d, REAL_T yr, REAL_T *yp);

```

**Dibujo de un punto en coordenadas reales siguiendo una ventana de transformación.** Esta función realiza internamente una transformación de coordenadas reales  $(xr, yr)$  a coordenadas de pantalla siguiendo la relación de transformación definida por `vp2d` y representa el “punto” correspondiente en la pantalla, realizando una operación de combinación con la imagen de destino según el parámetro `copymode`. El “punto” se representa mediante un pixel si el radio especificado es menor o igual a 1, o una aproximación circular de radio la cual puede ser de tipo sólida (`LINEMODE_DOTS_SOLID`) o de filtro anti- escalonamiento (`LINEMODE_DOTS_FILTERED`).

Esta función es conveniente para el trazo de gráficas técnicas, pues evita al usuario el manejo de coordenadas de pantalla.

```

void lb_gr_plot2d(PICTURE_T *Pic, VIEWPORT_2D_T vp2d, REAL_T xr, REAL_T yr, REAL_T w,
PIXEL_T color, COPYMODE_T copymode, LINEMODE_T linemode);

```

**Dibujo de una serie de puntos en coordenadas reales almacenados en las estructura L (línea) siguiendo una ventana de transformación.** Traza una serie de puntos en coordenadas reales siguiendo la relación de transformación de coordenadas definida por `vp2d`.

```
void lb_gr_plot2d_line(PICTURE_T *Pic, VIEWPORT_2D_T vp2d, LINE_2D_REAL_T *L, REAL_T w,  
PIXEL_T color, COPYMODE_T copymode, LINEMODE_T linemode);
```

**Dibujo de una línea definida por dos vectores en variable real con filtro anti-escalonamiento y grosor de trazo arbitrario.** Convierte, en forma transparente al usuario, las coordenadas reales a coordenadas de pantalla de cada uno de los puntos definidos por los vectores `Lx` y `Ly`, y traza la línea respectiva de grosor '`w`' mediante una técnica de exploración de los pixeles cercanos a la línea. Este método genera una gráfica de alta calidad a expensas de un mayor tiempo de procesamiento.

```
void lb_gr_plot2d_line_antialiasing_neighbor(PICTURE_T *Pic, VIEWPORT_2D_T vp2d,  
VECTOR_R_T *Lx, VECTOR_R_T *Ly, REAL_T w, PIXEL_T color, COPYMODE_T copymode);
```

Se presenta, además, una versión simplificada, la evalúa la totalidad de los pixeles de la imagen en lugar de hacerlo únicamente para aquellos pixeles de interés ubicados en cercanía a los segmentos de la imagen. No se recomienda el uso de este procedimiento excepto para experimentación o para su rediseño.

```
void lb_gr_plot2d_line_antialiasing_neighbor (PICTURE_T *Pic, VIEWPORT_2D_T vp2d,  
VECTOR_R_T *Lx, VECTOR_R_T *Ly, REAL_T w, PIXEL_T color, COPYMODE_T copymode);
```

#### **Función de proyección de una línea definida por pares ordenados complejos.**

Transforma una serie de puntos  $V(r,i)$  en un arreglo que contenga la proyección bidimensional de los mismos de acuerdo a la transformación de proyección definida por `vp2d` ('viewport'), interpretando la parte real como la variable `xp` y la imaginaria como la `yp`. Esta función es útil para el trazo de curvas definidas en variable compleja, tales como funciones de transferencia, señales circuitales, etc.

```
void lb_gr_project_2d_vector_c_to_line_int(VIEWPORT_2D_T vp2d, VECTOR_C_T *V,  
LINE_2D_SINT16_T *L);
```

Se define una función semejante, la cual en lugar de un arreglo de números complejos toma dos arreglos de números reales. Ambos arreglos deben tener el mismo número de elementos.

```
void lb_gr_project_2d_vector_r_to_line_int(VIEWPORT_2D_T vp2d, VECTOR_R_T *X,  
VECTOR_R_T *Y, LINE_2D_SINT16_T *L);
```

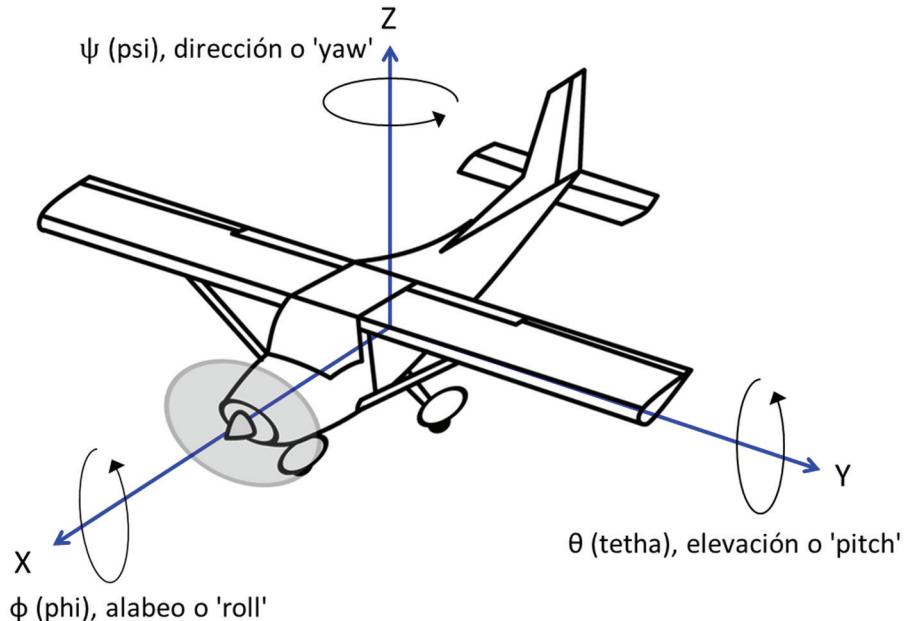
## Gráficas tridimensionales

La librería gráfica incluye algunas utilidades para la generación de gráficos tridimensionales en forma de gráficas de “marco de alambre”. Se ha elegido un esquema simplificado, con orientación basada en una sola convención de ángulos. Euler estableció que cualquier orientación deseada en tres dimensiones puede especificarse mediante 3 parámetros. Por ello, a los tres ángulos que indiquen como se efectúan las rotaciones se les llama “ángulos de Euler”. Existen 27 formas ( $3 \times 3 \times 3$ ) de efectuar rotaciones con 3 ejes, pero de ellas solamente 12 son válidas para alcanzar cualquier orientación que se desee. Estrictamente, las 12 formas de orientar un objeto son “ángulos de Euler”, pero se acostumbra separarlas en dos grupos: aquellos donde todos los ejes son diferentes se conocen como ángulos Tait-Bryan (o ángulos aeronáuticos o de navegación), y a los demás se les denominan “ángulos de Euler” o “ángulos propios de Euler”.

	Combinación de rotaciones	Válida para orientación 3D	Tait-Bryan	Euler
1	XXX	No (una sola rotación)		
2	XXY	No (rotaciones consecutivas equivalentes)		
3	XXZ	No (rotaciones consecutivas equivalentes)		
4	XYX	Si		Si
5	YYY	No (rotaciones consecutivas equivalentes)		
6	XYZ	Si	Si	
7	XZX	Si		Si
8	XZY	Si	Si	
9	XZZ	No (rotaciones consecutivas equivalentes)		
10	YXX	No (rotaciones consecutivas equivalentes)		
11	YXY	Si		Si
12	YXZ	Si	Si	
13	YYX	No (rotaciones consecutivas equivalentes)		
14	YYY	No (una sola rotación)		
15	YYZ	No (rotaciones consecutivas equivalentes)		
16	YZX	Si	Si	
17	YZY	Si		Si
18	YZZ	No (rotaciones consecutivas equivalentes)		
19	ZXX	No (rotaciones consecutivas equivalentes)		
20	ZXY	Si	Si	
21	ZXZ	Si		Si
22	→ ZYX	Si. Este es el sistema implementado.	Si	
23	ZYY	No (rotaciones consecutivas equivalentes)		
24	ZYZ	Si		Si
25	ZZX	No (rotaciones consecutivas equivalentes)		
26	ZZY	No (rotaciones consecutivas equivalentes)		
27	ZZZ	No (una sola rotación)		

**Tabla 2:** Rotaciones de orientación y tipos de ángulos [Recurso propio generado con base en la definición de los Ángulos de Euler]

En el sistema se utilizan los ángulos Tait-Bryan tipo ZYX (tipo 22 en la lista anterior), los cuales son de uso frecuente en ingeniería en áreas como la aeronáutica, la robótica, y el entretenimiento digital. Los ángulos de Euler son de mayor uso en matemáticas o en ciencias puras.



**Figura 41:** Definición de los Ángulos Tait-Bryan utilizados en el sistema. [Recurso propio, con uso de gráfico de aeronave recuperado de <http://clipart-library.com/clip-art/plane-clipart-transparent-12.htm>, de adaptación y uso libres para fines no comerciales].

**Dibujo de un sistema coordenado tridimensional.** Traza un sistema coordenado en el espacio, con opciones de retícula, etiquetas de ejes, colores y estilo de líneas:

```
void lg_gr_draw_axis_3d(PICTURE_T *Pic, VIEWPORT_3D_T vp3d,
REAL_T Rot[3][3],
FONT_T *font,
REAL_T axis_size,
REAL_T w_axis,
PIXEL_T color_axis,
REAL_T xr_min, REAL_T xr_max, REAL_T delta_grid_x,
REAL_T yr_min, REAL_T yr_max, REAL_T delta_grid_y,
REAL_T zr_min, REAL_T zr_max, REAL_T delta_grid_z,
REAL_T w_grid,
PIXEL_T color_grid,
UINT16_T options,
const char *label_o,
const char *label_x,
const char *label_y,
const char *label_z,
COPYMODE_T copymode,
LINEMODE_T linemode);
```

**Proyección de un punto tridimensional.** Proyecta un punto  $P(x,y,z)$  en variable real utilizando como parámetros la posición de la cámara, la matriz de rotación que define la orientación del objeto. Las coordenadas de pantalla son retornadas por referencia en los parámetros  $xp$  y  $yp$ . El valor  $hz$  retorna un valor positivo si el objeto está “en frente” de la cámara o negativo si está “detrás” (lo cual indica que no es visible).

```
void lb_gr_project_3d(VIEWPORT_3D_T vp3d, REAL_T Rot[3][3], POINT_3D_REAL_T P, REAL_T *xr,
REAL_T *yr, REAL_T *hz);
```

Se hace a continuación un recuento breve del procedimiento de proyección:

- Los cuerpos tridimensionales están definidos por un juego de objetos como puntos, líneas (sucesiones de puntos), superficies (sucesiones de líneas), o volúmenes (sucesiones de superficies). El asunto radica entonces, en cómo proyectar un punto de coordenadas espaciales a coordenadas de pantalla.
- Orientar el objeto (es decir, cada uno de sus puntos), efectuando el producto por la matriz de rotación  $R$ , definida como:

$$R = Rz_\psi \cdot Ry_\theta \cdot Rx_\phi$$

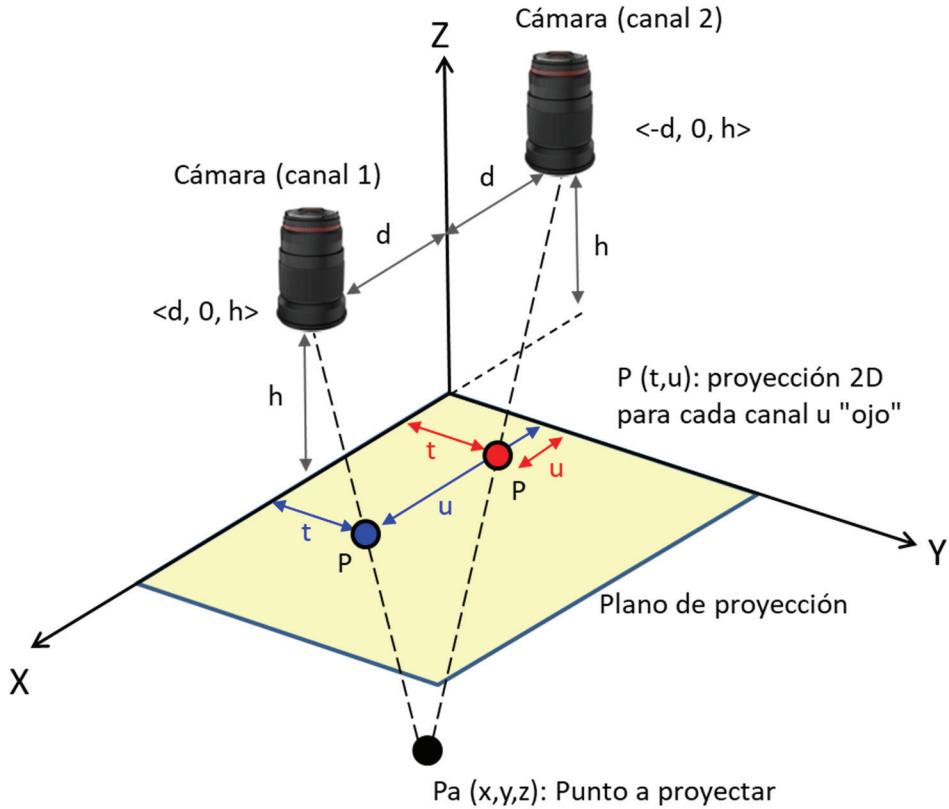
$$R = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

$$R \cdot \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}$$

- Desplazar el objeto rotado para considerar la posición  $Pc$  relativa de la cámara. La orientación de la cámara se considera fija. Por ejemplo, si el objeto se desplaza una distancia positiva en un eje determinado, su coordenada respecto al mismo eje del sistema local de la cámara disminuye.

$$Pa = R \cdot \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} - \begin{bmatrix} Pc_x \\ Pc_y \\ Pc_z \end{bmatrix}$$

- Se realiza la proyección. Los siguientes diagramas y fórmulas a continuación se basan en el sistema local de la cámara o observador.

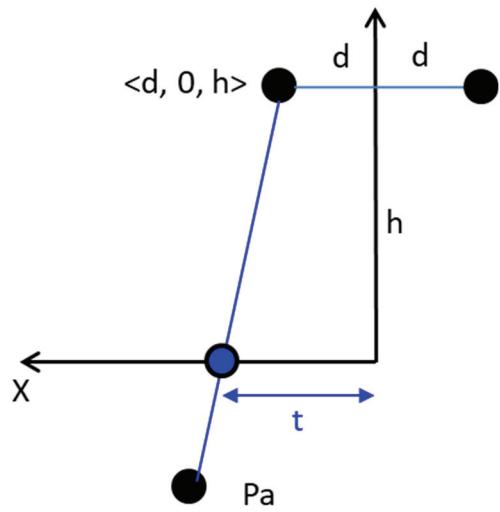


**Figura 42:** Proyección de coordenadas 3D a 2D con previsión de parámetro de separación entre cámaras (ocular) ‘d’ para estereoscopía. [Recurso propio].

La cámara tiene varios parámetros:

- **Excentricidad ‘d’ (distancia) respecto a un eje para sistemas de dos cámaras.** Permite incorporar de antemano la capacidad de generar gráficas para sistemas de doble canal (estereovisión), como aplicaciones de realidad virtual mediante técnicas estáticas dicromáticas, módulos autoestereoscópicos basados en barreras de paralaje, o métodos dinámicos como obturación electrónica de canal mediante filtros de cristal líquido.  
Para gráficas de un solo canal, se hace  $d=0$ .
- **Parámetro de perspectiva ‘h’(height).** Permite incorporar el efecto de perspectiva en la generación de gráficas. Menores valores de  $h$  aumentan la perspectiva. Un valor de ‘ $h$ ’ infinito corresponde a una proyección isométrica. El valor  $h=0$  se usa como bandera para ignorar la perspectiva y producir un efecto isométrico.

Con base en el diagrama, se pueden derivar las siguientes fórmulas de proyección para obtener las coordenadas de pantalla  $(t, u)$ , llamadas  $(xp, yp)$  en el código. El análisis resulta más claro a nivel de cada plano:

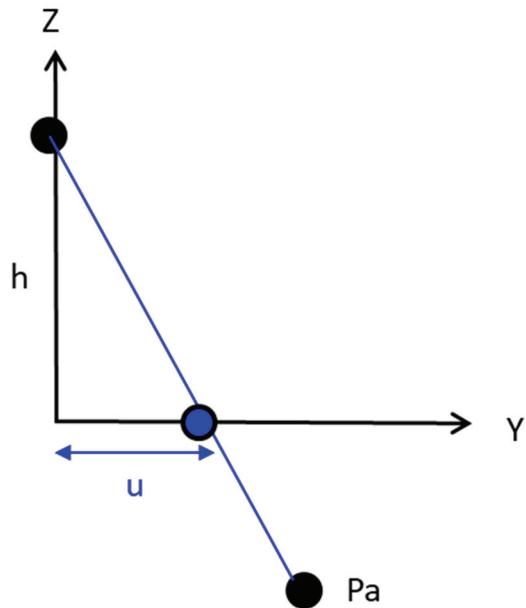


**Figura 43:** Justificación de fórmulas de proyección con separación ocular ‘d’. [Recurso propio].

$$z - h = \frac{h - Pa_z}{d - Pa_x} (x - d)$$

$$z = 0 \rightarrow x = t$$

$$t = d - h \cdot \frac{Pa_x \pm d}{Pa_z - h}$$



**Figura 44:** Justificación de fórmulas de proyección. [Recurso propio].

$$u = -h \cdot \frac{Pa_y}{Pa_z - h}$$

### Proyección y trazo de una superficie definida por una matriz de puntos tridimensionales en variable real.

```
void lb_gr_plot3d(PICTURE_T *Pic, VIEWPORT_3D_T vp3d, REAL_T Rot[3][3], POINT_3D_REAL_T
Pr, REAL_T w, PIXEL_T color, COPYMODE_T copymode, LINEMODE_T linemode);
```

Esta función proyecta un conjunto de puntos reales  $P(x,y,z)$  definidos en un arreglo bidimensional  $S$ . La posición de la cámara se define en la estructura  $vp3d$  (punto de vista o ‘viewport’. La orientación del objeto se define por la matriz de rotación  $Rot$ , la cual previamente debe haberse generado, y haber compuesto las rotaciones individuales individuales alrededor de cada eje. Es responsabilidad del usuario asegurar la integridad de la matriz de rotación. Su corrupción generaría representaciones equívocas del objeto.

Una matriz de rotación es válida si cumple las siguientes condiciones:

$$(1) \quad Rot * Rot^T = I$$

$$(2) \quad \text{Det}(Rot) = 1$$

Por ejemplo, tras ejecutar varias rotaciones sucesivas en un programa de prueba, se obtuvo la siguiente matriz de rotación:

$R[00,00] = 0.59112$	$R[00,01] = 0.76155$	$R[00,02] = -0.26576$
$R[01,00] = -0.12969$	$R[01,01] = 0.41494$	$R[01,02] = 0.90056$
$R[02,00] = 0.79609$	$R[02,01] = -0.49787$	$R[02,02] = 0.34404$

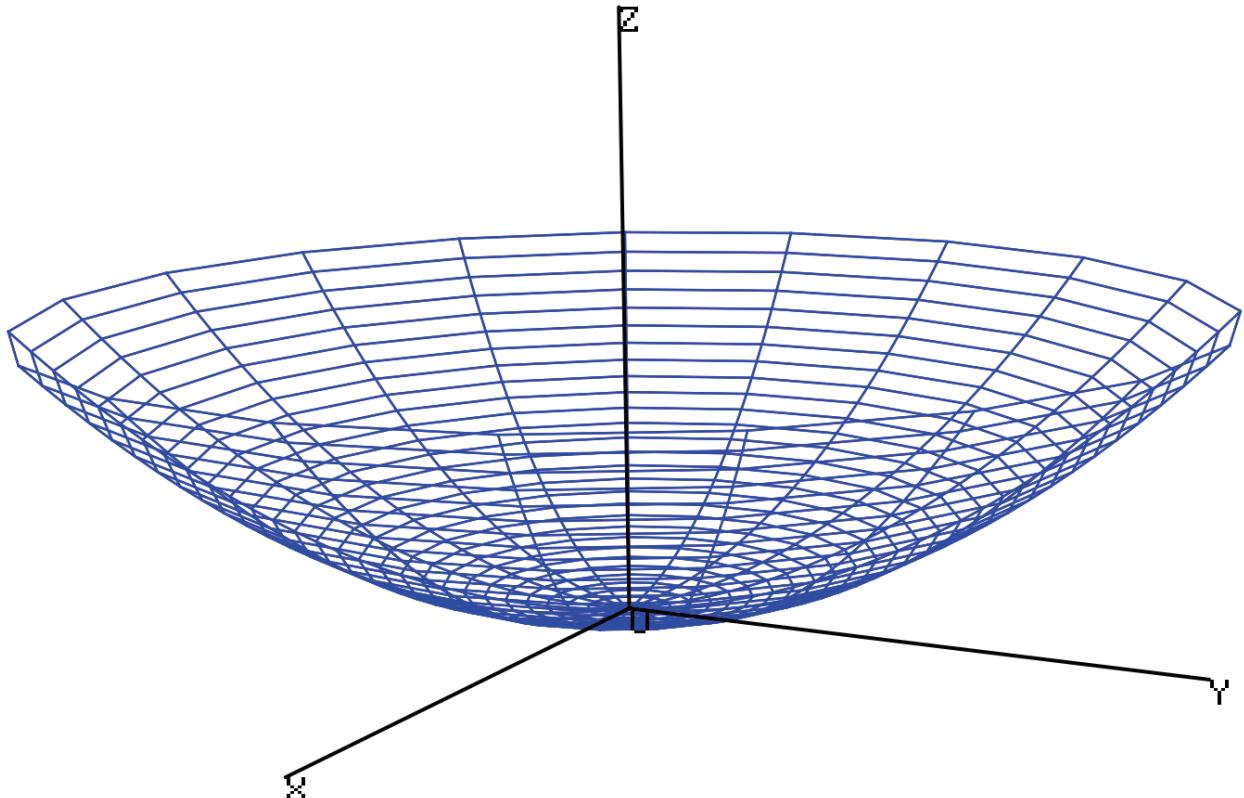
Al efectuar el producto matricial de la misma con su transpuesta se obtuvo:

$R*RT[00,00] = 1.00000$	$R*RT[00,01] = 0.00000$	$R*RT[00,02] = -0.00000$
$R*RT[01,00] = 0.00000$	$R*RT[01,01] = 1.00000$	$R*RT[01,02] = -0.00000$
$R*RT[02,00] = -0.00000$	$R*RT[02,01] = -0.00000$	$R*RT[02,02] = 1.00000$

Adicionalmente,  $\text{Det}(Rot) = 1.000000$

En la medida en que se compongan más rotaciones sucesivas, se acumularán errores numéricos y, por consiguiente, distorsión. Una matriz de rotación degenerada por errores numéricos puede sanearse mediante técnicas que la lleven a satisfacer nuevamente éstas condiciones.

```
void lb_gr_plot3d_surface(PICTURE_T *Pic, VIEWPORT_3D_T vp3d, REAL_T Rot[3][3],
MATRIX_POINT_3D_REAL_T *S, REAL_T w, PIXEL_T color, COPYMODE_T copymode, LINEMODE_T
linemode);
```



**Figura 45:** Paraboloide de revolución  $z = \frac{x^2+y^2}{5}$  entre  $r = [0.0; 5.0]$  y  $\theta = [0.0; 2 \cdot \pi]$ . [Recurso propio].

Funciones de tipo experimental que permiten trazar, pixel a pixel, una curva arbitraria definida por sus funciones paramétricas siempre y cuando sean continuas y diferenciables.

```
void lb_gr_plot_continuous_fn_2d(PICTURE_T *Pic, VIEWPORT_2D_T vp2d, FUNCTION_X
fx_t, FUNCTION_X fy_t, REAL_T t0, REAL_T t1, REAL_T delta, SINT16_T max_exp,
PIXEL_T color, COPYMODE_T copymode);

void lb_gr_plot_continuous_fn_2d_antialiasing(PICTURE_T *Pic, VIEWPORT_2D_T vp2d,
FUNCTION_X fx_t, FUNCTION_X fy_t, REAL_T t0, REAL_T t1, REAL_T delta,
SINT16_T max_exp, REAL_T w, PIXEL_T color, COPYMODE_T copymode);
```

**Conversión de una matriz de números reales a una imagen.** Convierte los valores almacenados en una matriz a una imagen en escala de grises. La matriz y la imagen pueden diferir en tamaño, pero se cargan en la imagen únicamente las celdas que no excedan el número de columnas y de filas de la imagen. Los datos se interpretan entre 0.0 y 1.0, los cuales corresponden al máximo color por canal según el número de bits definido en tiempo de compilación para la estructura `PICTURE_T`. Los valores negativos se convierten a cero, y los que igualen o excedan 1.0 se truncan al máximo valor de color por canal, por ejemplo, 0xFF para 8 bits.

```
SINT8_T lb_gr_matrix_gs_to_pic(MATRIX_R_T *A, PICTURE_T *Pic);
```

**Despliegue de imagen.** Transfiere el contenido de una imagen a un framebuffer compatible con SDL, con opciones de operación de combinación con el destino según el parámetro COPYMODE y opciones de despliegue como escalamiento entero, borde simple o doble (de color arbitrario de 12 bits) entre pixeles.

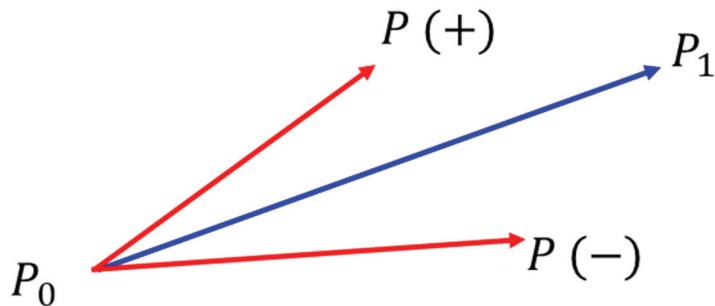
```
void lb_gr_render_picture(PICTURE_T *Pic, SINT16_T x, SINT16_T y, COPYMODE_T copymode,
RENDERMODE_T rendermode);
```

**Conversión a color con base en un entero decimal.** Convierte el dígito decimal menos significativo de un número signado de 8 bits en el color correspondiente definido por el estándar RS-279 de la Alianza de Industrias Electrónicas (EIA, Electronic Industries Alliance), conocido informalmente como el código de colores de resistencias.

```
PIXEL_T lb_gr_value_to_color(SINT8_T value);
```

**Comprobación si un punto está a la “izquierda” de un vector  $\overrightarrow{P_0P_1}$**  Retorna 1 si un punto P está a la “izquierda” (rotación en sentido contra-horario) del vector dirigido de P0 a P1, siguiendo la regla de la mano derecha. En caso contrario, retorna 0 si está sobre la línea asociada al vector, o -1 si está a la derecha. Este tipo de función es de interés para ciertos problemas geométricos como la determinación de pertenencia de un punto al interior de un polígono cerrado. Se proveen versiones enteras y de punto flotante.

```
SINT16_T lb_gr_check_left_i(POINT_2D_SINT16_T P0, POINT_2D_SINT16_T P1, POINT_2D_SINT16_T P);
```



**Figura 46:** Orientación (izquierda o derecha) de puntos P respecto a un vector  $\overrightarrow{P_0P_1}$ . [Recurso propio].

La función complementaria verifica si el punto está “a la derecha” (rotación en sentido horario):

```
SINT16_T lb_gr_check_left_r(POINT_2D_REAL_T P0, POINT_2D_REAL_T P1, POINT_2D_REAL_T P);
```

## Operaciones gráficas con aplicación del algoritmo z-buffer para ocultamiento de elementos no visibles

Este grupo de funciones es experimental. Constituyen la base para el desarrollo de nuevas primitivas tridimensionales con ocultamiento de elementos no visibles:

### Creación de un z-buffer:

```
void lb_gr_create_zbuffer(PICTURE_T *Pic, MATRIX_R_T *Z);
```

### Re-inicialización del z-buffer:

```
void lb_gr_reset_zbuffer(MATRIX_R_T *Z);
```

### Trazo de una línea tridimensional con escritura de los valores de distancia en el z-buffer:

```
void lb_gr_plot_zbuffer_line_1(PICTURE_T *Pic, VIEWPORT_3D_T vp3d, REAL_T Rot[3][3],  
MATRIX_R_T *Z, POINT_3D_REAL_T P0, POINT_3D_REAL_T P1, PIXEL_T color, COPYMODE_T  
copymode);
```

### Trazo de un “punto” tridimensional con escritura de su valor de distancia en el z-buffer:

```
void lb_gr_plot_zbuffer_dot(PICTURE_T *Pic, VIEWPORT_3D_T vp3d, REAL_T Rot[3][3],  
MATRIX_R_T *Z, POINT_3D_REAL_T P, PIXEL_T color, COPYMODE_T copymode);
```

### Trazo de un triángulo tridimensional con escritura de su valor de distancia en el z-buffer:

```
void lb_gr_plot_zbuffer_triangle(PICTURE_T *Pic, VIEWPORT_3D_T vp3d, REAL_T Rot[3][3],  
MATRIX_R_T *Z, POINT_3D_REAL_T PA, POINT_3D_REAL_T PB, POINT_3D_REAL_T PC,  
PIXEL_T color, COPYMODE_T copymode);
```

### Trazo de un pixel de coordenadas (x,y), calculando la distancia z-buffer como si dicho punto perteneciera a un triángulo tridimensional con coordenadas A(xa, ya, hza), B(xb, yb, hzb) y C(xc, yc, hzc).

```
void lb_gr_plot_zbuffer_pixel(PICTURE_T *Pic, MATRIX_R_T *Z, REAL_T xp, REAL_T yp,  
REAL_T PA_x, REAL_T PA_y, REAL_T PB_x, REAL_T PB_y, REAL_T PC_x, REAL_T PC_y,  
REAL_T PA_hz, REAL_T PB_hz, REAL_T PC_hz, PIXEL_T color, COPYMODE_T copymode);
```

## 15. Librerías de evaluación funcional “lb\_parser.c”

En el desarrollo de aplicaciones para ciencias e ingeniería, con frecuencia surge la necesidad de ingresar y procesar expresiones en notación matemática. El Lenguaje C no ofrece esta característica, y para el desarrollo de programas sencillos el usuario normalmente codifica en fijo la expresión y hace cargo de la manipulación de errores de rango, dominio y otros que se tengan. Por esta razón, se desarrolló como parte del sistema una librería que permite el procesamiento de funciones, basada en el algoritmo de “Despacho de Trenes o “Shunting Yard” de Edsger Dijkstra [16]

Se planteó un esquema compuesto por dos partes:

- a) Preprocesamiento de la expresión: esta parte genera una lista ordenada de constantes, variables, operadores y funciones en el orden en que deben ser aplicados para obtener el resultado. Esta lista sigue, fundamentalmente, la Notación Polaca Reversa (RPN, Reverse Polish Notation).
- b) En forma separada, se realiza la evaluación de la lista.

El uso de RPN ofrece ventajas como bajo uso de memoria y alta velocidad de procesamiento.

En la implementación final se lograron incorporar algunas características que se consideraron convenientes:

- a) Los operadores no se encuentran codificados en fijo o “hard coded” sino que el preprocesador comprueba la pertenencia de un símbolo que haya sido encontrado (por ejemplo, ‘+’ o ‘\*’) a una tabla de operadores, la cual define para cada uno de ellos su orden de prioridad (por ejemplo, la potencia ‘^’ tiene mayor precedencia que la multiplicación ‘\*’); asociatividad (de izquierda a derecha como la suma o de derecha a izquierda, como la potencia); y número de argumentos (operadores unarios o binarios).

Ahora bien, la ventaja de la definición de las reglas de evaluación mediante una tabla es que permite redefinirlas, adicionar operadores, o cambiar el comportamiento sin reescribir el preprocesador.

Sobre este punto, se presenta la definición actual de la tabla de operadores:

```
OPERATOR_T ops[MAX_OPERATORS] =  
{ { op_positive,           "+",  a_right, 5, op_unary },  
  { op_negative,          "-",  a_right, 5, op_unary },  
  { op_factorial,         "!",   a_right, 6, op_unary },  
  { op_power,              "^",  a_right, 5, op_binary },  
  { op_division,           "/",  a_left,   4, op_binary },  
  { op_product,            "*",  a_left,   4, op_binary },  
  { op_subtraction,        "-",  a_left,   3, op_binary },
```

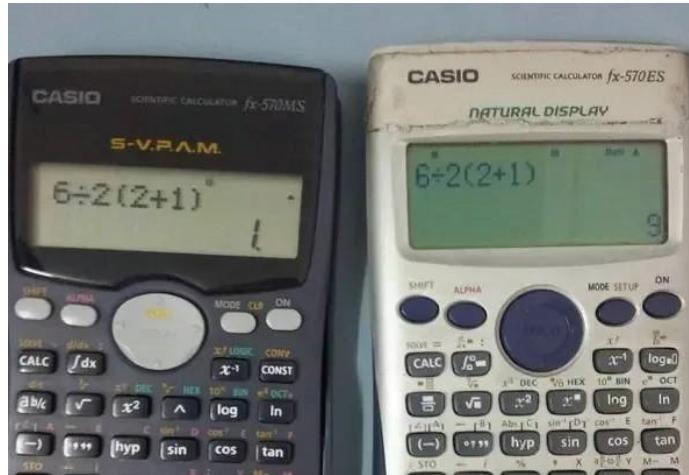
```

{ op_addition,           "+", a_left, 3, op_binary },
{ op_equal_than,         "=", a_left, 2, op_binary },
{ op_larger_than,        ">", a_left, 2, op_binary },
{ op_smaller_than,      "<", a_left, 2, op_binary },
{ op_different_than,    "!=" , a_left, 2, op_binary },
{ op_larger_or_equal_than, ">=", a_left, 2, op_binary },
{ op_smaller_or_equal_than, "<=", a_left, 2, op_binary } };

```

Se aclara que se evitó el soporte de la multiplicación implícita, por dos razones:

- En primer lugar, si se tiene una variable definida como ‘g’, la expresión ‘g(45.2)’ podría parecer al usuario como una función ‘g’ que es evaluada en el valor 45.2, en lugar del producto de la variable ‘g’ con la constante 45.2.
- La multiplicación implícita parece generar en algunos usuarios y desarrolladores confusión sobre las reglas para la evaluación de operadores, cuyo ejemplo más notorio (de amplia difusión como broma entre ingenieros) es la conocida discrepancia entre dos calculadoras Casio de casi el mismo modelo (y muchas otras):



**Figura 47:** Ambigüedad aparente por mala interpretación de las reglas PEMDAS (acrónimo en inglés para parenthesis, exponentes, multiplicación, división, adición y substracción) para expresiones con multiplicación implícita por yuxtaposición. Aguirre, A. (2019). *One Math Problem But Two Different Answers*. [Figura]. Recuperado de

<https://gineersnow.com/students/calculators-different-answers-one-math-problem>

- El preprocesador es el mismo tanto para expresiones reales como para expresiones complejas, esquema que se prefirió para evitar duplicidad de código, siendo las reglas muy similares.
- Existen dos versiones de las funciones evaluadores de la lista RPN: `1b_pa_eval_real()` y `1b_pa_eval_complex()`. Cada una realiza la evaluación en el tipo de datos respectivo, haciendo llamadas a diferentes tipos de funciones según se requiera. Por ejemplo, el evaluador real de la lista RPN llama internamente la función estándar seno ('sin') de C definida en `<math.h>`; mientras que el evaluador complejo llama a la función `1b_cp_sin()`, escrita como

parte del proyecto. Se aclara que la librería estándar `<complex.h>` define varias de estas funciones, pero se prefirió no utilizarla para alcanzar un mejor manejo de condiciones de error.

## Funciones de evaluación funcional

**Preprocesamiento inicial.** Transforma la expresión literal almacenada en `*fnslt` en una lista `*fnrecord` considerando únicamente que la expresión contenga símbolos que estén definidos como constantes, variables, operadores, funciones, separadores y paréntesis. Ignora deliberadamente si la expresión tiene sentido matemático o si contiene errores. Una expresión como ‘`sin(,cos,((`’ es válida, dado que todos sus símbolos están definidos. Otra expresión como ‘`xyz(2)`’ reporta una condición de error debido a que ‘`xyz`’ es una función que no está definida.

```
void lb_pa_parse(char *fnstr, FN_RECORD_T *fnrec, char *vars, SINT16_T *n_vars)
```

**Aplicación de reglas algebraicas y verificación de conformidad.** Esta función toma la lista de operaciones y la ordena siguiendo las reglas definidas en las tablas de operadores y de funciones, retornando una lista ordenada que puede ser evaluada.

```
void lb_pa_shunting_yard(FN_RECORD_T *fnrec)
```

Ejemplo: sea la función definida como:  $\sin \left[ 1 + y \cdot \cos \left( 3 + \frac{\pi}{4} \right) \right]$

- Se define como variable el carácter 'y' dentro del arreglo de variables válidas `vars`
  - La expresión, en texto, se escribe como '`sin(1+y*cos(3+pi/4))`'

La función auxiliar `lb pa print output()` imprime el estado de la lista de la evaluación.

Al efectuar el preprocessamiento con `lb.parse()`, se interpreta la expresión como:

```
item 0 of 14 = FUNCTION[0]=SIN, a0=0, a1=0, a2=0, a3=0, a4=0
item 1 of 14 = LPAR
item 2 of 14 = CONSTANT=1.000000
item 3 of 14 = OPERATOR[7]=[+] [B]
item 4 of 14 = VARIABLE[1]
item 5 of 14 = OPERATOR[5]=[*] [B]
item 6 of 14 = FUNCTION[1]=COS, a0=38, a1=70, a2=70, a3=127, a4=0
item 7 of 14 = LPAR
item 8 of 14 = CONSTANT=3.000000
item 9 of 14 = OPERATOR[7]=[+] [B]
item 10 of 14 = CONSTANT=3.141593
item 11 of 14 = OPERATOR[4]=[/] [B]
item 12 of 14 = CONSTANT=4.000000
item 13 of 14 = RPAR
item 14 of 14 = RPAR
```

Una vez aplicadas las reglas matemáticas con el procedimiento `lb_pa_shunting_yard()`, la lista se reduce a:

```
item 0 of 10 = CONSTANT=1.000000
item 1 of 10 = VARIABLE[1]
item 2 of 10 = CONSTANT=3.000000
item 3 of 10 = CONSTANT=3.141593
item 4 of 10 = CONSTANT=4.000000
item 5 of 10 = OPERATOR[4]=[/] [B]
item 6 of 10 = OPERATOR[7]=[+] [B]
item 7 of 10 = FUNCTION[1]=COS, a0=5, a1=0, a2=0, a3=0, a4=0
item 8 of 10 = OPERATOR[5]=[*] [B]
item 9 of 10 = OPERATOR[7]=[+] [B]
item 10 of 10 = FUNCTION[0]=SIN, a0=10, a1=0, a2=0, a3=0, a4=0
```

**Evaluación de una lista ordenada.** Para evaluar la lista, la función evaluadora inicia un barrido de la pila, iniciando por el primer elemento, hasta que encuentra un operador o una función. Dependiendo del número de argumentos definido para dicho operador (o función), según la tabla de operadores (o según la tabla de funciones), se colapsan dichos argumentos (a esto se le llama hacer un “pop”) y se expande en uno el resultado (esto es hacer un “push”).

Un tema avanzado abordado con éxito en el proyecto es el manejo de la pila para funciones de un número indefinido de argumentos, como por ejemplo, la función `max(x0, x1, x2..., xn)`.

```
REAL_T lb_pa_eval_real(FN_RECORD_T *fnrec, REAL_T *values, MATHERROR_T *error);
COMPLEX_T lb_pa_eval_complex(FN_RECORD_T *fnrec, COMPLEX_T *values, MATHERROR_T *error);
```

Se devuelve por referencia un indicador de validez de la evaluación, a través del parámetro `*error`.

La librería provee un juego de funciones definidas, el cual puede ser expandido según se requiera:

Funciones incorporadas al Evaluador de Expresiones	
SIN	Seno en variable real o compleja
COS	Coseno en variable real o compleja
TAN	Tangente en variable real o compleja
ASIN	Seno inverso en variable real o compleja
ACOS	Coseno inverso en variable real o compleja
ATAN	Tangente inversa en variable real o compleja
SINH	Seno hiperbólico en variable real o compleja
COSH	Coseno hiperbólico en variable real o compleja
TANH	Tangente hiperbólica en variable real o compleja
EXP	Exponencial en variable real o compleja
LN	Logaritmo natural en variable real o compleja
LOG	Logaritmo decimal en variable real o compleja
LOG2	Logaritmo en base 2 en variable real o compleja

LOGB	Logaritmo en base b en variable real o compleja
MAX2	Para variable real: retorna el máximo entre dos números. Para variable compleja: retorna el número con mayor norma.
MIN2	Para variable real: retorna el mínimo entre dos números. Para variable compleja: retorna el número con menor norma.
IF	Función condicional
SQR	Cuadrado de un número real o complejo
SQRT	Raíz cuadrada de un número real o complejo
ABS	Valor absoluto de un número real o magnitud o norma de un número complejo
ERF	Función de error gaussiano. El argumento debe ser un número real.
ERFC	Función de error gaussiano complementario. El argumento debe ser un número real.
RAND	Número pseudo-aleatorio equiprobable entre 0.0 y 1.0
MARSA	Número aleatorio gaussiano generado con un valor definido de varianza según el método de Marsaglia.
DSTD	Área bajo la campana de Gauss
REAL	Parte real de un número complejo. Arroja un resultado de error si se llama bajo <code>lb_pa_real()</code>
IMAG	Parte imaginaria de un número complejo. Arroja un resultado de error si se llama bajo <code>lb_pa_real()</code>
POLAR	Convierte a forma polar un número complejo en forma rectangular.
RECT	Convierte a forma rectangular un número complejo en forma polar.
AVG	Promedio aritmético de un número arbitrario de parámetros.
IDEF	Integral definida de una función de variable real o compleja, evaluada por el método de Simpson.
IDEF2	Integral doble definida de una función de variable real o compleja, evaluada por el método de Simpson.
SIGMA	Sumatoria de una función de variable real o compleja.
PROD	Productoria de una función de variable real o compleja.
DIFF	Derivada en un punto de una función de variable real o compleja.
DIFF2	Derivada de una función de dos variables en variable real o compleja.
PI	Constante: $\pi = 3.14159\dots$
E	Constante $e = 2.71828\dots$

Ejemplo: Al evaluar (proceso que toma alrededor de 60 minutos en un computador de escritorio convencional):

$$4 \sum_{i=1}^{10^{10}} \frac{(-1)^{i+1}}{2 \cdot i - 1}$$

```
'4*sigma(1,10^10,(-1)^(i+1)/(2*i-1),i,1)'
```

Se obtiene: 3.141592653. Esta serie, sin embargo, no es la más adecuada para obtener aproximaciones de  $\pi$ . En la actualidad el estado del Arte está dado por métodos de rápida convergencia como el algoritmo de Chudnovsky [17]:

$$\frac{1}{\pi} = \frac{1}{426880 \sqrt{10005}} \sum_{i=0}^{\infty} \frac{(6i)! (13591409 + 545140134i)}{(3i)! (i!)^3 (-640320)^{3i}}$$

## **16. Extensiones, mejoras y proyectos asociados:**

El sistema está sujeto a cambios, adiciones y mejoras. Un sistema de propósito general como el propuesto está sujeto a la profundización en cualquier área que se deseé. Se discuten a continuación algunos temas que se desean explorar:

### **16.1. Diseño y elaboración de un módulo para uso simplificado del sonido.**

Así como el módulo gráfico simplifica al usuario la creación y control de gráficos en forma independiente del dispositivo, conviene explorar el desarrollo de un módulo similar para el sonido. El manejo del sonido, sin embargo, impone demandas para garantizar la continuidad del mismo y requiere de un manejo cuidadoso de los eventos y de la temporización. El desafío consiste en simplificar el manejo en una forma eficiente para la síntesis en tiempo real de audio en una forma que sea simple para el usuario.

Hasta el momento, se ha hecho una exploración inicial del tema con el sistema de audio de SDL (Simple DirectMedia Layer), logrando síntesis en tiempo real de audio y se elaboró un ejemplo que genera tonos de señalización DTMF (Dual Tone Multi Frequency). Se considera que el método está sujeto a muchas mejoras.

Algunas aplicaciones de un módulo de este tipo pueden ser:

- Detección de eventos sonoros.
- Medición de distancias con base en procesamiento de audio.
- Desarrollo de equipos geosísmicos portátiles de bajo costo.
- Estimación de condiciones de flujo, viscosidad o contaminación en tuberías industriales o de acueducto o alcantarillado.
- Detección de anomalías en máquinas sin interrumpir su operación.
- Elaboración de instrumentos como generadores de señales, medidores de capacidad auditiva para fonoaudiología, marcadores automáticos para plantas o sistemas que hagan uso de señalización en banda de audio.
- Elaboración de maquetas didácticas para parques científicos o para centros de enseñanza.

### **16.2. Soporte de interrupciones asociadas al GPIO en sistemas Raspberry Pi**

El manejo de interrupciones bajo Linux, y la configuración de dispositivos que las generen es un tema que requiere profundización. El módulo GPIO es funcional para entrada y salida, con lectura recurrente de entradas o “polling”. Los foros y publicaciones especializadas [12] sugieren que no existe un mecanismo directo para el manejo de interrupciones en modo de usuario. El mismo FAIRHEAD [12] sugiere la extensión de las funciones `sysfs` (el cual es un pseudo sistema

de archivos para el intercambio de información del hardware) como método para implementar interrupciones. Este tema es avanzado [15] y se espera explorarlo en el futuro. Es de particular interés, en caso de funcionar, el tiempo de respuesta que se pueda obtener para la detección y despacho de una interrupción ¿es del orden de décimas, centésimas o milésimas de segundo? Esta pregunta se deja abierta por el momento.

### **16.3. Mejoras al sistema gráfico**

Las gráficas por computadora constituyen un campo extenso. Algunas mejoras de interés son las siguientes:

- 1. Soporte de fuentes de texto tipo vectorial:**

Una característica deseable es el soporte de fuentes vectoriales convencionales tipo Opentype o TrueType. En la actualidad se soportan fuentes de tipo fijo.

- 2. Desarrollo de las técnicas para ocultamiento de elementos no visibles en gráficas 3D:**

En la actualidad, los gráficos 3D son de tipo vectorial o de “marco de alambre”. Las técnicas de ocultamiento de elementos no visibles se encuentran en estado experimental, haciendo uso del algoritmo z-buffer aplicado a triángulos tridimensionales. Por ejemplo, si se definen dos o más triángulos en una región del espacio y se define un punto de vista (posición y orientación) de la cámara, el método bloquea las partes no visibles de los objetos. El triángulo es el elemento básico para la construcción de objetos complejos definidos o aproximados como poliedros.

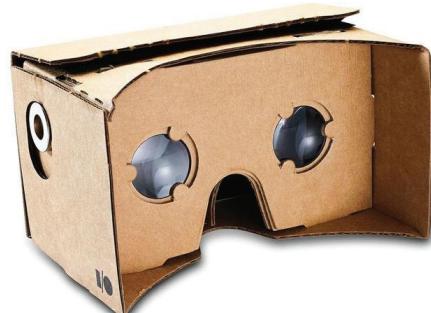
### **16.4. Proyecto asociado: Montaje de un sistema de realidad virtual de bajo costo.**

Un proyecto derivado es el desarrollo de un sistema de realidad virtual de bajo costo. La nueva tarjeta Raspberry Pi 4, de reciente liberación, facilita este emprendimiento pues posee dos salidas HDMI (High-Definition Multimedia Interface), una de las cuales podría utilizarse para la llamada al programa o para la visualización por parte de terceros, mientras que la segunda podría conectarse a un módulo HDMI de 5 pulgadas a ser instalado en un visor de realidad virtual de bajo costo con modificaciones. Existen opciones ya terminadas, por ejemplo, el “Utopia 360” (de costo aproximado US\$ 25) pero también puede construirse uno similar a las cajitas “Google Cardboard”. Una opción para la pantalla es el módulo con número de parte HTT50A-TPC-BLM-B0-H6-CH-V5, fabricado por Matrix Orbital, y disponible en Digikey bajo la referencia 635-1181-ND). Este tiene un valor de aproximadamente US \$75, pero existen módulos similares en venta en Alibaba y en eBay, con características en principio similares y de menor costo. Un sistema de este tipo podría incorporar acelerómetros para detección de

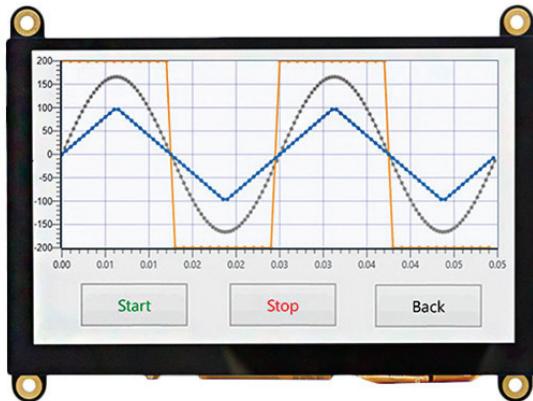
orientación y movimiento, los cuales pueden ser instalados en el visor y leídos a través del GPIO.



**Figura 48:** Dispositivo de Realidad Virtual de bajo costo. R3Trak. (2019). *Catálogo en línea de dispositivos de Realidad Virtual*. [Figura]. Recuperado de <http://r3trak.com/vr/manual.aspx>



**Figura 49:** Dispositivo de Realidad Virtual tipo “Google Cardboard”. Popular Science Staff. (2014). *Google Cardboard Is Virtual Reality On A Budget*. [Figura]. Recuperado de <https://www.popsci.com/google-cardboard/>



**Figura 50:** Módulo LCD de 5 pulgadas fabricado por Matrix Orbital, adecuado para un sistema de Realidad Virtual de bajo costo. Matrix Orbital. (2019). *Catálogo en línea*. [Figura].

Recuperado de <https://www.matrixorbital.com/communication-protocol/hdmi/htt50a>

Un aparato de este tipo puede ser aplicable para enriquecer la enseñanza de temas como el cálculo, la geometría analítica o los cursos de física. Otros usos son la elaboración de maquetas para centros de ciencia como el Parque Explora, o también el montaje de proyectos de entretenimiento digital.

### 16.5. Proyecto asociado: Desarrollo de hardware modular para instrumentación industrial.

Una idea para extender las capacidades del sistema (ejecutado sobre una tarjeta Raspberry Pi) dentro de un contexto industrial, es el desarrollo de tarjetas impresas miniatura que incorporen bloques de diseño como, por ejemplo:

- Aislamiento de entradas y salidas mediante opto-acopladores.
- Módulos que incorporen (o permitan conectar) fuentes de poder “limpias” (estables, con las protecciones del caso, y de bajo ruido), puesto que alimentar elementos de precisión como conversores Analogo-Digitales a partir de los buses digitales de 5.0V y 3.3V provistos por el GPIO no es una buena práctica, puesto que los buses digitales no son adecuados para la alimentación de dispositivos analógicos. Igualmente, se debe evitar que condiciones como el cambio en los niveles, ruido, o cortocircuito en el bus análogo afecten el bus digital.
- Módulos para el manejo de cargas “sucias” como relevadores electromecánicos o motores.
- Transceivers con interfaces de datos como RS-232 y RS-485.
- Tarjetas simples de adquisición con conversor análogo-digital de varios canales y resolución de 12 o superior.
- Tarjetas de salida análoga con resolución típica de 10 a 12 bits.

## 16.6. Proyecto Asociado: Desarrollo de un sistema magnificador para personas con limitación visual.

Una de las preguntas que se puede formular ante proyectos de software de este tipo es cuál es su aplicabilidad a la solución de problemas en el contexto local, o qué beneficio pueden ofrecer a la comunidad, al país, o a la humanidad. Se discute este punto con un ejemplo concreto:

Existen ciertas enfermedades, como la degeneración macular, las cataratas y el glaucoma, que reducen la capacidad visual de quienes las padecen. Otras condiciones pueden deberse a accidentes, como el daño a la retina por exposición a radiaciones intensas, tumores retinales, defectos congénitos, y otras. Estos padecimientos reducen la capacidad visual de las personas en diferentes grados y de diferentes maneras. Por ejemplo, la degeneración macular difumina (hace borrosa) el área central pero no el campo visual periférico; el glaucoma reduce el campo visual; las cataratas, causan opacidad; etc. Según la severidad de la condición, el paciente puede requerir de ayudas visuales para realizar actividades corrientes como leer un documento.

Existen equipos denominados “magnificadores de video” o simplemente “magnificadores” que pueden facilitar la visión a personas con limitaciones visuales. En el contexto de Colombia no son muy conocidos (incluso, sorprendentemente, por personal de la salud especializado en el tema). En Norteamérica, tampoco son de fácil adquisición, pues deben ordenarse de empresas especializadas y tienen un alto costo, típicamente de US\$ 2,000 a US\$ 3,500, el cual en el contexto colombiano está por fuera del alcance de una gran parte de la población, especialmente tras considerar gastos de despacho, nacionalización, e impuestos.



**Figura 51:** Magnificador comercial para personas con discapacidad visual. The Low Vision Store. (2019). *Catálogo en línea*. [Figura]. Recuperado de <https://shoplvs.com/20-merlin-ultra-full-hd-video-magnifier/>

Desde el punto de vista del diseño electrónico, un magnificador consta de los siguientes subsistemas:

- a) Un dispositivo visualizador, el cual es simplemente un monitor LCD o LED.
- b) Un módulo basado en microporcesador o sistema embebido.
- c) Una cámara de alta calidad con la distancia focal adecuada.
- d) Un sistema de iluminación y control de luminosidad.
- e) Un módulo de interfaz de usuario para operación mediante botones o perillas grandes y visibles.
- f) Software que permita la adquisición de la imagen, el despliegue de la misma, y que implemente filtros digitales tales como: aumento artificial del contraste; cambio de esquema de colores; efecto de inversión de texto (negativo); correcciones geométricas (distorsión con curvatura), y otros.

Un sistema de este tipo, se propone, puede construirse utilizando un monitor convencional, una tarjeta Raspberry Pi, una cámara dedicada, y un circuito electrónico básico a cargo de la iluminación y de los controles de usuario. El software asociado, en su mayor parte, puede elaborarse a partir de las librerías gráficas, matriciales, y del GPIO ya escritas. Se estima que un magnificador de este tipo, diseñado y construido localmente podría tener un costo en el rango de US\$ 200 a US\$ 750, dependiendo principalmente del tipo de cámara que se elija. Este proyecto está, a la fecha, a nivel de evaluación de las cámaras y otros componentes. Por ejemplo, se adquirieron dos pares de cámaras y su juego de lentes, de los cuales ya se ha descartado uno de ellos.



**Figura 52:** Cámara de bajo costo (US\$ 50), ref. ELP-USBFHD06H-MFV, en evaluación para uso en proyecto de magnificador visual. Posee un sistema varifocal, ajustable entre 5 y 50 mm. Ofrece una resolución de 1920 x 1080 y opera en condiciones de baja iluminación. [Recurso propio. Diseño industrial de la cámara pertenece a Ailipu Technology Company.]

Un sistema de este tipo puede mejorar la calidad de vida de personas con discapacidad visual parcial. Su provisión y uso en notarías, bibliotecas, escuelas, universidades, o en el hogar puede ofrecer un beneficio tangible a este sector de la población. Hasta donde se ha podido establecer, este diseño es innovador en cuanto a su enfoque como hardware y software abiertos y de bajo costo. Se espera que el listado de componentes, diseños, e instrucciones de ensamblaje se hagan disponibles a los interesados a través de la plataforma GitHub a mediados del año 2020.

## 17. Conclusiones

Se ha presentado un sistema mínimo para el desarrollo de aplicaciones científicas, el cual sigue los parámetros indicados en la propuesta inicial, como son un enfoque de caja de cristal, pocas dependencias de otras librerías, flexibilidad para hacer cambios, y aplicabilidad académica o industrial.

Se validó la herramienta mediante tres casos cercanos a la academia en diversas disciplinas:

- Problema de los n-cuerpos: solución 3D por métodos de Euler y Runge-Kutta 4.
- Estudio dinámico de mecanismo de levas.
- Caracterización de la modulación 2-FSK mediante análisis de Monte Carlo para un detector basado en cruces por cero.

Se aplicó con éxito el sistema a la solución de un caso industrial, el cual no fue planeado como parte del proyecto pero que se consideró factible implementar con el mismo: el control de un sistema neumático para pruebas de impacto.

La cercanía al hardware del lenguaje C, el control del puerto GPIO (General Purpose Input/Output) mediante un solo nivel de abstracción, y su manejo a nivel de registros permitieron la síntesis de una señal cuadrada con una frecuencia de 20.15 MHz, en comparación con los 329.8 KHz medidos haciendo uso de la librería convencional Rpi de Python. Así mismo, se alcanzó una resolución para las rutinas de temporización y de medida de tiempo del orden de 10  $\mu\text{s}$ .<sup>5</sup>

Estos resultados indican la aplicabilidad del sistema propuesto hacia el desarrollo de aplicaciones normalmente orientadas a sistemas operativos de tiempo real o a sistemas embebidos “puros”. Se puede hablar, por consiguiente, de que el proyecto permite el desarrollo de aplicaciones en modo “soft real time”. El manejo de interrupciones, sin embargo, requiere investigación adicional y posiblemente modificaciones a nivel del *kernel*, o el uso de técnicas avanzadas que permitan simular una interrupción, como el uso de un *thread* independiente dedicado exclusivamente a efectuar la lectura sostenida del puerto de entrada o “*polling*”.

Aunque el modelo de software no sufrió mayores cambios durante la fase de desarrollo de las librerías científicas, sí debieron ser evaluados tres modelos de software diferentes para abordar el soporte gráfico, pruebas que resultaron ser más dispendiosas y complejas de lo inicialmente considerado:

- Modelo basado en SDL (Simple DirectMedia Layer).
- Modelo basado en X11.
- Modelo basado en la interfaz “framebuffer”.

---

<sup>5</sup> Ambos resultados aplican en concreto al sistema en el cual se efectuaron las pruebas, una tarjeta Raspberry Pi 3 B+

Tal evaluación no fue meramente conceptual, sino que efectivamente se alcanzó en cada un estado funcional, durante el cual se comprobaron deficiencias y compromisos difíciles de aceptar:

- El modelo basado en X11 exhibió rezagos históricos, un bajo desempeño, y limitaciones para la consulta de datos de la imagen en el servidor. Aceptar este modelo obligaría adicionalmente a sacrificar la portabilidad a sistemas Windows.
- El modelo basado en “framebuffer” obliga al usuario a implementar su propio sistema de Entrada/Salida o consola y a sacrificar las ventajas ofrecidas por X11 (y por los ambientes de escritorio) para la ejecución simultánea y salto entre aplicaciones.

Por su parte el modelo considerado inicialmente basado en SDL, mantuvo un buen desempeño, probó ser inmediatamente compatible entre plataformas x86 y ARM, y es portable a sistemas Windows. La abstracción de las gráficas y su generación a través de estructuras de datos y bloques de memoria independientes garantizan la posibilidad de migrar a sistemas nuevos o a abandonar SDL cuando se deseé.

Se identificaron áreas de crecimiento del sistema y posibles mejoras, como son:

- El diseño y elaboración una librería que permita la detección y generación del sonido en forma simple, en forma análoga a como se hace para el caso de las gráficas.
- La adición de soporte de interrupciones en el GPIO (en sistemas Raspberry Pi).
- La adición de primitivas, mejoras al sistema gráfico, y mejor soporte de la eliminación de elementos ocultos en gráficas 3D.

Se encontraron alternativas a algunas limitaciones de los sistemas científicos actuales, como la baja portabilidad, un alto número de dependencias, y la dificultad de accederlos para efectuar cambios.

El desarrollo del proyecto brinda pautas para el mejor planeamiento de sistemas complejos de software, para su documentación, y para la división de las tareas para el trabajo en equipo.

Finalmente, se encontró que los proyectos modulares de tipo abierto pueden apalancar o servir de base a desarrollos derivados, entre los cuales se propusieron:

- El montaje de un sistema de realidad virtual de bajo costo para uso educativo y de entretenimiento.
- El desarrollo de un sistema magnificador para personas con limitación visual.

## 18. Bibliografía

- [1] MICROSOFT (2018). *Comparing Direct2D and GDI Hardware Acceleration*. Recurso en línea del Microsoft Windows Developer Center. Recuperado a partir de <https://docs.microsoft.com/en-us/windows/win32/direct2d/comparing-direct2d-and-gdi#differences-between-direct2d-and-gdi>
- [2] FREEDESKTOP.ORG (2014). *Wayland Frequently Asked Questions*. Recurso en línea de Freedesktop.org. Recuperado a partir de [https://wayland.freedesktop.org/faq.html#heading\\_toc\\_j\\_2](https://wayland.freedesktop.org/faq.html#heading_toc_j_2) <https://blogs.igalia.com/itoral/2014/07/29/a-brief-introduction-to-the-linux-graphics-stack/>
- [3] DILGER, D. (2018). *Why macOS Mojave requires Metal — and deprecates OpenGL*. Recurso en línea de AppleInsider.com. Recuperado de <https://appleinsider.com/articles/18/06/28/why-macos-mojave-requires-metal---and-deprecates-opengl>
- [4] VENTUREBEAT.COM (2018). *Apple defends end of OpenGL as Mac game developers threaten to leave*. Recurso en línea de Venturebeat.com. Recuperado a partir de a partir de <https://venturebeat.com/2018/06/06/apple-defends-end-of-opengl-as-mac-game-developers-threaten-to-leave/>
- [5] APPLE (2018). *Making Great OpenGL Applications on the Macintosh*. Recurso en línea de Apple Developer. Recuperado a partir de [https://developer.apple.com/library/archive/documentation/GraphicsImaging/Conceptual/OpenGL-MacProgGuide/opengl\\_pg\\_concepts/opengl\\_pg\\_concepts.html](https://developer.apple.com/library/archive/documentation/GraphicsImaging/Conceptual/OpenGL-MacProgGuide/opengl_pg_concepts/opengl_pg_concepts.html)
- [6] JONES, O. (1990). *Introduction to the X-Window System*. Englewood Cliffs, N.J.: Prentice Hall.
- [7] THAIN, D. (2011). *Gfx: A Simple Graphics Library*. Recurso en línea de la Universidad de Notre Dame. Recuperado a partir de <http://www.nd.edu/~dthain/courses/cse20211/fall2011/gfx>
- [8] BROWN, Z. (2015). *Diff -u: What's New in Kernel Development*. Recurso en línea de Linux Journal (no disponible en la actualidad, se adjunta copia bajo Anexo No. 1).  
<https://www.linuxjournal.com/content/diff-u-whats-new-kernel-development-16>
- [9] LARABEL, M. (2015). *A Call To Stop Making FBDEV Linux Frame-Buffer Drivers*. Recurso en línea de Phoronix. Recuperado de [https://www.phoronix.com/scan.php?page=news\\_item&px=Linux-FBDEV-Stop-Making-Drivers](https://www.phoronix.com/scan.php?page=news_item&px=Linux-FBDEV-Stop-Making-Drivers)
- [10] BURDEN R. y FAIRES D. (2004). *Numerical Analysis, 8th Edition, pp 278-290*. Thompson Brooks/Cole.

- [11] ISRAEL, R. (2002). *Fourth-Order Runge-Kutta Method*. Recurso en línea de la Universidad de British Columbia. Recuperado a partir de <http://www.math.ubc.ca/~israel/m215/runge/runge.html>
- [12] FAIRHEAD, H. (2016). *Raspberry Pi And The IoT In C - Input And Interrupts*. Recurso en línea de I/O Press. Recuperado de <https://www.iot-programmer.com/index.php/books/22-raspberry-pi-and-the-iot-in-c/chapters-raspberry-pi-and-the-iot-in-c/55-raspberry-pi-and-the-iot-in-c-input-and-interrupts?showall=1>
- [13] PORTER, T. y DUFF, T. (1984). *Compositing Digital Images*. Lucasfilm Ltd. Paper presentado en Siggraph, 1984. Recuperado de <https://keithp.com/~keithp/porterduff/p253-porter.pdf>
- [14] ABRAMOWITZ, M. y STEGUN, I. (1972). *Handbook of Mathematical Functions, sección 7.1.26*. United States Department of Commerce, Printing Office. Recurso en línea, disponible en [http://people.math.sfu.ca/~cbm/aands/abramowitz\\_and\\_stegun.pdf](http://people.math.sfu.ca/~cbm/aands/abramowitz_and_stegun.pdf)
- [15] MOCHEL, P. (2011). *sysfs – The filesystem for exporting kernel objects*. Recurso en línea de Kernel.org. Recuperado de <https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt>
- [16] DIJKSTRA, E. W. (1961). *Making a Translator for ALGOL 60*. Stichting Mathematisch Centrum. Recuperado de <http://www.cs.utexas.edu/~EWD/MCReps/MR35.PDF>
- [17] CHUDNOVSKY, D. y CHUDNOVSKY, G. (1988). *Approximation and complex multiplication according to Ramanujan*. Ramanujan revisited: proceedings of the centenary conference. Academic Press. Resumen de la publicación recuperada de Wikipedia: [https://en.wikipedia.org/wiki/Chudnovsky\\_algorithm](https://en.wikipedia.org/wiki/Chudnovsky_algorithm)
- [18] HALFACREE, G. (2018). *Benchmarking the Raspberry Pi 3*. Recurso en línea de Medium Technology. Recuperado de <https://medium.com/@ghalfacree/benchmarking-the-raspberry-pi-3-b-plus-44122cf3d806>

## 19. Anexo No. 1 (publicación ya no disponible)

diff -u: What's New in Kernel Development  
by Zack Brown on December 10, 2015

Linux capabilities are one of the more fluid and less defined regions of kernel development. Linus Torvalds typically has no trouble violating POSIX standards if he sees a better way of doing something. In the case of filesystem capabilities, however, there's no standard to violate. The best we've got is a POSIX draft document that was discarded before becoming official. So really, anyone with a good idea can come along and make big changes in that part of the kernel.

Filesystem capabilities refer to a finer-grained set of permissions than the traditional choice between running something as a regular user or running as root.

Recently, Eric W. Biederman and Andy Lutomirski found themselves tackling filesystem capabilities from opposite directions. Eric wanted to allow a process that's been granted one set of capabilities to invoke system calls using an even more constrained set of capabilities. Presumably, the goal would be to increase security by preventing system calls from being abused for nefarious purposes. And, Andy wanted to allow one process to allow a completely separate process to perform system calls on its behalf. This might allow the formation of system call services to centralize all system call usage and make it easier to secure those uses.

The discussion went round and round. Eric's idea, as he later clarified, was actually a bit broader than it appeared at first glance—he wanted to convert the Linux implementation of POSIX capabilities into "Real Capabilities". The term Real Capabilities refers to a computer science concept that pre-dates POSIX capabilities. It refers to the idea of giving a process some sort of token that allows it to perform a specified action on a specified object.

Ultimately, nothing about capabilities, or any new patches in that area, can have real clarity until they go into the kernel. Before then, there's always the possibility that they'll violate something important or aim in the wrong direction. But, it's cool to watch Eric and Andy, and lots of other folks, trying to figure it out.

One recurring problem with Linux is the need for backward compatibility. Actually, this affects virtually the whole Open Source world, but Linus Torvalds takes a particular strict stance on the issue with regard to the Linux kernel. If there's a compiled piece of user software out in the wild that relies on a kernel feature, even a dumb kernel feature, then future kernels have got to support that feature, so that the piece of user software will continue to run after a kernel upgrade.

It makes sense. But as Andy Lutomirski recently said, the result was a batch of features that existed only to support old programs. And by carrying these features perpetually into the future,

he said, newer software ran the risk of accidentally using those features or even becoming reliant on them.

He proposed allowing new software to turn off those compatibility features explicitly, but that turned out to be more complex than he'd originally thought. Specifically, one of his cornerstone ideas—granting the ability of newer software to turn off the vsyscall page—was not easy to arrange. Andy's initial idea was to have the compiler identify at compile time software that used newer versions of libc, and then have that piece of software elect to disable vsyscall at runtime. But, he didn't see a good way to accomplish that, and Brian Gerst pointed out that vsyscall was globally shared and couldn't be shut off for individual processes.

This actually turned out not to be 100% true. Although Andy agreed that vsyscall was shared globally, the mechanisms to execute it were all emulated in the kernel, and those could be disabled on a per-process basis.

Rich Felker proposed another workaround for vsyscall's global availability. He said the kernel could simply unmap all means of executing vsyscall. Any older software that tried to access it would generate a page fault, which the kernel could then catch and emulate vsyscall just for that program.

But, Andy didn't go for that idea. He said that modern instrumentation tools might want to read the targets of calls, and a page fault would prevent that. Any vsyscall solution, Andy said, had to maintain compatibility for those tools.

On the other hand, as Rich said, those modern tools might never be used on ancient binaries in practice. And even if they were, it might be possible to code up specific kernel workarounds for each use case in a less invasive way than trying to come up with a complete solution for vsyscall.

It's a robust debate, complicated by the fact that it's difficult to know for sure if anyone is actually running old binaries that depend on this or that kernel compatibility feature. But, Andy made it clear that cleaning out compatibility features was not really his primary goal, so much as it was to eliminate potential security holes. Apparently, Google's Project Zero had identified more exploits: <http://googleprojectzero.blogspot.com/2015/08/three-bypasses-and-fix-for-one-of.html>.

The Linux framebuffer, once a bastion of innovation, is now on the chopping block, in favor of the Direct Rendering Manager (DRM) subsystem. The fbdev maintainer, Tomi Valkeinen, has asked everyone to stop submitting new fbdev drivers and to aim their efforts at DRM instead.

It was not as uncontroversial as you might think. It turned out, as folks like Thomas Petazzoni said, that it still was easier to write very simple drivers for fbdev, than it was for DRM. Just in

terms of lines of code, Geert Uytterhoeven noticed that the simplest fbdev drivers were just a few hundred lines of code, while the simplest DRM drivers might require a couple thousand.

No one argued that this would be a permanent problem. If anything, the discussion highlighted the need to write some supporting libraries for DRM and help speed up the ultimate elimination of fbdev.

*Zack Brown is a tech journalist at Linux Journal and Linux Magazine, and is a former author of the "Kernel Traffic" weekly newsletter and the "Learn Plover" stenographic typing tutorials. He first installed Slackware Linux in 1993 on his 386 with 8 megs of RAM and had his mind permanently blown by the Open Source community. He is the inventor of the Crumble pure strategy board game, which you can make yourself with a few pieces of cardboard. He also enjoys writing fiction, attempting animation, reforming Labanotation, designing and sewing his own clothes, learning French and spending time with friends'n'family.*

**20. Anexo No. 2. Anteproyecto.**

**ANTEPROYECTO PARA TRABAJO DE GRADO**

**MAESTRÍA EN INGENIERÍA**

**MODALIDAD: PROFUNDIZACIÓN**

**DESARROLLO DE UNA HERRAMIENTA DE TIPO MINIMALISTA PARA COMPUTACIÓN CIENTÍFICA  
Y GRÁFICA PARA AMBIENTES DE ESCRITORIO Y SISTEMAS DE BAJO COSTO**

Presentado por:  
Diego Alberto Vélez Henao, IEE.

Director:  
Jorge Mario Londoño Peláez, PhD.

**UNIVERSIDAD PONTIFICIA BOLIVARIANA  
MEDELLÍN, COLOMBIA  
Abril de 2018**

## **DEVELOPMENT OF A MINIMALISTIC-TYPE SCIENTIFIC AND GRAPHICAL COMPUTING TOOL FOR DESKTOP AND LOW-COST ENVIRONMENTS**

### **Abstract**

How much scientific computing can be done with very little?

Nowadays, both the industrial and the academic sectors have at their disposal a wide variety of software tools for scientific computing. These packages generally incorporate graphic capabilities, either by own developments or by using independent libraries.

The reduction on the cost of the hardware and the increase of its capabilities (mainly memory, processing power, and transfer rates) which have occurred during the last decades, have unleashed software developers and allowed them to use a higher number of abstraction layers, more complex data structures, and larger memory blocks, while putting on a secondary plane other design goals, such as: a smaller memory footprint, higher efficiency, and code concision. There are, therefore, two tendencies: one whose main goal is to offer new features that could be commercialized, possibly at the expense of requiring larger resources; and another focused in technical aspects such as reaching a higher quality, efficiency, clearness in the code, a lower memory footprint, or reducing power consumption. The first approach is common in the development of systems for desktop environments, while the second is usually taken in embedded systems.

This project seeks to apply techniques often used in embedded and industrial development to the case of scientific computing, to obtain a scientific computing system applicable to both desktop and lower-cost modules. It should be noted that within this document the term "scientific computing" refers to performing science and engineering calculations including graphics as an indispensable method to present results. Likewise, the project is denominated "Minimalistic" in the sense of doing "more with less", akin to those who satisfy their vital necessities by making creative use of natural, free, or low-cost elements.

The following characteristics are proposed for the development of a minimalistic-type scientific computing tool:

- General simplification and elimination of redundant elements.
- A "crystal box" approach, where the internal operation of all components can be studied, modified, or improved.
- Measured use of abstraction layers, to obtain efficiency and to facilitate tracking between layers.
- Leverage in the most accepted and compatible language to this date: the C programming language.
- Development of a memory-based, hardware-independent graphic subsystem, with an associated API (Application Programming Interface), which will include graphical primitives and utilities.
- Creation of a basic library with algebraic and numeric functions, with functions to evaluate real and complex-valued expressions written in mathematical notation.
- Fewer restrictions than those stipulated by GPL (General Public License) and similar licenses.

The methodology to be followed consists of three phases: planning, implementation, and testing. Each one covers certain tasks:

- During the planning phase an investigative review of the tendencies and most common requirements for scientific computing will be performed; a design philosophy will be defined; and the software model will be outlined.
- The implementation phase encompasses the detailed specification of the software model; the development of the system at each layer's level; and the integration of all modules. This stage comprehends, therefore, the elaboration and testing at the individual level of all functions associated to each one of the constituent layers. It covers, finally, the integration of all constituent modules in a unified system.

- The testing phase seeks to ensure the correct operation of the system. It comprehends numeric, graphic, stability, and performance tests.

This project may have several types of impact:

- Contributes to disseminate scientific and engineering computing.
- Provides a tool for business or academic use which not only can be used, but also modified or extended as needed by basic or intermediate users. This isn't normally possible in commercial packages, and although it is for open source software, in practice is a task suited only to expert developers.
- Creates, within the open source world, a more relaxed alternative to packages licensed under GPL (General Public License).
- Makes easier the development of projects requiring numeric or graphic features.
- Positively influences the education in engineering, since the "crystal box" approach can encourage students to modify the system, or to write their own applications. It is suggested that undertaking projects offers a greater pedagogic value than simply learning the operation of finished programs. This contributes to developing a spirit of self-sufficiency which is convenient to organizations, academia and our country.

**Keywords:** scientific computing, engineering computing, numerical analysis, computer graphics, simulation, C/C++ language, crystal box.

## **DESARROLLO DE UNA HERRAMIENTA DE TIPO MINIMALISTA PARA COMPUTACIÓN CIENTÍFICA Y GRÁFICA PARA AMBIENTES DE ESCRITORIO Y DE BAJO COSTO**

### **Resumen**

¿Qué tanta computación de tipo científico puede hacerse con muy poco?

En la actualidad, tanto el sector industrial como el académico disponen de una amplia variedad de herramientas de software para la computación científica. Estos paquetes generalmente incorporan capacidades gráficas, bien sea por medio de desarrollos propios o mediante el uso de librerías independientes.

La disminución del costo del hardware y el aumento de sus capacidades (principalmente memoria, velocidad de procesamiento, y velocidad de transferencia) ocurridos durante las últimas décadas han dado rienda suelta a los desarrolladores de software para el uso de un número cada vez mayor de capas de abstracción, de estructuras de datos más complejas y de mayores bloques de memoria, dejando en un plano secundario otras metas de diseño, como son: menor huella en memoria, mayor eficiencia, y concisión en el código. Existen, pues, dos tendencias: una cuya meta principal de diseño es ofrecer nuevas características que se puedan comercializar, a expensas quizás de requerir mayores recursos; y otra centrada en aspectos técnicos como el mejoramiento de la calidad, eficiencia, claridad del código, huella en memoria, o el consumo de energía. El primer enfoque suele ser común en el ámbito de escritorio, mientras que la segundo lo es más en el embebido.

Este proyecto busca aplicar técnicas de desarrollo del ambiente embebido e industrial para el caso concreto del cálculo científico, y desarrollar un sistema aplicable tanto a sistemas de escritorio como a módulos de bajo costo. Se aclara que en el contexto de este documento el término "cálculo científico" hace referencia al cálculo para ciencias e ingeniería, incluyendo las gráficas como medio indispensable para la presentación de resultados. Se denomina el proyecto "Minimalista" en el sentido de hacer "más con menos", al estilo de aquellas personas que satisfacen sus necesidades vitales haciendo uso creativo de elementos naturales, gratuitos, o de bajo costo.

Se proponen las siguientes características para el desarrollo de una herramienta de computación científica de tipo minimalista:

- Simplificación general y eliminación de elementos redundantes.
- Enfoque de "caja de cristal", con el cual la operación interna de los componentes pueda ser estudiada, modificada, o mejorada.
- Utilización mesurada de capas de abstracción, a fin de mantener eficiencia y facilitar el seguimiento entre capas.
- Apalancamiento en el lenguaje con mayor aceptación y compatibilidad a la fecha: el lenguaje C.
- Desarrollo de un subsistema gráfico en memoria, independiente del hardware, con una API (Interfaz de Programas de Aplicación) asociada que incluya primitivas y utilidades gráficas.
- Elaboración de una biblioteca básica de funciones algebraicas, numéricas, con evaluación funcional en notación matemática de funciones en variable real y compleja.
- Restricciones menores a las estipuladas por licencias tipo GPL (General Public License).

La metodología a seguir comprende tres fases: planeación, implementación, y pruebas. Cada una de ellas cubre ciertas tareas:

- Durante la fase de planeación se investigarán las tendencias y los requerimientos más comunes en cómputo científico; se definirá una filosofía o conjunto de principios a seguir; y se esbozará en forma general el modelo de software.
- La fase de implementación abarcará la especificación detallada del modelo de software; el desarrollo del sistema a nivel de cada capa; y la integración de todos los módulos. Esta etapa

comprende, por consiguiente, la elaboración y pruebas a nivel individual de las funciones asociadas a cada una de las capas. También cubre, finalmente, la integración de los módulos constituyentes en un sistema unificado.

- La fase de pruebas busca asegurar la correcta operación del sistema. Comprende la ejecución de pruebas numéricas, gráficas, de estabilidad y de desempeño.

Este proyecto puede tener varios tipos de impacto:

- Contribuye a difundir la computación para ciencias e ingeniería.
- Provee una herramienta para uso empresarial o académico que no solamente pueda ser utilizada, sino modificada o extendida según se necesite por usuarios de nivel básico o intermedio. Esto normalmente no es posible en el caso del software comercial, y aunque lo es para el software libre, en la práctica es una tarea que solamente pueden acometer desarrolladores expertos.
- Crea, dentro del mundo del software libre, una alternativa más cómoda a los paquetes licenciados bajo GPL (General Public License).
- Facilita la elaboración de proyectos que requieran aspectos numéricos o gráficos.
- Influye positivamente en la educación en ingeniería, ya que el enfoque de "caja de cristal" puede animar estudiantes a modificar el sistema o a escribir sus propias aplicaciones. Se sugiere que emprender proyectos propios ofrece mayor valor pedagógico que aprender la operación de programas terminados. Esto contribuye a desarrollar un espíritu de autosuficiencia conveniente para las organizaciones, la academia, y nuestro país.

**Palabras clave:** computación científica, cálculo de ingeniería, análisis numérico, gráficas de computadora, simulación, lenguaje C/C++, caja de cristal.

## **1. PARTICIPANTES**

Estudiante:	Diego Alberto Vélez Henao
Código de Estudiante:	8374
Teléfono:	+1.587-700-0229
Correo electrónico:	diego.velezh@upb.edu.co
Programa:	Maestría en Ingeniería
Director:	Jorge Mario Londoño Peláez, PhD.
Teléfono:	+57-4354-4522
Correo electrónico:	jorge.londono@upb.edu.co

## **2. INTRODUCCIÓN Y PLANTEAMIENTO DEL PROBLEMA**

Pese a la abundancia de herramientas para cálculo científico y de ingeniería, tanto comerciales como libres, existen casos en los cuales estas no resultan las más adecuadas. Los casos que se describen a continuación han sido identificados en ambientes empresariales y, otros, por discusión con miembros de la academia.

### **2.1. Cuando la portabilidad real entre el código de escritorio y el del prototipo es menor que la deseada**

En el desarrollo de proyectos de ingeniería electrónica, es usual utilizar escribir código de estudio o simulación durante la fase inicial de estudio del problema, y en etapas posteriores, reescribirlo en su totalidad o en parte bajo la plataforma final elegida para el producto. La repetición de parte del trabajo se acepta como una parte natural del desarrollo de los proyectos, pero podría reducirse haciendo uso de un sistema de mayor compatibilidad que permita realizar estudios a nivel de escritorio y portarlos al sistema final.

### **2.2. Cuando existen restricciones empresariales al uso del software libre**

Las áreas Legales y de Administración de Sistemas en las organizaciones son reacias al uso del software libre por razones como: preocupaciones sobre la propiedad intelectual de productos derivados de una herramienta; ausencia de garantía; incertidumbre en caso de requerir soporte técnico; y consideraciones sobre la seguridad (por ejemplo, integridad de la fuente o código malicioso).

### **2.3. Cuando el alto costo del software comercial no permite su adquisición, o cuando su uso es ocasional**

El costo de las licencias para software comercial de ciencias e ingeniería es alto para las compañías y resulta difícil justificarlo sobre todo para uso no recurrente. Más aún, la tendencia de esta industria es llevar a sus clientes a modelos de suscripción, los cuales, a la larga, aumentan su costo, pese a algunos beneficios como mejor soporte y actualizaciones.

### **2.4. Cuando se desean hacer alteraciones a la herramienta de computación**

En ciertos tipos de desarrollos se desea hacer alteraciones a la forma en que opera la herramienta de desarrollo. Un caso común es cuando se desea que un sistema efectúe procesos de entrada o salida sobre un tipo de interfaz o puerto no soportado.

## **2.5. Cuando se desee adecuar código de una plataforma existente a otra con menores recursos**

Se busca hacer uso de elementos como: una planeación organizada, estructura ligera, reducción de redundancias, eliminación de componentes innecesarios, y de un enfoque de "caja de cristal" para lograr un sistema sencillo que se pueda adaptar con pocos cambios entre diferentes plataformas.

## **3. ESTADO DEL ARTE**

### **3.1. Exploración**

El estado del arte se corroboró mediante búsquedas en la biblioteca digital Xplore del Instituto de Ingenieros Electricistas y Electrónicos, IEEE. Pese a la variedad de herramientas para el cálculo científico, existen pocas alternativas que reúnan simultáneamente las siguientes características:

- Que no tengan costo y sean de libre uso y modificación por parte de desarrolladores o usuarios.
- De enfoque minimalista o reducido (enfoque en los elementos esenciales de la computación científica).
- Basados un lenguaje de programación tradicional, en lugar de implementar su propia gramática y sintaxis.
- Que generen programas ejecutables sin la pérdida de eficiencia de los lenguajes interpretados.
- Que provean soporte directo de gráficas por medio de una librería propia.
- Con orientación al cálculo científico, de ingeniería, y que permitan hacer simulaciones.

Se hace a continuación una breve reseña sobre artículos relevantes a partir del año 2002, presentados en orden cronológico. Posteriormente, se seleccionarán los sistemas que guardan mayor similitud con el proyecto propuesto explicando las diferencias.

#### **3.1.1. Simulation software and model reuse: a polemic (Pidd, M., 2002)**

- Hace un enfoque en los diferentes niveles de reutilización de código y de modelos en el software. Se discute la relación entre los componentes de una simulación.
- Discute el tema de la diferencia entre "validez" y "credibilidad" de un modelo.

#### **3.1.2. Parallelization of the GNU Scientific Library on heterogeneous systems (Aliaga, J., Almeida, F., et al., 2004)**

- Presenta un desarrollo para implementar paralelismo en la Librería Científica GNU.
- Muestra que este tipo de librerías libres e independientes son una opción válida para el desarrollo de software de ciencias.

#### **3.1.3. Sound Software Development for Engineering Simulations (Macisaac, D. & Chan., A., 2006)**

- Discute los tipos de errores más comunes en el desarrollo de software y su ocurrencia en un proyecto planteado en un ambiente universitario, utilizando Matlab.
- Plantea que "aunque los desarrolladores de simulaciones en general son cuidadosos en la implementación de modelos, no siguen buenas prácticas que permitan que el código sea amigable al usuario o reutilizable".

3.1.4. Python for Scientific Computing (Oliphant, T. E., 2007)

- Presenta las ventajas del uso de Python como lenguaje para la programación de tipo científico. En el informe final de tesis se discutirá este tema y, aunque la aplicabilidad de Python como herramienta no está en discusión, hay ventajas que hacen todavía de C un sistema adecuado para la industria.

3.1.5. A Software Chasm: Software Engineering and Scientific Computing (Kelly, D.F., 2007)

- Presenta la separación entre los mundos del desarrollo de software y el desarrollo de aplicaciones de dominio específico (en particular el de la computación científica), y las acciones a tomar por cada uno para cerrar la brecha.
- Es conveniente tener en cuenta este punto en la medida que el proyecto pretende acercar la computación científica a usuarios no especializados en programación.

3.1.6. Construction and Application of Linux Virtual Server Cluster for Scientific Computing (Tang, H., She, R., et al., 2008)

- Estudio sobre el paralelismo mediante *clusters*, para aplicaciones de ingeniería.

3.1.7. The PlayStation 3 for High-Performance Scientific Computing (Kurzak, J., Buttari, A., et al 2008)

- Discuten el uso de lenguajes de dominio específico para facilitar el uso de fórmulas matemáticas directamente, y en algunos casos, optimizar la evaluación.
- La conveniencia de utilizar fórmulas directamente es un tema abordado igualmente en el proyecto propuesto.

3.1.8. Daydreaming about Scientific Programming (Hinsen, K., 2013)

- El autor describe su sistema de cómputo científico "soñado". Se propone que dicho sistema debe aceptar los programas en una forma de alto nivel, similar al pseudocódigo, pero debe estar en capacidad de elaborar o al menos sugerir optimizaciones al programador.

3.1.9. Mobile Computers as Scientific Computing Machines (Smit, W., & Herbst, B., 2014)

- Presentan una reseña comparativa y medidas de desempeño relativo entre equipos de escritorio y algunos equipos móviles como Smartphones y Tabletas.

3.1.10. Scientific computing over the Web in various programming languages: Solving problems in Fortran, C, and Octave (Casquilho, M., & Cunha, M., 2014)

- Trata el tema de accesibilidad de usuario para el cómputo científico.

3.1.11. Vertical Integration (Turk, M., 2015)

- El autor presenta las ventajas y desventajas de la integración vertical en el caso del desarrollo del software científico, reconociendo que es una discusión compleja y que existen casos en los cuales la integración vertical puede ser conveniente y otros en los que no.

- Es un artículo de alta relevancia respecto al proyecto propuesto, pues uno de los dilemas que se plantean al desarrollar sistemas es utilizar recursos externos (por ejemplo, utilizar librerías externas especializadas) o desarrollar las propias.
- Una de las hipótesis presentadas en este artículo es que la integración vertical resulta conveniente porque ofrece beneficios como mayor consistencia en los tipos de datos, homogeneidad en los nombres de las funciones, menor dependencia de terceros y mayor control.

3.1.12. Scientific computing using consumer video-gaming embedded devices (Volkema, G., & Khanna, G., 2017)

- Presentan características (como el bajo costo por MFLOP, Millions of Floating Point Operations per Second) que hacen de interés las consolas de videojuegos para aplicaciones de supercomputación. No presentan (al menos no en detalle), información, o los pasos seguidos para la realización de las pruebas.

3.1.13. History of computer simulation software: An initial perspective (Nance, R. E., & Overstreet, C. M., 2017)

Este es el artículo de alta relevancia para evaluar el estado del arte, por varias razones:

- Los autores Nance y Overstreet revisan la historia, desarrollo, novedades, características y tendencias de la industria del software para simulación.
- Es un artículo relativamente reciente, del año 2017.
- Presenta un alto nivel de detalle y se apoya metodologías como encuestas y análisis del número de publicaciones en ciertos temas. Por ejemplo, algunas tendencias son: a) Disminución del interés en la computación distribuida o paralela después de un período de auge en los años 90's; b) Disminución en el soporte de gráficas, igualmente, después de un período de auge en los años 90's; c) Aumento sostenido del uso de extensiones para simulación a lenguajes de propósito general; d) Aumento sostenido del uso de lenguajes de simulación de propósito específico.

3.1.14. Julia: A Fresh Approach to Numerical Computing (Bezanson J., et al., 2017)

Hace una presentación general de Julia y de su aplicación al cálculo numérico. Discute el dilema de que se requiera un lenguaje para el desarrollador y otro para el usuario, problema que argumenta se evita con Julia. Presentan algunas de las características: alta eficiencia, soporte de datos sin tipos, despacho múltiple, y otras. Reconocen, sin embargo, que C es el "estándar de oro" en computación intensiva, el cual es un punto clave en la justificación de este proyecto.

3.1.15. Practically surreal: Surreal arithmetic in Julia (Roughan, 2019)

Presenta un estudio sobre los números surreales (los cuales son definidos recursivamente) y se exploran casos concretos, haciendo uso de Julia y, concretamente, de su recursividad (capacidad de una función de llamarse a sí misma).

La bibliografía anterior ayuda a identificar los productos que constituyen el Estado del Arte de la computación científica de propósito general, asequible al público. Se resumen los principales sistemas en la siguiente matriz:

Sistema	Criterio				
	Tipo de Licencia	Adaptabilidad a Sistemas de bajo costo	Compatibilidad con lenguajes tradicionales	Soporte de Gráficas	Orientada al cálculo científico
Matlab	Software comercial de alto costo.	Limitada, se basa en la conversión del código al lenguaje C.	El código escrito en Matlab no es compatible con otros lenguajes, pero permite enlaces de funciones en C y de otros lenguajes.	Abundante.	Si. Es un estándar en la academia e industria.
Octave	Libre (GPL)	Baja o no adaptable.	El lenguaje de Octave no es directamente compatible con lenguajes tradicionales. Es parcialmente compatible con Matlab y se puede enlazar a otros lenguajes.	Abundante pero no tiene librería gráfica propia. Se basa en OpenGL y/o gnuplot.	Si
Scilab	Libre (GPL)	Limitada, se basa en la conversión del código al lenguaje C.	No es compatible con otros lenguajes.	Limitado.	Si
FreeMat	Libre (GPL)	Baja o no adaptable.	No es compatible con otros lenguajes.	Basado en OpenGL	Si
GNU Scientific Library (GSL)	Libre (GPL)	Altamente adaptable, por estar escrita en el lenguaje C, el cual es normalmente soportado en sistemas embebidos.	Altamente compatible.	No provisto. Requiere el uso de librerías externas.	Si
Python con uso de librerías: Numpy, Scipy, etc.	Libre (BSD)	Limitada. El proyecto Micropython implementa una versión reducida de Python 3 compatible solamente con algunos procesadores.	Es un caso especial: Python no es compatible directamente con un lenguaje tradicional, pero se ha convertido en uno por derecho propio. Soporta enlaces a programas escritos en C y en otros lenguajes.	No provisto. Requiere el uso de librerías externas.	Depende. Es de propósito general, pero se adecúa muy bien al cálculo científico.
Julia	Libre (GPL)	Baja o con carácter experimental a la fecha.	Es de alta compatibilidad con C, pues es posible llamar funciones de otras librerías en C.	Abundante, a través de librerías que han sido desarrolladas tanto en Julia.	Sólida orientación al cálculo científico.

Tabla 1: Comparación de las más conocidas herramientas para cálculo en ciencias o ingeniería

Se aclara que hay muchas más herramientas similares (las cuales pueden ser independientes o asociadas). La tabla anterior da una buena idea de los sistemas con mayor aceptación.

### 3.2. Sistemas más representativos

Las opciones que guardan mayor similitud con el objetivo propuesto son: la librería científica GNU, el lenguaje de programación Python (con el apoyo de librerías numéricas y gráficas externas), y el lenguaje Julia (con librerías gráficas externas).

#### 3.2.1. Librería científica GNU

Es una librería extensa, con más de 1,000 funciones matemáticas. Es de un alto grado de portabilidad y simpleza. No incorpora un sistema gráfico. Es utilizada usualmente en espacios académicos, aunque no lo suele ser en el empresarial. Por ejemplo, no fue posible identificar su uso por parte de colegas o de compañeros de trabajo. Su menor difusión puede deberse a varios factores:

- a) La licencia GPL (General Public License) suele ser considerada “poco amigable a los negocios”.
- b) No incorpora un sistema gráfico que facilite al usuario la presentación de resultados.
- c) Es una librería de propósito específico y no general.
- d) Es más compleja de utilizar que productos comerciales tipo Matlab.

#### 3.2.2. Python

No provee soporte gráfico en forma directa. Las gráficas se realizan mediante librerías externas, las cuales son en general proyectos independientes y de difícil modificación por parte de usuarios principiantes o intermedios. En general, es difícil encontrar librerías gráficas para Python que tengan buen desempeño y que ofrezcan un juego de primitivas amplio. Algunas, como PyGame, son “wrappers” de otras librerías como SDL (Simple DirectMedia Layer), la cual, a propósito, tampoco incluye primitivas de dibujo. Otras, como PyGnuplot, son de alto nivel y orientadas a la elaboración de gráficos matemáticos, pero no son ideales para otras tareas como la manipulación directa de pixeles o para hacer animaciones.

Al ser un lenguaje interpretado, Python requiere una huella de memoria mayor que aquella de los sistemas compilados, factor que resulta adverso para su uso en sistemas basados en microcontrolador. Su eficiencia es menor a los sistemas compilados (Zhang, 2016). Además, el soporte de Python en sistemas embebidos es por ahora de tipo experimental.

No alcanza a la fecha el mismo grado de portabilidad que C. Por ejemplo, C es soportado por los microcontroladores AVR, PIC, y otros.

#### 3.2.3. Julia

Julia es un lenguaje de alto nivel, que logra ofrecer amigabilidad al usuario final, mantiene un nivel de desempeño razonable comparado con lenguajes compilados como C, y posee una fuerte orientación a la solución de problemas numéricos. En estos sentidos comparte algunas metas con el proyecto propuesto. Se mencionan a continuación las diferencias y las ventajas de abordar el problema de manera diferente:

- El esquema propuesto hace énfasis en utilizar el lenguaje C (el cual es de amplia aceptación y de uso extensivo en sistemas embebidos) en lugar de definir un nuevo lenguaje o forzar al usuario a aprender uno nuevo. Se complementará C con recursos para la elaboración de aplicaciones numéricas y gráficas, facilitando el desarrollo fácil de aplicaciones científicas no interpretadas, sin pérdida de eficiencia.
- Julia, al igual que otros lenguajes, no provee un sistema gráfico propio, sino que se apalanza en librerías asociadas como, por ejemplo, Plots.jl u otras. El esquema propuesto, aunque también es en sí otra librería, tiene como ventaja que realizará todas

sus operaciones en "pantallas virtuales" bajo el control del usuario. El proyecto deja a criterio de este la elección del método para transferir los datos de imagen a pantalla, los cuales podrían ser: DirectX (en sistemas Windows); OpenGL (sistemas Unix, Linux y Windows); SDL (Simple DirectMedia Layer); o escribiendo un controlador propio como suele ser el caso para el uso de pantallas LCD o OLED en sistemas embebidos.

- La adaptabilidad a un sistema más pequeño tipo microcontrolador se considera menor que la del lenguaje C.
- Julia, al ser un producto terminado, se debe aceptar tal como es (excepto quizás en el caso de usuarios muy avanzados capaces de modificar y depurar su código fuente). En contraste, las herramientas del sistema propuesto estarán escritas al estilo de caja de cristal: la manipulación del código no solamente le será permitida al usuario, sino que es algo que se anticipa y desea.

En conclusión, razones como la mayor aceptación de C, su mayor cercanía al hardware; la necesidad de un sistema gráfico independiente, y la meta de lograr un sistema modificable por los usuarios justifican la elaboración de un proyecto independiente.

Se hace sin embargo una aclaración: el proyecto propuesto, por desarrollarse en C, podría enlazarse con Julia (y con otros lenguajes) y utilizarse de maneras creativas. Por ejemplo, dado que Julia puede hacer llamados directos a funciones en C sin "lógica de pegamiento", podría beneficiarse de las capacidades gráficas del sistema propuesto.

## 4. OBJETIVOS

### 4.1. Objetivo general

Desarrollar, haciendo uso de técnicas como la simplificación, la reducción de componentes innecesarios, el apalancamiento en un lenguaje existente, y la escritura de código al estilo de "caja de cristal", un sistema de cómputo científico que integre capacidades gráficas, con el fin de lograr una herramienta para la industria y a la academia que se pueda usar directamente en ambientes Linux de escritorio y también en sistemas de bajo costo (tipo Raspberry Pi), que sea libre y de fácil modificación o extensión por parte de usuarios.

### 4.2. Objetivos específicos:

- 4.2.1. Desarrollar subsistemas que faciliten la creación de aplicaciones científicas bajo el lenguaje C, tales como: manejo matricial en memoria dinámica, utilidades para evaluación funcional en notación matemática convencional, y otras.
- 4.2.2. Implementar en software un sistema gráfico independiente del sistema y demostrar su funcionamiento bajo el sistema operativo Linux.
- 4.2.3. Desarrollar una API (Interfaz de Programas de Aplicación) que incorpore las principales primitivas gráficas y algunos módulos que faciliten la elaboración de gráficas planas y tridimensionales mas comunes en matemáticas o ciencias.
- 4.2.4. Alcanzar el estado de Prototipo Funcional y presentar casos de aplicación.

## **5. METODOLOGÍA DE LA INVESTIGACIÓN**

La metodología de investigación y desarrollo comprende las siguientes fases:

### **5.1. Revisión investigativa de los requerimientos mas comunes y tendencias actuales**

En esta fase se capturará información acerca las necesidades más comunes por parte de usuarios de sistemas de cálculo científico para consolidar una lista de características o funciones deseadas. Estos datos se obtendrán por medio de encuestas breves dirigidas a la población relevante: profesionales de la ingeniería en ejercicio técnico, docentes, y desarrolladores de software que hagan uso de funciones de ciencias o matemáticas.

Algunos elementos de esta lista se incorporarán al sistema si se determina que son de uso general y de práctica implementación. En caso contrario, no se elaborarán, pero se tendrán en cuenta sus características (por ejemplo, el número y tipos de argumentos requeridos, la cantidad de memoria necesaria, o su relación con otras partes del sistema) para permitir su adición en el futuro sin incurrir en demasiados cambios.

### **5.2. Definición de la filosofía de diseño**

Aquí se establecerán los principios fundamentales a seguir durante la elaboración del sistema, sin detallar cómo se hará posteriormente. No son reglas inamovibles, pero empiezan a establecer algunas pautas de desarrollo. La siguiente es una lista sujeta a revisión, la cual se presenta para ilustrar la idea:

Filosofía de diseño:

- "Menos es más": énfasis en concisión, simpleza y medida en el uso de recursos.
- Uso práctico de capas de abstracción y solamente en casos necesarios.
- El desempeño es importante, pero no al punto que arruine la claridad del código para la revisión o edición posterior por parte de terceros.
- Enfoque de caja de cristal: es una herramienta hecha para ser utilizada, modificada y adaptada por otros.
- Adherencia a la programación tradicional estructurada.
- Uso mínimo de librerías externas. Se emplearán únicamente para independizar el desarrollo de aspectos específicos del hardware como dispositivos apuntadores, de sonido, tarjeta de video, etc.

### **5.3. Definición del modelo de software**

Consiste en conceptualizar las capas y módulos que constituirán el software, indicando su función, como interactúan entre sí y su relación con el usuario y el hardware. El modelo normalmente se sintetiza en un diagrama de capas y bloques con conectores para denotar las interfaces y con flechas para indicar el flujo de la información.

### **5.4. Implementación**

Durante la implementación se materializarán la filosofía de diseño y el modelo del software en documentos de especificación funcional, código fuente, y programas ejecutables aptos para la ejecución de cálculos, generación de gráficas, o elaboración de simulaciones.

#### **5.4.1. Definición de interfaces**

Se definirá la información que intercambiarán las diferentes capas que fueron indicadas el modelo de software, incluyendo los tipos de datos, sus formatos, y códigos de éxito o de error.

#### 5.4.2. Implementación de funciones

Cada una de las capas en el modelo está constituida por grupos de funciones. En esta fase se desarrollará el código asociado a cada una de las funciones y se realizan pruebas a nivel individual. Al tratarse de un sistema científico en el cual se busca libertad en términos de licencias, la implementación es extensa, debido al número, nivel de detalle y pruebas requeridas para la puesta a punto de las funciones.

#### 5.4.3. Integración y pruebas de conjunto

Consiste en vincular los elementos del sistema buscando su operación conjunta. Este proceso puede señalar la necesidad de efectuar cambios a algunos de los componentes. El proceso de integración implica la realización de pruebas donde se evalúan, por ejemplo, la compatibilidad entre módulos, y se determina si afectan el desempeño de los demás. Se identifican elementos de menor desempeño o "cuellos de botella" y se elaboran estrategias para resolverlos o mitigarlos. El sistema empieza a adquirir una forma más cercana a un producto final.

#### 5.4.4. Pruebas de detalle

En esta etapa se identificarán errores o supuestos incorrectos. Para validar este sistema científico se emplearán las siguientes estrategias:

##### 5.4.4.1. Para aspectos del sistema

Se efectúan pruebas para confirmar la operación del sistema en relación con el hardware. Por ejemplo, se verifica la operación de dispositivos de entrada y salida, la asignación de memoria, y la escritura en archivos.

##### 5.4.4.2. Para aspectos numéricos y computacionales:

A fin de asegurar un grado razonable de calidad numérica se emplearán las siguientes técnicas:

- Comprobación por comparación con referentes tipo Matlab y Mathcad.
- Comprobación por medio de valores específicos: algunas funciones o métodos matemáticos arrojan valores notables bajo ciertos parámetros de entrada. Por ejemplo, el resultado de evaluar  $e^{i\pi}$  deberá ser  $-1 + 0 \cdot i$  con un número razonable de dígitos de precisión.
- Verificación contra casos anómalos: se probará como el sistema se comporta ante condiciones que deberían generar un error.
- Pruebas mediante funciones inversas: en algunos casos, un módulo de la herramienta puede utilizarse para verificar su contraparte. Por ejemplo, al aplicar, en variable compleja,  $\text{asin}(\sin(x))$ , el resultado debe ser x.
- Pruebas mediante funciones similares internas al sistema: bajo ciertas condiciones, algunas funciones deben ofrecer los mismos resultados. Por ejemplo, el resultado de una Transformada Rápida de Fourier puede corroborarse mediante la Transformada Directa de Fourier (y viceversa, siempre y cuando el tamaño del vector sea una potencia de dos).
- Comprobación mediante métodos aproximados: en algunos casos un cálculo puede comprobarse mediante la comparación con un resultado aproximado, como por ejemplo la aproximación de una expresión mediante el uso de las Series de Taylor.

#### **5.4.4.3. Para aspectos gráficos:**

Se utilizarán los siguientes métodos:

- Comprobación de estabilidad, para asegurar, por ejemplo, que no existan "fugas" o corrupción de memoria durante la generación de gráficas.
- Verificación de la correspondencia entre datos que representen imágenes en un bloque de memoria y el resultado pictórico representado en pantalla.
- Obtención de métricas de desempeño, por ejemplo, el número de cuadros por segundo que pueden presentados en un hardware determinado.
- Cada una de las primitivas gráficas será probada en forma independiente, y luego en forma conjunta con otras, llegando a la elaboración de imágenes más sofisticadas.
- Se revisará el funcionamiento de los mecanismos de transformación de coordenadas para la elaboración de imágenes en dos y tres dimensiones.

#### **5.4.5. Alcance del estado de Prototipo Funcional, compuesto mínimamente por un módulo de evaluación funcional de expresiones en variable real y compleja; un sistema gráfico en memoria; una API (Interfaz de Programas de Aplicación) para imágenes en dos y tres dimensiones; y librerías con funciones matemáticas o numéricas esenciales o, en defecto de alguna de ellas, de un método que permita su incorporación futura.**

En esta etapa el sistema alcanzará el estado de Prototipo Funcional, en el cual será posible su uso por parte de terceros.

### **5.5. Elaboración de casos de aplicación**

Al llegar a este punto se implementarán, sobre el sistema terminado, algunos "casos de aplicación". Es decir, se escribirán breves programas completos que resuelvan algún problema matemático o científico. Los casos de aplicación elaborados constituirán, además:

- Pruebas finales de la operación del sistema.
- Muestras del alcance de los objetivos durante la defensa de la tesis.
- Ejemplos para que otros usuarios conozcan el uso del sistema.

### **5.6. Documentación**

Se elaborará una guía de usuario con instrucciones de instalación, ejemplos, y una lista general de funciones que indique los tipos de datos requeridos, su orden, y posibles restricciones.

### **5.7. Liberación**

Consiste en hacer disponible el sistema a la comunidad académica y demás interesados. Se publicarán imágenes o "snapshots" en una página web, o en la plataforma GitHub.

## **6. ALCANCES, IMPACTOS Y CONTRIBUCIÓN**

### **6.1. Alcances**

Son parte del alcance del proyecto:

- El desarrollo al nivel de Prototipo Funcional de un sistema de computación científica que incluya un subsistema gráfico en software; una API (Interfaz de Programas de Aplicación) asociada; un

modulo procesador de expresiones en lenguaje matemático convencional; y librerías matemáticas de soporte. El sistema será compatible al momento de la entrega con PCs de escritorio en ambiente Linux y con sistemas de bajo costo tipo Raspberry Pi 3, a los cuales está primordialmente enfocado el proyecto, en tanto que son plataformas de bajo costo, de amplia difusión y orientados a la educación. El soporte bajo otros sistemas al momento de la entrega se dará por medio de herramientas de virtualización gratuitas, para lo cual se proveerán instrucciones.

- La elaboración de casos de aplicación que demuestren las características y el uso del sistema.  
Se proponen las siguientes:

- a) Caracterización de modulaciones digitales mediante análisis de Montecarlo.
- b) Estudio del problema gravitatorio de n-cuerpos para un conjunto de objetos de masas, posiciones y velocidades aleatorias, con agregación de materia y con comparación de los resultados bajo diferentes técnicas numéricas.
- c) Implementación fácil de un sistema de estereoscopía para la didáctica de la Geometría Analítica y el Cálculo Vectorial empleando la visualización en tercera dimensión.

- La documentación asociada para uso del sistema.
- La publicación y liberación del sistema.

No hacen parte del alcance del proyecto:

- Garantizar compatibilidad con ambientes de programación diferentes al lenguaje C.
- Re-implementar, en estilo o contenido, productos existentes, o emularlos.
- El desarrollo de componentes, "widgets", o de utilidades para elaborar interfaces gráficas de usuario (GUI's).
- El desarrollo de controladores específicos, por ejemplo, para dispositivos de entrada/salida.
- La migración a otras plataformas de hardware.

## 6.2. Impactos y contribuciones

- 6.2.1. Este proyecto contribuye a hacer más asequible la computación científica y a difundirla.
- 6.2.2. En el sector empresarial, es posible que en algunos casos evite la compra innecesaria de software y a que disminuya el tiempo de desarrollo de productos. Además de no tener costo, el sistema propuesto no impondrá condiciones exigentes respecto a trabajos derivados.
- 6.2.3. En el sector académico, el sistema podría ser material de apoyo en programas de ingeniería en asignaturas como: Geometría Analítica, Álgebra Lineal, Cálculo Diferencial, Cálculo Integral, Física Estática, Física Dinámica, Electromagnetismo, Programación de Computadores, Simulación, Sistemas y Señales, Control, o como herramienta de apoyo en investigaciones.
- 6.2.4. Finalmente, el sector de desarrolladores independientes o aficionados de productos de software electrónicos pueden aplicar el sistema (o partes del mismo) a sus proyectos.

## 7. PRODUCTOS ESPERADOS

El resultado concreto es software, a nivel de Prototipo Funcional, sus librerías, su código fuente, ejemplos y documentación. Estos materiales se harán publicarán en una página web o en GitHub.

## 8. CRONOGRAMA

El proyecto de simulación minimalista se origina en desarrollos e investigaciones elaboradas por el proponente, por interés personal en los temas desde su pregrado, y del que se ha consolidado un material substancial. Estos avances no han sido publicados ni utilizados en programas académicos. Se aclara que no es un proyecto de industria, dado que no ha sido solicitado por el empleador al proponente; no está asociado a ningún producto o servicio de la compañía; no hace uso de "know how", información o saberes específicos del empleador; no se desarrolla con herramientas o instrumentos del empleador; y tampoco hay un escenario de competencia entre el proyecto y algún producto o servicio ofrecido por la compañía. En breve: este es un proyecto de software esencialmente matemático aplicable a cualquier campo.

El proponente aclara que ha hecho en forma ocasional uso de módulos preliminares propios, no conformados como sistema completo, para apoyar algunos estudios técnicos relacionados con su trabajo, los cuales también se hubieran podido hacer con productos tipo Matlab. Estas pruebas no quieren decir que el proyecto sea de industria, sino que indican que un sistema como el propuesto, una vez terminado, podría ser aplicable a desarrollos industriales.

El cronograma propuesto a continuación indica las actividades aun por desarrollar, las cuales se espera tengan una duración aproximada de 600 horas, y es relativo al visto bueno de inicio del proyecto.

Actividad	Mes 1	Mes 2	Mes 3	Mes 4	Mes 5	Mes 6
Definición de Filosofía y Modelo						
Implementación, énfasis en librería 2D						
Implementación, énfasis en librería 3D						
Implementación, énfasis en el manejo dinámico de arreglos de 1, 2, y múltiples dimensiones						
Implementación, énfasis en Módulos de Evaluación Funcional						
Implementación, énfasis en Integración						
Implementación, énfasis en aspectos de Hardware						
Elaboración de ejemplo de simulación (tipo académico).						
Elaboración de ejemplo de simulación (tipo industrial).						
Elaboración informe escrito y de guía de usuario.						

Tabla 2 Cronograma propuesto

## 9. PRESUPUESTO

Un objetivo del proyecto es que no tenga costo, y como tal se desarrolla en sistemas con software libre y sin instrumentos especializados. Por consiguiente, no se considera necesaria la adquisición de elementos como software. El desarrollo se realiza con computadores sencillos o máquinas virtuales ejecutadas en ellos. También se utilizará un sistema Raspberry Pi para verificar compatibilidad y evaluar el desempeño. El mayor componente del proyecto es el tiempo necesario para la investigación y desarrollo directos, los cuales son parte del esfuerzo académico que se espera por parte del estudiante y el tiempo de apoyo por parte del Director.

Recurso	Cantidad	Provisto por el Estudiante	Provisto por la Universidad
Horas de trabajo de Estudiante	500	X	
Horas de trabajo del Director	80		X
Sistema Linux para el Director (puede ser un sistema dedicado o una Máquina Virtual)	1		X
Sistema Linux Estudiante	1	X	
Sistemas Raspberry Pi 3 B+	2	X	
Acceso a Internet		X	X
Papelería e Impresión	Según se requiera	X	
Software de oficina (Microsoft Office)		X	

Tabla 3 Recursos para la elaboración del proyecto

Para la realización del proyecto los desembolsos son menores y corren a cargo del estudiante.

## 10. REFERENCIAS BIBLIOGRÁFICAS

- ALIAGA, J., ALMEIDA, F., BADIA, J. M., BARRACHINA, S., BLANCO, V., CASTILLO, M., DORTA, U., et al. (2004). *Parallelization of the GNU Scientific Library on Heterogeneous Systems*. Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks. Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/ISPDC.2004.39>
- BELLETTI, F., COTALLO, M., CRUZ, A., FERNANDEZ, L. A., GORDILLO-GUERRERO, A., GUIDETTI, M., MAIORANO, A., et al. (2009). *Janus: An FPGA-Based System for High-Performance Scientific Computing*. Computing in Science & Engineering, 11(1), pp. 48-58. Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/MCSE.2009.11>
- BEZANSON, J., EDELMAN, A., KARPINSKI, S. and SHAH, Viral. (2017). *A fresh approach to Numerical Computing*. SIAM review, 59(1), pp. 65–98. Society for Industrial and Applied Mathematics. Recuperado a partir de <https://doi.org/10.1137/141000671>.
- CASQUILHO, M., & CUNHA, M. (2014). *Scientific computing over the Web in various programming languages: Solving problems in Fortran, C, and Octave*. 2014 9th Iberian Conference on Information Systems and Technologies (CISTI). Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/CISTI.2014.6877031>
- CHEN, P., DONG, X. H., & ZHOU, X. (2009). *Design and implementation of the communication experiments based on open source software SCILAB/SCICOS*. 2009 IEEE International Workshop on Open-source Software for Scientific Computation (OSSC). Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/OSSC.2009.5416737>
- HIMMELSPACH, J. (2009). *Toward a Collection of Principles, Techniques, and Elements of Modeling and Simulation Software*. 2009 First International Conference on Advances in System Simulation. Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/SIMUL.2009.19>

- HINSEN, K. (2013). *Daydreaming about Scientific Programming*. Computing in Science & Engineering, 15(5), pp. 77-79. Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/MCSE.2013.104>
- KELLY, D. F. (2007). *A Software Chasm: Software Engineering and Scientific Computing*, IEEE Software, 24(6), pp. 120-119. Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/MS.2007.155>
- KURZAK, J., BUTTARI, A., LUSCZEK, P., & DONGARRA, J. (2008). *The PlayStation 3 for High-Performance Scientific Computing*. Computing in Science & Engineering, 10(3), pp. 84-87. Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/MCSE.2008.85>
- MACISAAC, D., C. CHAN, A. (2006). *Sound Software Development for Engineering Simulations*. 2006 Canadian Conference on Electrical and Computer Engineering. Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/CCECE.2006.277497>
- NANCE, R. E., & OVERSTREET, C. M. (2017). *History of computer simulation software: An initial perspective*. 2017 Winter Simulation Conference (WSC). Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/WSC.2017.8247792>
- OLIPHANT, T. E. (2007). *Python for Scientific Computing*. Computing in Science & Engineering, 9(3), pp. 10-20. Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/MCSE.2007.58>
- PIDD, M. (2002). *Simulation software and model reuse: a polemic*. Proceedings of the Winter Simulation Conference. Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/WSC.2002.1172959>
- ROUGHAN, M. (2019). *Practically surreal: Surreal arithmetic in Julia*. RC Centre of Excellence for Mathematical & Statistical Frontiers (ACEMS). School of Mathematical Sciences, University of Adelaide. Recuperado a partir de <https://www.sciencedirect.com/science/article/pii/S2352711018302152>
- SMIT, W., & HERBST, B. (2014). *Mobile Computers as Scientific Computing Machines*. 2014 IEEE International Conference on High Performance Computing and Communications. Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/HPCC.2014.214>
- TANG, H., SHE, R., HE, C., & DOU, Y. (2008). *Construction and Application of Linux Virtual Server Cluster for Scientific Computing*. 2008 IFIP International Conference on Network and Parallel Computing. Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/NPC.2008.53>
- TERREL, A. R. (2011). *From Equations to Code: Automated Scientific Computing*. Computing in Science & Engineering, 13(2), pp. 78-82. Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/MCSE.2011.31>
- TURK, M. (2015). *Vertical Integration*. Computing in Science & Engineering, 17(1), 64-66. Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/MCSE.2015.27>
- VOLKEMA, G., & KHANNA, G. (2017). *Scientific computing using consumer video-gaming embedded devices*. 2017 IEEE High Performance Extreme Computing Conference (HPEC). Institute of Electrical and Electronics Engineers (IEEE). Recuperado a partir de <http://dx.doi.org/10.1109/HPEC.2017.8091055>
- ZHANG, H., & NIE, J. (2016). *Program performance test based on different computing environments*. 2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS). Institute of

Electrical and Electronics Engineers (IEEE). Recuperado a partir de  
<http://dx.doi.org/10.1109/ICOACS.2016.7563073>

## 11. DECLARACIÓN DE ORIGINALIDAD

El contenido de este documento fue elaborado en forma propia, original y autónoma por el suscrito. Todo trabajo por parte de terceros que esté relacionado con este anteproyecto ha sido citado indicando la fuente. Ninguna parte del contenido ha sido utilizada en forma anterior en ningún programa académico en Colombia o en el exterior.

## 12. NOTAS SOBRE ASPECTOS LEGALES Y DE PROPIEDAD INTELECTUAL

La evaluación por parte del primer Sr. (o Sra.) Jurado indica una posible preocupación respecto a avances en el proyecto y a posibles inconvenientes legales por la concurrencia con mi empleo. Quiero brindar información adicional sobre el tema:

### 12.1. Sobre aspectos legales y la relación del proyecto con mi actividad laboral

Mi empleador desarrolla sistemas de telemedida para aplicación en la industria del petróleo, objeto para el cual fui contratado y el cual no ha tenido cambios en cinco años. Dentro del curso del trabajo, naturalmente hay contacto con información específica de productos, de tecnologías concretas que están desarrollando o en las que se desea avanzar, y en general "know how" o propiedad intelectual que le pertenece a la empresa y a la cual modestamente trato de contribuir en mi calidad de empleado. He sido cuidadoso de que mi propuesta de Proyecto de Grado no toque en absoluto ningún saber específico de mi empresa y de hecho la propuesta no incluye ni menciona ninguna marca, producto o tecnología de mi empleador. Quiero desatacar los siguientes puntos:

- a) La propuesta es de tipo estrictamente académico en el ámbito de la computación científica y gráfica. Mi empleador no desarrolla ni comercializa sistemas numéricos, gráficos, ni de cálculo científico, sino que hace uso de productos comerciales tipo Matlab y Mathcad.
- b) En ocasiones, sí he aplicado material de desarrollo propio o estudios hechos por interés personal a la solución de problemas de empresa. Por decirlo de algún modo, es lo que ocurre cuando un trabajador lleva a la compañía donde labora una herramienta o instrumento hechos por él mismo porque desea y le es permitido probarlos, o porque estos tienen alguna característica que los haga aplicables al problema en cuestión.
- c) Deseo aclarar además que el ámbito laboral en el que me desenvuelvo es en general flexible y se basa en el cumplimiento de metas. Mientras que estas sean satisfechas, existe cierto grado de libertad para explorar intereses, estudiar, o realizar otras actividades. Se hace esta aclaración para diferenciar mis estudios o avances en el tema propuesto como Proyecto de Grado de mi trabajo de empresa.

### 12.2. Sobre avances preexistentes

- a) El proyecto de computación científica minimalista no empezó como proyecto empresarial. Lo que ocurre es que temas como el cálculo numérico, la computación gráfica, la programación y la simulación han sido de especial interés para el proponente desde el pregrado (participación en Ingeniería 94 y 95; participación en 1995 en la conferencia de SPIE, The International Society for Optics and Photonics); y también a lo largo de la Maestría. Efectivamente, hay bloques que vistos de forma individual presentan avances (por ejemplo, en la evaluación de matemáticas literales; en la implementación de métodos numéricos; o en el análisis de Fourier), pero nunca se ha puesto a punto un sistema completo, y de hecho no se ha escrito, dibujado, ni definido un modelo.

- b) Declaré la existencia de avances en aras de acreditar que el proyecto se pudiera culminar en 6 meses y que no fuera rechazado por requerir un periodo mucho más largo. En un sistema como el propuesto, algunos módulos son bastante dispensiosos, y constituyen mini-proyectos en sí mismos. Durante los años 2015 y 2016, principalmente, tuve un periodo de menor ocupación laboral durante el cual pude desarrollar algunos temas de interés, incluyendo algunos relacionados con la propuesta, pensando en avanzar en ellos como Proyecto de Grado y enlazarlos. Durante ese tiempo, además, inicié el proceso para reingresar a la Maestría y quise llegar con algunos elementos delineados. Esto no quiere decir que el modelo o producto estén definidos, resueltos o que haya un producto terminado. Simplemente, hay avances similares a las notas que tenga un escritor antes de iniciar una novela, o a las de un docente antes de iniciar una investigación.
- c) El Sistema de Cómputo Minimalista no me ha sido solicitado dentro de mi trabajo; no está asociado como todo o como parte a algún producto o servicio ofrecido por mi empleador; no utiliza ningún saber propietario de este; no emula o imita a ninguno de sus productos o servicios, ni tampoco compite con los mismos; no involucra a ningún otro empleado; no se ha desarrollado con uso de instrumentos especializados que sean propiedad de la empresa; y ni siquiera está asociado a la industria del petróleo. Se trata de una herramienta de uso general aplicable a cualquier área de la ingeniería.
- d) El Reglamento Estudiantil de Formación Avanzada no objeta que preexistan avances en algún tema que se vaya a desarrollar como Proyecto de Grado. Lo que señala es que el trabajo debe ser pertinente, original, y que tenga impacto [Artículo 84]. También indica, como debe ser, que no haya sido utilizado ("bajo la misma forma o con variaciones") para optar a un título en cualquier Universidad [Artículo 92]. Todas estas condiciones se cumplen. Se respeta profundamente el comentario del Sr. o Sra. Jurado porque es importante asegurar y mantener la integridad académica.

## Tabla de Contenido

1.	Introducción .....	3
2.	Modelo de Software .....	5
2.1.	Sistemas gráficos bajo Unix/Linux y Windows .....	5
2.2.	Concepto y diagrama de capas del sistema de Simulación finalmente elegido.....	8
2.3.	Elección de SDL (Simple DirectMedia Layer) como opción para un sistema de simulación.....	10
2.4.	Ubicación y uso de las librerías.....	12
2.5.	Otros modelos de software evaluados, desarrollos y resultados .....	12
2.6.	Modelo basado en X-Window (descartado) .....	13
2.7.	Modelo basado en el controlador “Framebuffer” .....	15
3.	Instalación del sistema.....	19
4.	Notación y convenciones.....	20
5.	Casos de uso .....	21
5.1.	Caso de uso: el problema de los n-cuerpos.....	21
5.1.1.	Algoritmo para solución por el Método de Euler: .....	24
5.1.2.	Algoritmo para solución por el método de Runge-Kutta 4.....	26
5.1.3.	Análisis del error y comparación entre métodos.....	31
5.2.	Caso de uso: estudio dinámico de mecanismo de levas.....	33
5.3.	Caso de uso: caracterización de modulaciones digitales arbitrarias utilizando análisis de Monte Carlo .....	40
5.3.1.	Caracterización de la modulación por medio de una simulación .....	43
5.3.2.	Resultados: .....	46
5.4.	Caso de uso: control para un sistema de pruebas de impacto para componentes electrónicos .....	47
5.4.1.	Algoritmo simplificado para validación de operación ante impacto: .....	49
5.4.2.	Características del proyecto aplicables a la solución de problemas industriales: .....	50
5.4.3.	Otras aplicaciones: .....	51
5.4.4.	Desempeño del GPIO: .....	51
5.4.5.	Problemas encontrados:.....	53
6.	Tipos y Estructuras de Datos.....	55
7.	Librería Algebraica “lb_algebra.c” .....	73

8.	Librería de Análisis de Fourier “lb_fourier.c” .....	79
9.	Librería de funciones en variable real “lb_real.c” .....	80
10.	Librería de Funciones Estadísticas “lb_statistics.c”.....	83
11.	Funciones de variable compleja “lb_complex.c” .....	85
12.	Librería geométrica “lb_geometry.c” .....	87
13.	Librería para uso del GPIO (General Purpose Input and Output) Puerto de Propósito General de Entrada y Salida “lb_gpio.c” .....	88
14.	Librería gráfica “lb_graphics.c” .....	92
15.	Librerías de evaluación funcional “lb_parser.c” .....	119
16.	Extensiones, mejoras y proyectos asociados: .....	125
16.1.	Diseño y elaboración de un módulo para uso simplificado del sonido.....	125
16.2.	Soporte de interrupciones asociadas al GPIO en sistemas Raspberry Pi.....	125
16.3.	Mejoras al sistema gráfico.....	126
16.4.	Proyecto asociado: Montaje de un sistema de realidad virtual de bajo costo. ....	126
16.5.	Proyecto asociado: Desarrollo de hardware modular para instrumentación industrial. ....	128
16.6.	Proyecto Asociado: Desarrollo de un sistema magnificador para personas con limitación visual.....	129
17.	Conclusiones .....	132
18.	Bibliografía .....	134
19.	Anexo No. 1 (publicación ya no disponible) .....	136
20.	Anexo No. 2. Anteproyecto. ....	139