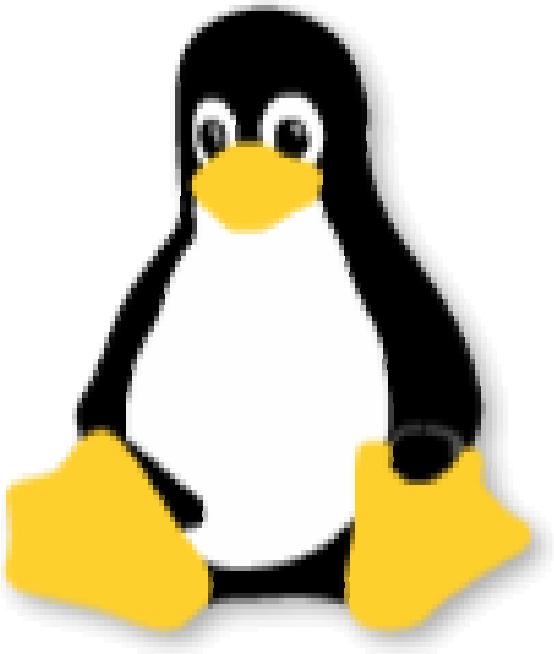


MANUAL ALLEGRO

ÍNDICE

Pag.

USANDO ALLEGRO

- **install_allegro**

```
int install_allegro(int system_id, int *errno_ptr, int (*atexit_ptr)());
```

Inicializa Allegro. Debe llamar a esta función o a allegro_init() antes de hacer otra cosa. Los identificadores de sistema disponibles (system_id) variarán dependiendo de la plataforma utilizada, pero casi siempre querrá usar SYSTEM_AUTODETECT. Alternativamente, SYSTEM_NONE instalará una versión reducida de Allegro que no intentará realizar accesos directos al hardware ni operaciones que sean particulares a una plataforma específica: esto puede resultar útil para situaciones en las que solamente quiera manipular bitmaps de memoria, como en una utilidad que maneje ficheros de datos o las funciones de interfaz con el GDI de Windows. Los parámetros errno_ptr y atexit_ptr deben apuntar a la variable errno y la función atexit() de su librería C respectivamente. Estos parámetros son requeridos ya que cuando Allegro es enlazado como una DLL no tiene acceso directo a los datos locales de su librería C. Por ahora esta función siempre devuelve cero. Si no se puede usar ningún driver de sistema, el programa será abortado.

- **allegro_init**

```
int allegro_init();
```

Inicializa Allegro. Esta función es equivalente a la llamada install_allegro(SYSTEM_AUTODETECT, &errno, atexit).

- **allegro_exit**

```
void allegro_exit();
```

Cierra el sistema Allegro. Esto incluye devolver el sistema al modo texto y desinstalar todas las rutinas de ratón, teclado o temporización que estuviesen instaladas. Normalmente no tiene que molestarse en hacer una llamada explícita a esta función, ya que allegro_init() la instala como una función atexit(), por lo que será llamada automáticamente cuando su programa finalice.

- **allegro_id**

```
extern char allegro_id[];
```

Cadena de texto que contiene la fecha y número de versión de Allegro, en caso de que quiera enseñar estos datos en alguna parte.

- **allegro_error**

```
extern char allegro_error[ALLEGRO_ERROR_SIZE];
```

Cadena de texto usada por set_gfx_mode() e install_sound() para indicar mensajes de error. Si éstas funciones fallan y quiere decirle al usuario porqué, éste es el lugar en el que se encuentra una descripción del problema.

- **os_type**

```
extern int os_type;
```

Variable ajustada por allegro_init() a uno de los siguientes valores:

OSTYPE_UNKNOWN	- desconocido, o MSDOS normal
OSTYPE_WIN3	- Windows 3.1 o versiones anteriores
OSTYPE_WIN95	- Windows 95
OSTYPE_WIN98	- Windows 98
OSTYPE_WINME	- Windows ME
OSTYPE_WINNT	- Windows NT
OSTYPE_WIN2000	- Windows 2000
OSTYPE_WINXP	- Windows XP
OSTYPE_OS2	- OS/2
OSTYPE_WARP	- OS/2 Warp 3
OSTYPE_DOSEMU	- Linux DOSEMU
OSTYPE_OPENDOS	- Caldera OpenDOS
OSTYPE_LINUX	- Linux
OSTYPE_FREEBSD	- FreeBSD
OSTYPE_UNIX	- variante Unix desconocida
OSTYPE_BEOS	- BeOS
OSTYPE_QNX	– QNX
OSTYPE_MACOS	- MacOS

- **os_version**

```
extern int os_version;
extern int os_revision;
```

La versión mayor y menor del Sistema Operativo usado. Ajustado por allegro_init(). Si Allegro por alguna razón no es capaz de obtener la versión de su Sistema Operativo, os_version y os_revision valdrán -1. Por ejemplo: bajo Win98 SE (v4.10.2222) os_version valdrá 4 y os_revision valdrá 10.

- **os_multitasking**

```
extern int os_multitasking;
```

Ajustado por allegro_init() a TRUE o FALSE dependiendo de si su Sistema Operativo es multitarea o nó.

- **allegro_message**

```
void allegro_message(char *msg, ...);
```

Muestra un mensaje usando una cadena con formato printf(). Esta función sólo debe ser utilizada cuando no esté en un modo gráfico, es decir, antes de llamar a set_gfx_mode() o después de llamar a set_gfx_mode(GFX_TEXT). En plataformas que tengan una consola en modo texto (DOS, Unix y BeOS), mostrará la cadena en la consola reduciendo los carácteres acentuados a aproximaciones en 7 bits de códigos ASCII, evitando en lo posible los problemas con las diferentes páginas de códigos. Bajo Windows, el mensaje aparecerá en un ventana de mensajes.

- **set_window_title**

```
void set_window_title(const char *name);
```

En las plataformas que sean capaces de ello, esta rutina cambia el título de la ventana utilizada para su programa. Tenga en cuenta que Allegro no es capaz de modificar el título de la ventana cuando ejecute una ventana DOS en Windows.

- **set_window_close_button**

```
int set_window_close_button(int enable);
```

En las plataformas que sean capaces de ello, esta rutina activa o desactiva el botón de cerrar ventana de su programa Allegro. Si lo desea, puede llamar esta rutina antes de que la ventana sea creada. Si el botón de cerrar ventana es desactivado con éxito, esta función devolverá cero.

En plataformas en las que el botón de cerrado no existe o no puede ser desactivado, la función devolverá -1. Si esto ocurre, quizás le interese usar set_window_close_hook() para manejar manualmente el evento de cierre de la ventana.

Cuando active el botón de cierre, la función devolverá el mismo valor que devolvió su plataforma al desactivarlo. Esto significa que devolverá distinto de cero si el botón no se puede desacivar, a pesar de que no esté intentando desactivarlo.

Tenga en cuenta que Allegro no puede manipular el botón de cerrado de una ventana DOS bajo Windows.

- **set_window_close_hook**

void set_window_close_hook(void (*proc)());

En las plataformas que tengan botón de cierre, esta rutina instala una función de enganche en el evento de cierre. En otras palabras, cuando el usuario pinche en el botón que cerraría la ventana de su programa, se llamará a la función que especifique aquí. Puede usar esta función para mostrar un diálogo ofreciendo salvar los datos o verificar que el usuario realmente desea salir, o puede usarla para salvar los datos, liberar memoria y salir.

Esta función generalmente no debería intentar salir del programa o salvar datos por sí misma. Esta función podría ser llamada en cualquier momento, y hay peligro de que los datos que intente salvar sean inválidos en ese momento. Por eso, debería activar una variable durante esta función, y verificar esta variable regularmente en su programa principal.

Pase NULL a esta función para recuperar la funcionalidad por defecto del botón de cierre. En Windows y BeOS, el siguiente mensaje aparecerá:

Aviso: forzar la finalización del programa puede ocasionar pérdidas de datos y resultados inesperados. Es preferible que use el comando de salir dentro de la ventana.

¿Desea continuar de todas maneras?
[Si] [No]

Este mensaje será traducido al lenguaje seleccionado si hay una traducción dispobible en language.dat (vea get_config_text()).

Si el usuario selecciona [Si], el programa finalizará inmediatamente en el mismo estilo que cuando el usuario pulsa Ctrl+Alt+End (vea three_finger_flag).

En otros sistemas operativos, el programa saldrá inmediatamente sin preguntar nada al usuario.

Tenga en cuenta que Allegro no puede interceptar el botón de cierre de una ventana DOS bajo Windows.

- **desktop_color_depth**

int desktop_color_depth();

En plataformas que pueden ejecutar programas de Allegro en una ventana de un escritorio existente, devuelve la profundidad de color usada por el escritorio en ese momento (su programa posiblemente correrá más si usa la misma profundidad de color. En plataformas donde esta información no es disponible o no tiene sentido, devuelve cero.

- **get_desktop_resolution**

```
int get_desktop_resolution(int *width, int *height);
```

En plataformas que pueden ejecutar programas de Allegro en una ventana de un escritorio existente, permite obtener la resolución actual usada por el escritorio (ej: le itneresará llamar a esta función antes de crear una gran ventana, porque en algunos drivers de modo ventana, ésta no puede ser creada si es mayor que el escritorio). Devuelve cero si hubo éxito, o un número negativo si la información no está disponible o no es aplicable a su situación, en cuyo caso los valores almacenados en width y heigth serán indefinidos.

- **yield_timeslice**

```
void yield_timeslice();
```

En los sistemas que lo soportan, libera el resto de la "rebanada temporal" (timeslice) que la CPU le había asignado. Esta opción también es conocida como "pórtate bien con la multitarea".

- **check_cpu**

```
void check_cpu();
```

Detecta el tipo de CPU, asignando valores a las siguientes variables globales. Normalmente no necesita llamar a esta función, ya que allegro_init() lo hará por usted.

- **cpu_vendor**

```
extern char cpu_vendor[];
```

Contiene el nombre del proveedor de la CPU si éste es conocido. En plataformas no-Intel, contiene una cadena vacía.

- **cpu_family**

```
extern int cpu_family;
```

Contiene el tipo de CPU Intel, en las CPUs donde sea aplicable: 3=386, 4=486, 5=Pentium, 6=PPro, etc.

- **cpu_model**

```
extern int cpu_model;
```

Contiene el submodelo de una CPU Intel, en las CPUs donde sea aplicable. En un 486 (cpu_family=4), cero o uno indica un chip DX, 2 un SX, 3 indica la presencia de un coprocesador matemático (486 SX + 487 ó 486 DX), 4 un SL, 5 un SX2, 7 un DX2 write-back enhanced, 8 un DX4 o un overdrive DX4, 14 un Cyrix y 15 desconocido. En un chip Pentium (cpu_family=5), 1 indica un Pentium (510\66, 567\66), 2 un Pentium P54C, 3 un procesador Pentium overdrive, 5 un Pentium overdrive para IntelDX4, 14 un Cyrix y 15 desconocido.

- **cpu_capabilities**

```
extern int cpu_capabilities;
```

Contiene bits de la CPU que indican qué características están disponibles. Los bits pueden ser una combinación de:

CPU_ID	-Indica que la instrucción "cpuid" está disponible. Si este bit está activo, entonces todas las variables CPU de Allegro son fiables al 100%, en caso contrario podría haber fallos.
CPU_FPU	-Hay disponible una FPU x87.
CPU_MMX	-Conjunto de instrucciones Intel MMX disponible.
CPU_MMXPLUS	-Conjunto de instrucciones Intel MMX+ disponible.
CPU_SSE	-Conjunto de instrucciones Intel SSE disponible.
CPU_SSE2	-Conjunto de instrucciones Intel SSE2 disponible.
CPU_3DNOW	-Conjunto de instrucciones AMD 3DNow! disponible.
CPU_ENH3DNOW	-Conjunto de instrucciones AMD Enhanced 3DNow! disponible.
CPU_CMOV	-Instrucción "cmov" del Pentium Pro disponible.

Puede verificar múltiples características haciendo una OR de los bits. Por ejemplo, para ver si la CPU tiene una FPU y un conjunto de instrucciones MMX podría hacer:

```
if ((cpu_capabilities & (CPU_FPU | CPU_MMX)) == (CPU_FPU | CPU_MMX))
      printf("iLa CPU tiene tanto una FPU como instrucciones MMX!\n");
```

RUTINAS UNICODE

Allegro puede manipular y mostrar texto usando cualquier carácter en el rango que va de 0 a 2^32-1 (aunque por ahora grabber sólo puede crear fuentes usando carácteres hasta 2^161). Puede escoger entre varios tipos de formatos de codificación de texto, los cuales controlan la forma en la que las cadenas son almacenadas y cómo Allegro interpreta las que usted le pase. Esta configuración afecta a todos los aspectos del sistema: cada vez que observe una función que devuelve un puntero a carácter (char *) o que toma un puntero a carácter como argumento, el texto utilizará el formato que se le haya indicado a Allegro.

Por defecto, Allegro usa texto codificado en el formato UTF-8 (U_UTF8). Este es un formato de anchura variable donde los carácteres pueden tener cualquier tamaño de 1 a 6 bytes. Lo bueno de este formato es que los carácteres de 0 a 127 pueden ser almacenados directamente, o lo que es igual, significa que es compatible con códigos ASCII de 7 bits ("Hola, Mundo!" es exactamente la misma cadena en ASCII que en UTF-8). Cualquier carácter por encima del 128 como vocales acentuadas, el símbolo monetario inglés o carácteres árabes y chinos serán codificados como una secuencia de dos o más bytes con un valor en el rango 128-255. Esto permite que al mostrar la cadena no se obtengan carácteres extraños en ASCII que en realidad forman parte de la codificación de un carácter con diferente valor, lo que hace realmente fácil manipular cadenas UTF-8.

Existen algunos editores de texto que entienden ficheros UTF8, pero alternativamente, puede escribir sus cadenas en ASCII plano o en formato Unicode de 16 bit y luego utilizar el programa textconv (suministrado con Allegro) para convertir el texto a UTF-8.

Si prefiere usar otros formatos de texto, Allegro permite la utilización de formatos ASCII de 8 bits (U_ASCII) o Unicode de 16 bits (U_UNICODE). Además puede programar sus propias funciones para manejar otros formatos de texto con Allegro (sería sencillo añadir soporte para el formato UCS-4 de 32 bits, o el formato GB-code chino).

También existe soporte limitado para páginas de códigos alternativas de 8 bits a través del modo U_ASCII_CP. Lo malo de este modo es que es muy lento y no debería utilizarse para aplicaciones serias. Sin embargo, puede utilizarse para convertir texto fácilmente entre diferentes páginas de códigos. Por

defecto, el modo U_ASCII_CP es activado para convertir texto al formato ASCII de 7 bits, convirtiendo las vocales acentuadas en sus equivalente (por ejemplo, allegro_message() utiliza este modo para mostrar los mensajes en una consola DOS). Si necesita trabajar con otras páginas de códigos, puede hacerlo pasando un mapeado de carácteres a la función set_ucodepage().

- **set_uformat**

void set_uformat(int type);

Establece el formato de codificación de texto a utilizar. Esta operación afectará a todos las funciónes de Allegro que devuelvan un puntero a carácter (char *) o acepten un puntero a carácter como parámetro. El parámetro type debe ser uno de los siguientes:

U_ASCII	-carácteres ASCII de 8 bits de tamaño fijo
U_ASCII_CP	-página de códigos alternativa de 8 bits (ver set_ucodepage())
U_UNICODE	-carácteres Unicode de 16 bits de tamaño fijo
U_UTF8	-carácteres Unicode UTF-8 de tamaño variable

Aunque es posible cambiar el formato de texto en cualquier momento, no es una práctica demasiado recomendable. Muchas cadenas (entre ellas los nombres de los drivers de hardware y algunas traducciones) son inicializadas en la llamada a allegro_init(), por lo tanto, si se cambia el formato de texto tras dicha llamada, las cadenas estarán en un formato diferente al seleccionado y el sistema no funcionará de forma apropiada. Como normal general, sólo debería llamarse a set_uformat() una vez, antes de llamar a allegro_init(), y utilizar el formato de texto seleccionado durante toda la ejecución de su programa.

- **get_uformat**

int get_uformat(void);

Devuelve el formato de codificación de texto actualmente seleccionado.

- **register_uformat**

void register_uformat(int type, int (*u_getc)(const char *s), int (*u_getx)(char **s), int (*u_setc)(char *s, int c), int (*u_width)(const char *s), int (*u_cwidth)(int c), int (*u_isok)(int c));

Instala un conjunto de funciones para el manejo de un nuevo formato de codificación de carácteres. El parámetro type identifica el nuevo formato, que debería ser una cadena de 4 carácteres como las producidas por la macro AL_ID(). Esta cadena será la que se pase posteriormente a funciones como

set_uformat() y uconvert(). Los punteros a funciones pasados como parámetros, son manejadores que implementan el acceso a los carácteres de una cadena formateada con la nueva codificación (vea más abajo para más detalles).

- **set_ucodepage**

void set_ucodepage(const unsigned short *table, const unsigned short *extras);

Cuando se selecciona el modo U_ASCII_CP, los carácteres de 8 bits son convertidos a sus equivalentes en Unicode a través de un conjunto de tablas. Se puede usar esta función para especificar un conjunto de tablas personalizadas que permitan la utilización de páginas de códigos de 8 bits alternativas. El parámetro table apunta a un array de 256 enteros short que contienen el valor Unicode para cada carácter en la página de códigos. El parámetro extras, si no es NULL, apunta a una lista de pares de valores que es utilizada para realizar la conversión inversa, es decir, reducir los valores Unicode a la representación correcta dentro de la nueva página de codigos. Cada par de valores consiste en un valor Unicode seguido por la forma de representación correspondiente dentro de la página de códigos. La tabla debe finalizar con el valor cero en Unicode. Esta tabla permite que varios carácteres Unicode puedan representarse mediante un solo valor en la página de codigos (por ejemplo para reducir vocales acentuadas a ASCII de 7 bits).

- **need_uconvert**

int need_uconvert(const char *s, int type, int newtype);

Dado un puntero a una cadena, el tipo de la cadena y el tipo al que se desea convertir, esta función indica si dicha conversión es necesaria. La conversión no es necesaria si type y newtype son el mismo tipo o son tipos equivalentes (por ejemplo ASCII y UTF-8 y la cadena contiene carácteres menores que 128). Como valor para uno de los parámetros type, se puede pasar U_CURRENT que representa el tipo actualmente seleccionado.

- **uconvert_size**

int uconvert_size(const char *s, int type, int newtype);

Devuelve el número de bytes que serán necesarios para almacenar la cadena especificada tras una conversión al nuevo tipo, incluyendo el carácter terminador nulo. Los parámetros type pueden usar el valor U_CURRENT para indicar el tipo actualmente seleccionado.

- **do_uconvert**

void do_uconvert(const char *s, int type, char *buf, int newtype, int size);

Convierte la cadena especificada del tipo type al tipo newtype, guardando como mucho size bytes en el buffer buf. Los parámetros type pueden utilizar el valor U_CURRENT para indicar el tipo actualmente seleccionado.

- **uconvert**

```
char *uconvert(const char *s, int type, char *buf, int newtype, int size);
```

Esta es una función de alto nivel que sirve como ayuda para ejecutar do_uconvert(). Al igual que do_uconvert(), convierta la cadena especificada del tipo type al tipo newtype, guardando como mucho size bytes en el buffer buf. La ventaja que obtenemos a usar uconvert() es que comprueba los tipos antes de realizar la conversión, para asegurarse de que son tipos diferentes, no realizando conversión alguna si los tipos son iguales o equivalentes. Si la conversión fue realizada, devuelve un puntero a buf, en caso contrario, devuelve una copia de la cadena a convertir (s). Por lo tanto, debe usar el valor devuelto en lugar de asumir que la cadena ha sido movida a buf. Si buf es NULL, la cadena será convertida en un buffer estático interno. Sin embargo, debería tener cuidado al usar este comportamiento, ya que el buffer será sobreescrito cada vez que la rutina sea invocada, por lo que no espere que los datos persistan tras haber hecho llamadas a otras funciones de la biblioteca.

- **uconvert_ascii**

```
char *uconvert_ascii(const char *s, char buf[]);
```

Macro auxiliar para convertir cadenas desde ASCII al formato actual de codificación. Se expande a uconvert(s, U_ASCII, buf, U_CURRENT, sizeof(buf)).

- **uconvert_toascii**

```
char *uconvert_toascii(const char *s, char buf[]);
```

Macro auxiliar para convertir cadenas desde el formato actual de codificación a ASCII. Se expande a uconvert(s, U_CURRENT, buf, U_ASCII, sizeof(buf)).

- **empty_string**

```
extern char empty_string[];
```

No se puede fiar de que "" sea una cadena vacía válida en todos los formatos de codificación. Este búffer global contiene un número de ceros consecutivos, así que siempre será una cadena vacía válida, sin tener importancia si el programa se está ejecutando en modo ASCII, Unicode o UTF-8.

- **ugetc**

int ugetc(const char *s);

Función auxiliar de bajo nivel para leer datos de texto en Unicode. Dado un puntero a una cadena en el formato de codificación actual devuelve el siguiente caracter de la cadena.

- **ugetx**

int ugetx(char **s); int ugetxc(const char **s);

Función auxiliar de bajo nivel para leer datos de texto en Unicode. Dada la dirección de un puntero a un string en el formato de codificación actual devuelve el siguiente caracter de la cadena y avanza el puntero al siguiente caracter.

ugetxc es para trabajar con datos char constantes puntero a puntero.

- **usetc**

int usetc(char *s, int c);

Función auxiliar de bajo nivel para escribir datos de texto en Unicode. Escribe el caracter especificado en la dirección dada usando el formato de codificación actual y devuelve el número de bytes escritos.

- **uwidth**

int uwidth(const char *s);

Función auxiliar de bajo nivel para comprobar datos de texto en Unicode. Devuelve el número de bytes ocupados por el primer caracter de la cadena especificada en el formato de codificación actual.

- **ucwidth**

int ucwidth(int c);

Función auxiliar de bajo nivel para comprobar datos de texto en Unicode. Devuelve el número de bytes que ocuparía el caracter especificado en caso de codificarse en el formato actual.

- **uisok**

int uisok(int c);

Función auxiliar de bajo nivel para comprobar datos de texto en Unicode. Comprueba que el valor especificado puede ser codificado correctamente en en el formato actual.

- **uoffset**

int uoffset(const char *s, int index);

Devuelve el desplazamiento en bytes desde el comienzo de la cadena hasta el caracter correspondiente al índice especificado. Si el índice es negativo, cuenta hacía atrás desde el final de la cadena, así que un índice de -1 devolverá un desplazamiento al último caracter.

- **ugetat**

int ugetat(const char *s, int index);

Devuelve el valor del caracter de la cadena correspondiente al índice especificado. Un parámetro indice igual a 0 devolverá el primer caracter de la cadena. Si el índice es negativo, cuenta hacia atrás desde el final de la cadena. Así pues un índice de -1 devolverá el ultimo caracter de la cadena.

- **usetat**

int usetat(char *s, int index, int c);

Reemplaza el caracter del string con el índice especificado por el valor c, haciendo los ajustes necesarios debido a la anchura de la variable (ej: si c se codifica con una anchura diferente que el valor que había antes en esa posición). Devuelve el número de bytes que ha sido desplazada la parte derecha de la cadena. Si el índice es negativo cuenta hacia atrás desde el final de la cadena.

- **uinsert**

int uinsert(char *s, int index, int c);

Inserta el caracter c en la posición de la cadena especificada por index, desplazando el resto de los datos para hacer sitio. Devuelve el número de bytes que ha desplazado la parte derecha. Si el índice es negativo cuenta hacia atras desde el final de la cadena.

- **uremove**

int uremove(char *s, int index);

Borra el caracter que hay en la posición index de la cadena, desplazando el resto de los datos para llenar la posición vacía. Devuelve el número de bytes que se ha tenido que desplazar la parte derecha de la cadena, si index es negativo empieza a contar desde el final de la cadena

- **ustrsize**

int ustrsize(const char *s);

Devuelve el tamaño de la cadena especificada en bytes, sin incluir el caracter nulo finalizador de cadena.

- **ustrsizez**

int ustrsizez(const char *s);

Devuelve el tamaño de la cadena especificada en bytes, incluyendo el caracter nulo finalizador de cadena.

- **uwidth_max**

int uwidth_max(int type);

Rutinas ayudantes de bajo nivel para trabajar con texto Unicode. Devuelven el mayor número de bytes que un carácter puede ocupar en el formato de codificación especificado. Pase U_CURRENT para indicar el formato de codificación actual.

- **utolower**

int utolower(int c);

Esta función devuelve c, convirtiéndola a minúsculas si estaba en mayúsculas.

- **utoupper**

int utoupper(int c);

Esta función devuelve c, convirtiéndola a mayúsculas si
estaba en minúsculas.

- **uisspace**

int uisspace(int c);

Devuelve distinto de cero si c es carácter de espaciado, es decir, un retorno de carro, una nueva línea, página, un tabulador, un tabulador vertical o un espacio.

- **uisdigit**

int uisdigit(int c);

Devuelve distinto de cero si c es un dígito.

- **ustrdup**

char *ustrdup(const char *src)

Esta función copia la cadena src terminada en NULL en una nueva área de memoria reservada. La memoria devuelta por esta llamada debe ser liberada por el usuario. Devuelve NULL si no puede reservar espacio para la cadena duplicada.

- **_ustrdup**

```
char *_ustrdup(const char *src, void* (*malloc_func) (size_t))
```

Hace lo mismo que ustrdup(), pero permite especificar al usuario su propia rutina para reservar memoria.

- **ustrcpy**

```
char *ustrcpy(char *dest, const char *src);
```

Esta función copia src en dest (incluyendo el terminador de cadena NULL). El valor de retorno es el valor de dest.

- **ustrzcpy**

```
char *ustrzcpy(char *dest, int size, const char *src);
```

Esta función copia src en dest (incluyendo el terminador de cadena NULL), cuya longitud en bytes es especificada por size y que está garantizado que termine en carácter NULL. El valor de retoron es el valor de dest.

- **ustrcat**

```
char *ustrcat(char *dest, const char *src);
```

Esta función concatena src al final de dest. El valor de retorno es el valor de dest.

- **ustrzcat**

```
char *ustrzcat(char *dest, int size, const char *src);
```

Esta función concatena src al final de dest, cuya longitud en bytes es especificada por size y que está garantizado que termine en carácter NULL. El valor de retorno es el valor de dest.

- **ustrlen**

```
int ustrlen(const char *s);
```

Esta función devuelve el número de carácteres de s. Tenga en cuenta que esto no tiene que ser igual que el tamaño de la cadena en bytes.

- **ustrcmp**

int ustrcmp(const char *s1, const char *s2);

Esta función compara s1 con s2. Devuelve cero si las cadenas son iguales, un número positivo si s1 va detrás de s2 en la secuencia ASCII apropiada, o negativo en caso contrario.

- **ustrncpy**

char *ustrncpy(char *dest, const char *src, int n);

Esta función es como ustrcpy() excepto que no copiará más de n carácteres de src a dest. Si src es menor en longitud que n carácteres, se añadirán carácteres NULL en dest hasta rellenar los n carácteres especificados. Note que si src es mayor que n carácteres, dest no terminará en NULL. El valor de retorno es el valor de dest.

- **ustrzncpy**

char *ustrzncpy(char *dest, int size, const char *src, int n);

Esta función es como ustrzcpy() excepto que no copiará más de n carácteres de src a dest. Si src es menor en longitud que n carácteres, se añadirán carácteres NULL en dest hasta rellenar los n carácteres especificados. Note que está garantizado que dest acabe en carácter NULL. El valor de retorno es el valor de dest.

- **ustrncat**

char *ustrncat(char *dest, const char *src, int n);

Esta función es como `strcat' con la excepción de que no se añadirán más de n bytes de src al final de dest. Si el carácter terminador NULL es encontrado en src antes de haber escrito n carácteres, el carácter NULL será copiado, pero no se escribirán más carácteres. Si n carácteres son copiados antes de encontrar el carácter NULL, la función añadirá automáticamente el carácter NULL a dest, por lo que se escribirán n+1 carácteres. El valor de retorno es dest.

- **ustrzncat**

char *ustrzncat(char *dest, int size, const char *src, int n);

Esta función es como ustrszcat() con la excepción de que no se añadirán más de n carácteres de src al final de dest. Si el carácter terminador NULL es encontrado en src antes de haber escrito n carácteres, el carácter NULL será copiado, pero no se escribirán más carácteres. Note que está garantizado que dest acabe en carácter NULL. El valor de retorno es el valor de dest.

- **ustrncmp**

```
int ustrncmp(const char *s1, const char *s2, int n);
```

Esta función compara hasta n carácteres de s1 con s2. Devuelve cero si las cadenas son iguales, un número positivo si s1 va detrás de s2 en la secuencia ASCII apropiada, o negativo en caso contrario.

- **ustricmp**

```
int ustricmp(const char *s1, const char *s2);
```

Esta función compara s1 con s2, ignorando las mayúsculas.

- **ustrlwr**

```
char *ustrlwr(char *s);
```

Esta función sustituye todas las letras mayúsculas de s con minúsculas.

- **ustrupr**

```
char *ustrupr(char *s);
```

Esta función sustituye todas las letras minúsculas de s con mayúscylas.

- **ustrchr**

```
char *ustrchr(const char *s, int c);
```

Esta función devuelve un puntero a la primera ocurrencia de c en s, o NULL si s no contiene c. Tenga en cuenta que si c es NULL, esta función devolverá un puntero al final de la cadena.

- **ustrrchr**

```
char *ustrrchr(const char *s, int c);
```

Esta función devuelve un puntero a la última ocurrencia de c en s, o NULL si s no contiene c.

- **ustrstr**

```
char *ustrstr(const char *s1, const char *s2);
```

Esta función busca la primera ocurrencia de s2 en s1. Devuelve un puntero dentro de s1, o NULL si s2 no fue encontrada.

- **ustrpbrk**

```
char *ustrpbrk(const char *s, const char *set);
```

Esta función encuentra el primer carácter de s que esté contenido en set. Devuelve un puntero a la primera ocurrencia, o NULL si no se encontró nada.

- **ustrtok**

```
char *ustrtok(char *s, const char *set);
```

Esta función recupera palabras de s que están delimitadas por carácteres de set. Para iniciar la búsqueda, pase la cadena que quiere analizar como s. Para el resto de las palabras, pase NULL en su lugar. Devuelve un puntero a la palabra, o NULL si no se encontró nada. Aviso: dado que ustrtok altera la cadena que está analizando, debe copiar siempre su cadena a un buffer temporal antes de analizarla. Además, esta función no es reentrante (ej: no puede analizar dos cadenas simultáneamente).

- **ustrtok_r**

```
char *ustrtok_r(char *s, const char *set, char **last);
```

Versión reentrante de ustrtok. El último parámetro es usado para almacenar por dónde iba el procesado de la cadena y debe ser un puntero a una variable char * reservada por el usuario que no debe ser modificada mientras se procesa la misma cadena.

- **uatof**

```
double uatof(const char *s);
```

Convierte tanto como sea posible de la cadena a un número equivalente en coma flotante de doble precisión. Esta función es casi como `ustrtod(s, NULL)'. Devuelve un valor equivalente, o cero si la cadena no representa un número.

- **ustrtol**

```
long ustrtol(const char *s, char **endp, int base);
```

Esta función convierte la parte inicial de s a un entero con signo, el cual será devuelto como un valor de tipo `long int', haciendo que *endp apunte al primer carácter no convertido, si endp no es un puntero nulo. Si el parámetro base es cero, la base es determinada buscando cosas como `0x', `0X', o `0' como parte inicial de la cadena, y ajusta la base a 16, 16 u 8 respectivamente si se encuentra algo. La base por defecto es 10 en el caso de que no se detecte ninguno de esos prefijos.

- **ustrtod**

```
double ustrtod(const char *s, char **endp);
```

Convierte en número de coma flotante tantos carácteres de s que parezcan un número en coma flotante, y hace que *endp apunte al primer carácter no usado, si endp no es un puntero nulo.

- **ustrerror**

const char *ustrerror(int err);

Esta función devuelve una cadena que describe el código de error `err', que normalmente vendrá de la variable `errno'. Devuelve un puntero a una cadena estática que no debe ser modificada o liberada. Si hace llamadas posteriores a ustrerror, la cadena puede ser sobreescrita.

- **usprintf**

int usprintf(char *buf, const char *format, ...);

Esta función escribe datos formateados en el buffer de salida. El carácter NULL es escrito para marcar el final de la cadena. Devuelve el número de carácteres que fueron escritos, sin incluír el carácter terminador nulo.

- **uszprintf**

int uszprintf(char *buf, int size, const char *format, ...);

Esta función escribe datos formateados en el buffer de salida, cuya longitud en bytes es especificada por size, y que está garantizado que acabará en carácter NULL. Devuelve el número de carácteres que se hubiesen escrito sin contar la truncación eventual (como con usprintf), y sin incluír el carácter terminador NULL.

- **uvsprintf**

int uvsprintf(char *buf, const char *format, va_list args);

Esto es como usprintf, pero usted pasa una lista variable de argumentos en vez de los propios argumentos.

- **uvszprintf**

int uvszprintf(char *buf, int size, const char *format, va_list args);

Esto es como uszprintf(), pero usted pasa una lista variable de argumentos en vez de los propios argumentos.

RUTINAS DE CONFIGURACIÓN

Varias partes de Allegro, como las rutinas de sonido y la función load_joystick_data, requieren cierta información de configuración. Estos datos son almacenados en ficheros de texto como una colección de líneas "variable=valor", junto con comentarios que empiezan con el caracter '#' y acaban al fina de la línea. El fichero de configuración puede estar dividido en secciones que empiezan con una línea "[nombresección]". Cada sección tiene un nombre único, para prevenir conflictos con los nombres, pero la variable que no esté en una sección determinada es considerada perteneciente a todas simultáneamente.

Por defecto los datos de configuración se lee de un fichero llamado allegro.cfg, que puede estar en el mismo directoria que el programa ejecutable o en el directorio apuntado por la variable de entorno ALLEGRO. En Unix también busca en ~/allegro.cfg, ~/.allegrorc, /etc/allegro.cfg, y /etc/allegrorc, en ese orden; bajo BeOS sólo comprueba los dós últimos. Si esto no te gusta puedes especificar cualquier nombre de fichero, o usar un bloque binario de datos de configuracion proporcionado por tu programa (que, por ejemplo, puede ser cargado desde un fichero de datos).

Puede almacenar cualquier información que quiera en el fichero de configuración, junto con las variables estándar usadas por Allegro (mire abajo).

- **set_config_file**

```
void set_config_file(const char *filename);
```

Especifica el fichero de configuración que será usado por las siguientes rutinas de configuración. Si no llama esta función, Allegro usará el fichero allegro.cfg, mirando primero en el directorio de su programa y luego en el directorio apuntado por la variable de entorno ALLEGRO.

¡Todos los punteros devueltos por llamadas previas a get_config_string() y demás funciones similares son invalidados tras llamar a esta función.

- **set_config_data**

```
void set_config_data(const char *data, int length);
```

Especifica un bloque de datos que será usados por las siguientes rutinas de configuración, que previamente ha cargado del disco (ejemplo: como parte de un formato propio más complicado, o desde un fichero de datos). Esta rutina hace una copia de la información, por lo que puede liberar los datos después de llamarla.

- **override_config_file**

```
void override_config_file(const char *filename);
```

Especifica un fichero que contiene una configuración de sobreescritura. Sus datos serán usados además de los parámetros del fichero de configuración principal, y si hay una misma variable en ambos ficheros, la del fichero de sobreescritura tendrá preferencia. Esto puede ser usado por las aplicaciones del programador que deben tener ciertos valores de configuración fijos, pero otros del fichero de configuración principal pueden ser modificados por el usuario. Por ejemplo, podría especificar una frecuencia de muestreo de sonido y un fichero de instrumentos IBK específicos, pero el usuario podría usar un fichero sound.cfg o allegro.cfg para especificar los ajustes de su tarjeta de sonido (puertos, valores IRQ, etc).

- **override_config_data**

```
void override_config_data(const char *data, int length);
```

Versión de override_config_file() que usa un bloque de datos que ya ha sido cargado en memoria.

- **push_config_state**

```
void push_config_state();
```

Almacena el estado actual de configuración (nombre de fichero, valores de las variables, etc) en una pila interna, permitiéndole seleccionar otro fichero de configuración para después recuperar la configuración actual llamando pop_config_state(). Esta función está pensada para uso interno por otras funciones de la biblioteca, por ejemplo, cuando quiere especificar el fichero de configuración de la función save_joystick_data(), almacena la configuración actual antes de usar la del fichero especificado.

- **pop_config_state**

```
void pop_config_state();
```

Recupera el estado previo de la configuración almacenado por la función push_config_state() sobreescribiendo el código del fichero de configuración actual.

- **flush_config_file**

void flush_config_file();

Escribe el contenido del fichero de configuración actual al disco en caso de que haya cambiado desde que fue cargado o desde la última llamada a esta función.

- **reload_config_texts**

void reload_config_texts(const char *new_language);

Recarga las cadenas traducidas devueltas por get_config_text. Esto es útil para cambiar a otro lenguaje en su programa en tiempo de ejecución. Si quiere modificar manualmente la variable de configuración [system] language, o ha cambiado de ficheros de comfiguración, querrá pasar NULL para recargar el lenguaje que esté seleccionado actualmente. O puede pasar una cadena que contenga el código de dos letras del lenguaje al que desea cambiar, y la función modificará la variable language. Tras la llamada a esta función, los punteros previamente devueltos por get_config_text serán inválidos.

- **hook_config_section**

void hook_config_section(const char *section, int (*intgetter)(const char *name, int def), const char *(*stringgetter)(const char *name, char *def), void (*stringsetter)(const char *name, const char *value));

Toma control de la sección especificada del fichero de configuración, para que sus funciones de enganche sean usadas para manipularlo, en vez del acceso de disco normal. Si tanto las funciones getter como setter son NULL, un enganche presente será desenganchado. Las funciones enganchadas tienen la máxima prioridad. Si una sección está enganchada, la función de enganche siempre será llamada, por lo que puede enganchar una sección '#': incluso tengrá prioridad sobre la función verride_config_file().

- **config_is_hooked**

int config_is_hooked(const char *section);

Devuelve TRUE si la sección especificada está siendo enganchada.

- **get_config_string**

char *get_config_string(const char *section, const char *name, const char *def);

Recupera la cadena de texto de la variable name del fichero de configuración actual. Si la variable mencionada no es encontrada o su entrada está vacía, el valor def es devuelto. El nombre de sección puede ser NULL para aceptar

variables de cualquier parte del fichero, o puede ser usado para controlar en qué conjunto de parámetros (ejemplo: sonido o joystick) va a buscar la variable.

- **get_config_int**

int get_config_int(const char *section, const char *name, int def);

Lee un entero de la variable name del fichero de configuración actual. Lea el comentario de la función get_config_string().

- **get_config_hex**

int get_config_hex(const char *section, const char *name, int def);

Lee un entero de la variable name del fichero de configuración actual, en formato hexadecimal. Lea el comentario de la función get_config_string().

- **get_config_float**

float get_config_float(const char *section, const char *name, float def);

Lee un número en coma flotante de la variable name del fichero de configuración actual. Lea el comentario de la función get_config_string().

- **get_config_id**

int get_config_id(const char *section, const char *name, int def);

Lee una variable ID de 4 letras del fichero de configuración actual. Lea el comentario de la función get_config_string();

- **get_config_argv**

char **get_config_argv(const char *section, const char *name, int *argc);

Lee una lista de tokens (palabras separadas por espacios) del fichero de configuración actual, devolviendo una lista de argumentos al estilo de argv, y ajustando argc al número de tokens (a diferencia de argc/argv, esta lista tiene como base el cero). Devuelve NULL y ajusta argc a cero si la variable no esta presente. La lista de tokens es almacenada en un buffer temporal que será sobreescrito por la siguiente llamada a get_config_argv(), por lo que no espere que los datos persistan allí.

- **get_config_text**

char *get_config_text(const char *msg);

Esta función es usada principalmente por código interno de la biblioteca, pero también puede serle útil a los programadores de aplicaciones. Usa el fichero

language.dat o los ficheros XXtext.cfg (donde XX es el código del lenguaje) para mirar una versión traducida del parámetro en el lenguage actualmente seleccionado, devolviendo una traducción si ésta existe o una copia del parámetro si no hay otra cosa disponible. Esto es básicamente lo mismo que llamar get_config_string() con [language] como sección, msg como nombre de variable, y msg como como valor por defecto, pero tiene código especial añadido para manejar conversiones entre formatos Unicode. El parámetro msg siempre se pasa en ASCII, pero la cadena devuelta será convertida al formato de texto actual, reservando memoria siempre que sea necesario, por lo que puede asumir que el puntero devuelto persistirá, sin que tenga que reservar memoria manualmente para cada cadena.

- **set_config_string**

void set_config_string(const char *section, const char *name, const char *val);

Escribe una cadena en la variable name del fichero de configuración actual, sobreescribiendo cualquier valor previo, o borra la variable si val es NULL. El nombre de sección puede ser NULL para escribir la variable en la raíz del fichero, o puede ser usada para especificar la sección en la que desea insertar la variable. El fichero alterado será puesto en memoria cache, y no será escrito en disco hasta que llame allegro_exit(). Tenga en cuenta que sólo puede escribir en los ficheros de este modo, por lo que la función no tendrá efecto si el fichero de configuración actual fue especificado con set_config_data() en vez de con set_config_file().

Como caso especial, las variables o nombres de sección que comienzan con el carácter '#' son tratadas especialmente y no serán leídas o escritas en disco. Los paquetes adicionales pueden usar esto para almacenar información de version y otra información de estado en el módulo de configuración, desde el cual puede ser leída con la función get_config_string().

- **set_config_int**

void set_config_int(const char *section, const char *name, int val);

Escribe un entero en una variable en el fichero de configuración actual. Lea el comentario de set_config_string().

- **set_config_hex**

void set_config_hex(const char *section, const char *name, int val);

Escribe un entero en una variable en el fichero de configuración actual, en formato hexadecimal. Lea el comentario de set_config_string().

- **set_config_float**

void set_config_float(const char *section, const char *name, float val);

Escribe un número en coma flotante en una variable en el fichero de configuración actual. Lea el comentario de set_config_string().

- **set_config_id**

void set_config_id(const char *section, const char *name, int val);

Escribe una variable ID de 4 letras en el fichero de configuración actual. Lea el comentario de la función set_config_string().

- **standard config variables**

Allegro usa las siguientes variables estándar del fichero de configuración:

[system]
Sección que contiene variables de propósito general, que son:

· system = x
Especifica que driver de sistema usar. Actualmente ésto sólo es útil en Linux, para escoger entre los modos XWindows ("XWIN") o consola ("LNXC").

· keyboard = x
Especifica el mapa de teclado a usar. El parámetro es el nombre de un mapa de teclado producido por la utilidad keyconf, y puede ser un nombre de fichero normal o un nombre base como "us" o "uk". En este último caso, Allegro buscará primero un fichero de configuración independiente con ese nombre (ej: "uk.cfg") y después un objeto con ese nombre en el fichero keyboard.dat (ej: "UK_CFG"). El fichero de configuración o el fichero keyboard.dat pueden ser almacenados en el mismo directorio que su programa, o en una directorio apuntado por la variable de entorno ALLEGRO. Mire en el fichero keyboard.dat para ver los mapas de teclado disponibles.

· language = x
Especifica que fichero de lenguaje se usará para los mensajes de error y otros trozos de texto de sistema. El parámetro es el nombre de un fichero de traducción, y puede ser o un path completo o un nombre "base" como "en" o "sp". Si es esto último Allegro buscará primero un fichero de cofiguración a parte, con un nombre con la forma "entext.cfg", y después por un objeto con ese nombre en el fichero language.dat (ej: " ENTEXT_CFG"). El fichero de configuración o el fichero language.dat pueden ser guardados en el mismo directorio que el programa o en el directorio apuntado por la variable de

entorno ALLEGRO. Mire el fichero language.dat para saber que traducciones están disponibles actualmente.

· menu_opening_delay = x

Ajusta el tiempo que tiene que pasar antes de que un menú se abra automáticamente. El tiempo se da en milisegundos (por defecto 300). Especificando -1 se desactivará la característica de auto-apertura.

· dga_mouse = x

Sólo X: desactive para evitar un fallo en algunos servidores X en modo DGA, concerniente al ratón. Por defecto está activado; active el parche ajustando la variable a "0".

· dga_centre = x

Sólo X: indica al controlador DGA de Allegro que centre la pantalla si la resolución actual es mayor que la del programa Allegro. Por defecto activada; desactívela ajustando la variable a "0".

· dga_clear = x

Sólo X: indica al controlador DGA de Allegro limpiar la memoria de vídeo visible durante la inicialización. Por defecto activada; desactívela ajustando la variable a "0".

[graphics]

Sección que contiene la configuración gráfica, usando las siguientes variables:

· gfx_card = x

Especifica el controlador gráfico a usar cuando el programa haga una petición GFX_AUTODETECT. Se pueden sugerir múltiples controladores posibles con líneas extra en la forma "gfx_card2 = x", "gfx_card3 = x", etc, o puede especificar diferentes controladores para cada modo y profundidad de color con variables tipo "gfx_card_24bpp = x", "gfx_card_640x480x16 = x", etc.

· gfx_cardw = x

Especifica qué driver gráfico usar cuando el programa use GFX_AUTODETECT_WINDOWED. Esta variable funciona en lo demás exáctamente igual que gfx_card. Si no existe, Allegro usará la variable gfx_card.

· vbeaf_driver = x

Sólo DOS y Linux: especifica dónde buscar el controlador VBE/AF (vbeaf.drv). Si esta variable no está activada, Allegro buscará en el directorio del programa, y luego pasaría a buscará en los sitios estándar (c:\ para DOS,

/usr/local/lib, /usr/lib, /lib, y / para Linux, o el directorio especificado por la variable de entorno VBEAF_PATH).

· framebuffer = x

Sólo para Linux: especifica el fichero de dispositivo que hay que usar con el controlador fbcon. Si esta variable no está activada, Allegro mirará en la variable de entorno FRAMEBUFFER. Por defecto se usará /dev/fb0.

[mouse]

Sección que contiene la configuración del ratón, usando las siguientes variables:

· mouse = x

Tipos de driver para ratón. Los drivers disponibles para DOS son:

MICK -driver en modo mickey (normalmente el mejor)
I33 -Driver para atender a la interrupción 0x33
POLL -muestreo temporizado (para uso bajo NT)

Los drivers para la consola de Linux son:

MS -Ratón série de MicrosoftI
MS -Ratón série de Microsoft con extensión Intellimouse
LPS2 -Ratón PS2
LIPS -Ratón PS2 con extensión Intellimouse
GPMD -repetidor de datos GPM (Protocolo Mouse Systems)

· num_buttons = x

Ajusta del número de botones de ratón vistos por Allegro. Normalmente no necesitará ajustar esta variable porque Allegro la autodetectará. Sólo podrá usarla para restringir el número de botones del ratón.

· emulate_three = x

Especifica si hay que emular un tercer botón de ratón detectando pulsaciones simultáneas de los botones izquiedo y derecho (si o no). Por defecto activado (yes) si se dispone de un ratón de dos botones. En caso contrario, desactivado (no).

· mouse_device = x

Sólo para Linux: especifica el nombre del archivo de dispositivo del ratón (ej: /dev/mouse).

· mouse_accel_factor = x

Sólo Windows: especifica el factor de aceleración del ratón. Por defecto es 1. Modifíquelo a 0 para desactivar la aceleración. Un 2 acelerará el ratón el doble que un 1.

[sound]
Sección que contiene la configuración de sonido, usando las siguientes variables:

· digi_card = x
Elige el controlador para reproducir samples.

· midi_card = x
Elige el controlador para reproducir música MIDI.

· digi_input_card = x
Determina el controlador de entrada de sonido digital.

· midi_input_card = x
Determina el controlador de entrada para datos MIDI.

· digi_voices = x
Especifica el número mínimo de voces que reservará el controlador de sonido digital. El número de voces posibles depende del controlador.

· midi_voices = x
Especifica el número mínimo de voces que reservará el controlador de música MIDI. El número de voces posibles depende del controlador.

· digi_volume = x
Ajusta el volumen de reproducción de sonidos, de 0 a 255.

· midi_volume = x
Ajusta el volumen de reproducción de música midi, de 0 a 255.

· quality = x
Controla la balanza calidad vs. rapidez del sonido del código de mezcla de samples. Esto puede ser cualquiera de los siguientes valores:

0 -mezcla rápida de datos 8 bit en buffers de 16 bits
1 -mezcla verdadera de 16 bits (requiere una tarjeta de 16 bits estéreo)
2 -mezcla 16 bits interpolada

· flip_pan = x

Ajustando esto entre 0 y 1 invierte la panoramización izquierda/derecha de los samples, que puede ser necesaria porque algunas tarjetas SB (incluyendo la mía :-) crean la imagen estéro al revés.

· sound_freq = x

DOS, Unix y BeOS: ajusta la frecuencia de los samples. Con el controlador de la SB, los valores posibles son 11906 (cualquier SB), 16129 (cualquier SB), 22727 (SB 2.0 y superiores), y 45454 (sólo en SB 2.0 o SB16, no la SB Pro estéreo). Con el controlador de la ESS Audiodrive, los valores posibles son 11363, 17046, 22729, o 44194. Con la Ensoniq Soundscape, los valores posibles son 11025, 16000, 22050, o 48000. Con Windows Sound System, los valores posibles son 11025, 22050, 44100, o 48000. No se preocupe si establece un valor inexistente por error: Allegro lo redondeará automáticamente a la frecuencia más cercana soportada por su tarjeta de sonido.

· sound_bits = x

Unix y BeOS: establece el número de bits deseados (8 o 16).

· sound_stereo = x

Unix y BeOS: selecciona salida mono o estéreo (0 o 1 respectivamente).

· sound_port = x

Sólo DOS: establece el puerto de la tarjeta de sonido (normalmente 220).

· sound_dma = x

Sólo DOS: establece el canal DMA de la tarjeta de sonido (normalmente 1).

· sound_irq = x

Sólo DOS: establece el canal IRQ de la tarjeta de sonido (normalmente 7).

· fm_port = x

Determina el puerto del chip OPL (esto es normalmente 388).

· mpu_port = x

Determina el puerto de la interfaz MIDI MPU-401 (esto es normalmente 330).

· mpu_irq = x

Determina el IRQ del MPU-401 (esto es normalmente igual a sb_irq).

· ibk_file = x

Especifica el nombre de un fichero .IBK que será usado para sustituir el conjunto estándar de instrumentos Adlib.

· ibk_drum_file = x
Especifica el nombre de un fichero .IBK que será usado para sustituir el conjunto estándar de percusión Adlib.

· oss_driver = x
Sólo Unix: establece el nombre del dispositivo OSS. Normalmente /dev/dsp o /dev/audio, pero podría ser un dispositivo particular (ej: /dev/dsp2).

· oss_numfrags = x
oss_fragsize = x
Sólo Unix: establece el número de fragmentos (buffers) del controlador OSS y el tamaño de cada buffer en samples. Los buffers son rellenados con datos durante la interrupción donde el intervalo entre las siguientes interrupciones no es menor que 10 ms. Si el hardware puede reproducir toda la información de los buffers en menos de 10 ms, entonces se oirán clicks, cuando el hardware haya reproducido todos los datos y la biblioteca todavía no ha preparado los nuevos datos. Por otra parte, si el hardware tarda mucho en reproducir los datos de los buffers, entonces habrá restrasos entre las acciones que provocan los sonidos y los propios sonidos.

· oss_midi_driver = x
Sólo Unix: establece el nombre del dispositivo MIDI OSS. Normalmente /dev/sequencer.

· oss_mixer_driver = x
Sólo Unix: establece el nombre del mezclador OSS. Normalmente /dev/mixer.

· esd_server = x
Sólo Unix: indica dónde encontrar el servidor ESD (Enlightened Sound Daemon).

· alsa_card = x
alsa_pcmdevice = x
Sólo Unix: parámetros del sistema de sonido ALSA.

· alsa_numfrags = x
Sólo Unix. número de fragmentos (buffers) del sistema de sonido ALSA.

· alsa_fragsize = x
Sólo Unix: tamaño de cada fragmento ALSA, en samples.

· alsa_rawmidi_card = x
Sólo Unix: para el controlador MIDI de ALSA.

· alsa_rawmidi_device = x
Sólo Unix: para el controlador MIDI de ALSA.

· alsa_input_card = x
Sólo Unix: para el controlador MIDI de ALSA.

· alsa_rawmidi_input_device = x
Sólo Unix: para el controlador MIDI de ALSA.

· be_midi_quality = x
Sólo BeOS: calidad del sintetizador de instrumentos MIDI. 0 usa baja calidad de samples de 8-bit a 11kHz, 1 usa samples de 16-bit a 22kHz.

· be_midi_freq = x
Sólo BeOS: frecuencia de mezclado de samples MIDI en Hz. Puede ser 11025, 22050 o 44100.

· be_midi_interpolation = x
Sólo BeOS: especifica el método de interpolación de samples MIDI. 0 para no usar interpolación, es rápido pero de mala calidad; 1 hace una interpolación rápida con buen rendimiento, pero es algo más lento que lo anterior; 2 usa una interpolación lineal entre samples, que es el método más lento pero con el cual se obtiene la mejor calidad.

· be_midi_reverb = x
Sólo BeOS: intensidad de reverberación, de 0 a 5. 0 la desactiva, un 5 es el valor más fuerte.

· patches = x
Especifica dónde encontrar el conjunto de samples para el controlador DIGMID. Esto puede ser un directorio al estilo Gravis conteniendo una colección de ficheros .pat y un índice default.cfg, o un fichero de datos producido por la utilidad pat2dat. Si esta variable no es especificada, Allegro buscará un fichero default.cfg o patches.dat en el directorio del programa, en el directorio apuntado por la variable de entorno ALLEGRO, y en un directorio estándar GUS apuntado por la variable de entorno ULTRASND.

[midimap]
Si está usando los controladores de salida MIDI SB o MPU-401 con un sintetizador externo que no es compatible General MIDI, puede usar la sección midmap del fichero de configuración para especificar una tabla de mapa para convertir los números de los patches GM en sonidos apropiados de su sintetizador. Esto es una muestra real de indulgencia propia. Tengo un

Yamaha TG500, que tiene algunos sonidos geniales pero no tiene conjunto de patches GM, y simplemente tenía que hacerlo funcionar de alguna manera...
Esta sección consiste de una serie de líneas en la forma:

· p<n> = bank0 bank1 prog pitch

Con este comando, n es el número de programa GM a cambiar (1128), bank0 y bank1 son los dos bancos de mensajes de cambio a mandar a tu sintetizador (en controladores #0 y #32), prog es el mensaje de cambio de programa a tu sintetizador, y pitch es el número de semitonos a cambiar para todo lo que suene con ese sonido. Ajustando los números de cambio de banco a -1 hará que no sean mandados.

Por ejemplo, la línea:

p36 = 0 34 9 12

especifica que cuando el programa 36 GM (que es un bajo) sea seleccionado, Allegro mandará un mensaje de cambio de banco #0 con el parámetro 0, un mensaje de cambio de banco #32 con el parámetro 34, un cambio de programa con el parámetro 9, y entonces lo subirá todo una octava.

[joystick]

Sección que contiene la configuración del joystick, usando las siguientes variables:

· joytype = x

Especifica qué driver de joystick usar cuando el programa solicita usar JOY_TYPE_AUTODETECT.

· joystick_device = x

Sólo BeOS: especifica el nombre del dispositivo joystick que será usado. Por defecto se usa el primer dispositivo encontrado.

·

· throttle_axis = x

Sólo Linux: establece en qué eje está localizado el mando de gases. Esta variable será usada por cada joystick detectado. Si quiere especificar el eje de cada joystick individualmente, use variables con la forma throttle_axis_n, donde n sea el número del joystick.

RUTINAS DE RATÓN

- **install_mouse**

int install_mouse();

Instala el controlador del ratón de Allegro. Debe hacer esto antes de usar cualquier otra función del ratón. Devuelve -1 si hubo error (ej. si el controlador int33 no está cargado), o el número de botones del ratón.

- **remove_mouse**

void remove_mouse();

Quita el controlador de ratón. Normalmente no necesita llamar esta función, porque allegro_exit() lo hará por usted.

- **poll_mouse**

int poll_mouse();

Siempre que sea posible, Allegro leerá la entrada del ratón asíncronamente (ej: dentro de una interrupción), pero en algunas plataformas esto puede no ser posible, en cuyo caso debe llamar a esta rutina en intervalos regulares para actualizar las variables de estado del ratón. Para ayudarle a comprobar que su código de muestreo del ratón funciona incluso en una plataforma que no lo necesita, tras la primera llamada a esta rutina, Allegro entrará en modo muestreo, por lo que desde entonces en adelante deberá llamar manualmente a esta función para obtener cualquier dato del ratón, sin importar si el controlador actual necesita ser muestreado o nó. Devuelve cero con éxito, o un número negativo si hubo un fallo (ej: no hay driver de ratón instalado).

- **mouse_needs_poll**

int mouse_needs_poll();

Devuelve TRUE si el controlador de ratón actual está siendo operado en modo muestreo.

- **mouse_x**

extern volatile int mouse_x; extern volatile int mouse_y; extern volatile int mouse_b; extern volatile int mouse_pos;

Variables globales que contienen la posición actual del ratón y el estado de los botones. Las posiciones mouse_x y mouse_y son enteros que van de cero a la esquina inferior derecha de la pantalla. La variable mouse_b es un campo de

bits indicando el estado de cada botón: bit 0 es el botón izquierdo, bit 1 es del derecho, y bit 2 el botón central. Por ejemplo:

```
if (mouse_b & 1)
	printf("El botón izquierdo está pulsado\n");
if (!(mouse_b & 2))
	printf("El botón derecho no está pulsado\n");
```

La variable pouse_pos contiene la coordenada X actual en la palabra alta y la coordenada Y en la palabra baja. Esto es útil en bucles rápidos de lectura donde una interrupción del ratón podría ocurrir mientras lee las dos variables por separado, ya que puede copiar este valor a una variable local con una instrucción, y entonces separarlo con tranquilidad.

- **mouse_sprite**

```
extern BITMAP *mouse_sprite;
extern int mouse_x_focus;
exterm int mouse_y_focus;
```

Variables globales que contienen el sprite actual del ratón y el punto del foco. Estas variables sólo son de lectura, y sólo se pueden modificar usando las funciones set_mouse_sprite() y set_mouse_sprite_focus().

- **show_mouse**

```
void show_mouse(BITMAP *bmp);
```

Le dice a Allegro que muestre el puntero del ratón en la pantalla. Esto sólo funcionará si el módulo de temporización está instalado. El puntero del ratón será dibujado sobre el bitmap especificado, que será normalmente 'screen' (lee más abajo información sobre bitmaps). Para ocultar el puntero del ratón, llame show_mouse(NULL). Aviso: si dibuja algo en la pantalla mientras el puntero está visible, podría ocurrir una interrupción de movimiento en medio de su operación de dibujo. Si esto ocurre, el buffer del ratón y el código de cambio de banco SVGA se confundirán, y dejarán 'rastros de ratón' por toda la pantalla. Para evitar esto, debe asegurarse que oculta el puntero del ratón siempre que vaya a dibujar la pantalla.

- **scare_mouse**

```
void scare_mouse();
```

Función de ayuda para ocultar el puntero del ratón antes de una operación de dibujado. Esto se deshará temporalmente del puntero del ratón, pero sólo si es realmente necesario (ej: el ratón es visible, y está siendo visualizado en la

pantalla física, y no se trata de un cursor por hardware). El estado previo del ratón es almacenado para las llamadas siguientes a unscare_mouse().

- **scare_mouse_area**

```
void scare_mouse_area(int x, int y, int w, int h);
```

Como scare_mouse(), pero sólo ocultará el cursor si éste se encuentra dentro del rectángulo especificado. Si no lo está, el cursor simplemente será congelado hasta que llame a unscare_mouse(), para que no pueda interferir con su dibujado.

- **unscare_mouse**

```
void unscare_mouse();
```

Deshace el efecto de una llamada previa a scare_mouse(), recuperando el estado original del puntero.

- **freeze_mouse_flag**

```
extern int freeze_mouse_flag;
```

Si esta variable está activa, el puntero del ratón no será redibujado cuando mueva el ratón. Esto le puede evitar tener que ocultar el puntero cada vez que dibuje en la pantalla, siempre que no dibuje sobre la posición actual del puntero.

- **position_mouse**

```
void position_mouse(int x, int y);
```

Mueve el ratón a la posición de pantalla especificada. Puede llamar esta función incluso mientras el puntero esté visible.

- **position_mouse_z**

```
void position_mouse_z(int z);
```

Establece la variable que contiene la posición de la ruedecilla del ratón al valor indicado.

- **set_mouse_range**

```
void set_mouse_range(int x1, int y1, int x2, int y2);
```

Crea un área de pantalla sobre la que el ratón se podrá desplazar. Pase los parámetros de las esquinas del recuadro (coordenadas inclusivas). Si no llama

esta función, el área por defecto se activará a (0, 0, SCREEN_W-1, SCREEN_H-1).

- **set_mouse_speed**

void set_mouse_speed(int xspeed, int yspeed);

Ajusta la velocidad del ratón. Valores grandes de xspeed e yspeed significan un movimiento más lento: por defecto ambos son 2.

- **set_mouse_sprite**

void set_mouse_sprite(BITMAP *sprite);

¿No le gusta mi puntero de ratón? No problemo. Use esta función para usar uno propio alternativo. Si cambia el puntero y luego quiere volver a ver mi querida flecha otra vez, llame set_mouse_sprite(NULL).

Como bonificación, set_mouse_sprite(NULL) usa la paleta de colores actualmente seleccionada para elegir los colores de la flecha. Por lo que si el cursor se ve feo tras cambiar la paleta, llame a set_mouse_sprite(NULL).

- **set_mouse_sprite_focus**

void set_mouse_sprite_focus(int x, int y);

El foco del ratón es la parte del puntero que representa la posición actual del ratón, vamos, la posición (mouse_x, mouse_y). Por defecto el foco está arriba a la izquierda de la flecha, pero si va a usar un puntero de ratón diferente, quizás deba alterar esto.

- **get_mouse_mickeys**

void get_mouse_mickeys(int *mickeyx, int *mickeyy);

Mide cómo de lejos se ha movido el ratón desde la última llamada a esta función. El ratón seguirá generando unidades de movimiento incluso cuando llegue al borde de la pantalla, por lo que esta forma de control puede ser útil en juegos que requieran un rango de movimiento del ratón infinito.

- **mouse_callback**

extern void (*mouse_callback)(int flags);

Llamado por el controlador de interrupciones siempre cuando el ratón se mueva o el valor de los botones cambie. Esta función debe ser bloqueada en memoria (locked), y debe ejecutarse imuy rápido! Se le pasan los bits de evento que activaron la llamada, que son una máscara de bits que puede contener cualquiera de los siguientes valores:

MOUSE_FLAG_MOVE, MOUSE_FLAG_LEFT_DOWN
MOUSE_FLAG_LEFT_UP
MOUSE_FLAG_RIGHT_DOWN
MOUSE_FLAG_RIGHT_UP
MOUSE_FLAG_MIDDLE_DOWN
MOUSE_FLAG_MIDDLE_UP
MOUSE_FLAG_MOVE_Z

RUTINAS DE TEMPORIZACIÓN

Allegro puede establecer varias funciones virtuales de temporización, todas funcionando a diferentes velocidades, y reprogramará el reloj constantemente para asegurarse de que todas se llaman en el momento adecuado. Dado que estas rutinas alteran el chip de temporización de bajo nivel, estas rutinas no deben usarse con otras rutinas de temporización del DOS, como la rutina uclock() del djgpp.

- **install_timer**

```
int install_timer();
```

Instala el controlador de temporización de Allegro. Debe hacer esto antes de instalar cualquier rutina de temporización propia, e incluso antes de visualizar el puntero del ratón, reproducir una animación FLI, reproducir música MIDI y usar cualquiera de las rutinas GUI. Devuelve cero con éxito, o un número negativo si hubo problemas (pero puede decidir si quiere verificar el valor de retorno de esta función, dado que es muy poco probable que pueda fallar).

- **remove_timer**

```
void remove_timer();
```

Quita el controlador de temporización de Allegro y pasa el control del reloj a la BIOS. Normalmente no hace falta llamar esta función, porque allegro_exit() lo hará por usted.

- **install_int**

```
int install_int(void (*proc)(), int speed);
```

Instala un temporizador con el tiempo dado en número de milisegundos entre cada tick. Esto es lo mismo que hacer install_int_ex(proc, MSEC_TO_TIMER(speed)). Si llama esta rutina sin haber instalado primero el módulo de temporización, install_timer() será llamado automáticamente. Si no hay más espacio para añadir otro temporizador de usuario, install_int() devolverá un número negativo, en otro caso devolverá cero.

- **install_int_ex**

```
int install_int_ex(void (*proc)(), int speed);
```

Añade una función a la lista de temporizadores del usuario, o si ya está instalada, ajusta su velocidad retroactivamente (es decir, hace como si el

cambio de velocidad hubiese ocurrido precisamente en el último tick). El valor se da en ticks de reloj, que son 1193181 por segundo. Puede convertir la velocidad a partir de otros formatos de tiempo con las siguientes macros:

SECS_TO_TIMER(secs) -pase el número de segundos entre cada tick
MSEC_TO_TIMER(msec) -pase el número de milisegundos entre cada tick
BPS_TO_TIMER(bps) -pase el número de ticks por segundo
BPM_TO_TIMER(bpm) -pase el número de ticks por minuto

Si no queda espacio para un temporizador nuevo, install_int_ex() devolverá un número negativo, o cero de otro modo. Sólo puede haber 16 temporizadores a la vez, y algunas partes de Allegro (código GUI, rutinas para visualizar el puntero del ratón, rest(), el reproductor de ficheros FLI o MIDI) necesitan instalar sus propios temporizadores, por lo que debería evitar usar muchos a la vez.

Su función será llamada por el controlador de interrupciones de Allegro y no directamente por el procesador, por lo que puede ser una función normal en C, y no necesita ninguna función de envoltura. Sin embargo tenga en cuenta que será llamada en contexto de interrupción, lo que impone muchas restricciones sobre lo que puede hacer en ella. No debería usar grandes cantidades de pila, no puede hacer llamadas al sistema operativo o usar funciones de la biblioteca de C, o contener código con operaciones en coma flotante, y debe ejecutarse rápidamente. No intente hacer cosas complicadas con su temporizador: como regla general debería ajustar un par de valores y actuar en consecuencia de éstos dentro de su bucle de control principal.

En un entorno DOS en modo protegido como djgpp, la memoria es virtualizada y puede ser paginada a disco. Debido a la noreentrancia del DOS, si una paginación al disco ocurre dentro de su función de temporización, el sistema morirá de forma dolorosa, por lo que debe asegurarse de bloquear (lock) toda la memoria (de código y datos) que sea modificada dentro de su rutina de temporización. Allegro bloqueará todo lo que use, pero usted es responsable de bloquear su rutina de temporización. Las macros LOCK_VARIABLE(variable), END_OF_FUNCTION(nombre_de_funcion), y LOCK_FUNCTION(nombre_de_funcion) pueden ser usadas para simplificar esta tarea. Por ejemplo, si quiere que su temporizador incremente una variable de contador, debería escribir:

```
volatile int contador;
void mi_temporizador()
{
      contador++;
}END_OF_FUNCTION(mi_temporizador );
```

y en su código de inicio debería bloquear la memoria de esta manera:

```
LOCK_VARIABLE(contador);
LOCK_FUNCTION(mi_temporizador);
```

Obviamente esto puede ser extraño si usa estructuras de datos complicadas y llama otras funciones desde su temporizador, por lo que debería crear sus temporizadores tan simples como pueda.

- **remove_int**

```
void remove_int(void (*proc)());
```

Quita una función de la lista de temporizadores de usuario. Al finalizar su programa, allegro_exit() hará esto automáticamente.

- **install_param_int**

```
int install_param_int(void (*proc)(void *), void *param, int speed);
```

Como install_int(), pero a la rutina callback se le pasará una copia del puntero void especificado. Para desactivar este temporizador, use remove_param_int() en vez de remove_int().

- **install_param_int_ex**

```
int install_param_int_ex(void (*proc)(void *), void *param, int speed);
```

Como install_int_ex(), pero a la rutina callback se le pasará una copia del puntero void especificado. Para desactivar este temporizador, use remove_param_int() en vez de remove_int().

- **remove_param_int**

```
void remove_param_int(void (*proc)(void *), void *param);
```

Como remove_int(), pero se usa con las rutinas de temporización que tienen parámetros. Si hay más de una copia de la misma rutina activa a la vez, elegirá la rutina a desactivar comprobando el valor del parámetro (por lo que no puede tener más de una copia de un mismo temporizador usando un parámetro idéntico).

- **timer_can_simulate_retrace**

```
int timer_can_simulate_retrace()
```

Comprueba si es posible sincronizar el módulo de temporización con el retrazo del monitor, dependiendo del entorno y plataforma actual (por el momento

esto sólo es posible ejecutándo un el programa en modo DOS puro y en una resolución VGA o modo-X). Devuelve distinto de cero si la simulación es posible.

- **timer_simulate_retrace**

void timer_simulate_retrace(int enable);

El controlador DOS de temporización puede ser usado para simular interrupciones de retrazo vertical. Una interrupción de retrazo puede ser extremadamente útil para implementar una animacion suave, pero desafortunadamente el hardware de la VGA no puede hacerlo. La Ega lo podía hacer, y algunas SVGA pueden pero no completamente, y de forma no suficientemente estandarizada para que sea útil. Allegro soluciona esto programando el reloj para que genere una unterrupción cuando crea que va a ocurrir, y leyendo la VGA dentro del controlador de interrupción para asegurarse de que está sincronizado con el refresco del monitor. Esto funciona bastante bien en algunas situaciones, pero hay muchos problemas:

- Nunca use el simulador de retrazo en modos SVGA. Funcionará con algunas tarjetas, pero no en otras, y tiene conflictos con la mayoría de las implementaciones VESA. La simulación de retrazo sólo es fiable en el modo 13 de la VGA y en el modoX.

- La simulación de retrazo no funciona bajo win95, porque win95 devuelve basura cuando intento leer el tiempo transcurrido del PIT. Si alguien sabe cómo solucionar esto, ique por favor me mande un email!

- La simulación de retrazo conlleva mucha espera del controlador de temporización con las interrupciones desactivadas. Esto reducirá la velocidad del sistema de forma significante, y podría causar estática el reproducir sonidos con tarjetas SB 1.0 (ya que no soportan la autoinicialización DMA: las SB 2.0 y superiores funcionarán bien).

Considerando todos estos problemas, se aconsejaría no depender del simulador de retrazo vertical. Si está trabajando en modo-X, y no le importa que su programa funcione bajo win95, está bien, pero sería buena idea dejar al usuario la posibilidad de desactivarlo.

La simulación de retrazo debe ser activada antes de usar las funciones de triple buffer en resoluciones del modo-X. Esto puede ser útil también como una simple detección de retrazo, ya que leer vsync() puede hacer que ignore algún retrazo de vez en cuando si justo una interrupción de sonido o temporización ocurre a la vez. Cuando la simulación de retrazo está activada, vsync() comprobará la variable retrace_count en vez de leer los registros de la

VGA, para que no pierda ningún retrazo incluso si está siendo enmascarado por otras interrupciones.

- **timer_is_using_retrace**

```
int timer_is_using_retrace();
```

Comprueba si el modulo de temporización está, en ese momento, sincronizado con el monitor o no. Devuelve distinto de cero si lo está.

- **retrace_count**

```
extern volatile int retrace_count;
```

Si el simulador de retrazo está instalado, esto es incrementado con cada retrazo vertical, de otro modo es incrementado 70 veces por segundo (ignorando los retrazos). Esto le permite controlar la velocidad de su programa sin tener que instalar funciones de temporización propias.

La velocidad del retrazo depende del modo gráfico. En el modo 13h y resoluciones en modo-X de 200/400 líneas hay 70 retrazos por segundo, y en modos-X de 240/480 líneas hay 60. Puede ser tan bajo como 50 (en modo 376x282) o tan alto como 92 (en modo 400x300).

- **retrace_proc**

```
extern void (*retrace_proc)();
```

Si el simulador de retrazo está instalado, esta función será llamada durante cada retrazo, de otro modo es llamada 70 veces por segundo (ignorando los retrazos). Póngala a NULL para desactivar las llamadas. Esta función obedece las mismas reglas que los temporizadores normales (es decir: debe estar bloqueada (locked), y no puede llamar al DOS o funciones de libc) pero incluso más: debe ejecutarse _muy_ rápido, o fastidiará la sincronización del reloj. El único uso que veo para esta función es para hacer manipulaciones de paleta, ya que el triple buffering puede hacerse con la función request_scroll(), y la variable retrace_count puede ser usada para temporizar su código. Si quiere alterar la paleta dentro de retrace_proc, debe usar la función inline _set_color() en vez de la corriente set_color() o set_palette(), y no debería intentar alterar más de dos o tres colores de la paleta en un mismo retrazo.

- **rest**

```
void rest(long time);
```

Una vez que Allegro reprograma el reloj, la función estándar delay() no funcionará, por lo que tendrá que usar ésta. El tiempo time se pasa en milisegundos.

- **rest_callback**

void rest_callback(long time, void (*callback)())

Como rest(), pero llama continuamente la función específica mientras está esperando que pase el tiempo requerido.

RUTINAS DE TECLADO

El controlador de teclado de Allegro proporciona entrada con búffer y un conjunto de variables que guardan el estado actual de cada tecla. Fíjese que no es posible detectar correctamente todas y cada una de las combinaciones de teclas, debido al diseño del teclado del PC. Combinaciones de dos o tres teclas funcionarán bien, pero si presiona más, probablemente las extras serán ignoradas (exactamente qué combinaciones son posibles parecen variar de un teclado a otro).

- **install_keyboard**

```
int install_keyboard();
```

Instala el controlador de interrupciones de teclado de Allegro. Debe llamarla antes de usar cualquier otra función de teclado. Una vez instalado el controlador no podrá usar las llamadas a sistema o las funciones de librería de C para acceder al teclado. Devuelve cero con éxito, o un número negativo si hubo problemas (pero puede decidir si quiere verificar el valor de retorno dado que es muy poco probable que esta función falle).

- **remove_keyboard**

```
void remove_keyboard();
```

Desintala el controlador de teclado, devolviendo el control al sistema operativo. Normalmente no necesita llamar a esta función, porque allegro_exit() lo hará por usted.

- **install_keyboard_hooks**

```
void install_keyboard_hooks(int (*keypressed)(), int (*readkey)());
```

Sólo debería usar esta funcion si *no* va a usar el resto del controlador de teclado. Debe ser llamada en vez de install_keyboar(), y le deja proporcionar rutinas de atención para detectar y leer pulsaciones de teclado, que serán usadas por las funciones principales keypressed() y readkey(). Esto puede ser útil si quiere usar el código GUI de Allegro con un controlador de teclado propio, ya que permite al GUI acceder a la entrada de teclado desde su código, saltándose el sistema de entrada normal de Allegro.

- **poll_keyboard**

```
int poll_keyboard();
```

Siempre que sea posible Allegro intentará leer del teclado asíncronamente (por ejemplo deste un controlador de interrupción), pero en algunas plataformas puede que no sea posible, en cuyo caso deberá llamar a esta rutina a intervalos regulares para actualizar las variables de estado del teclado. Para ayudarle a comprobar su código que muestrea el teclado incluso si está programando en una plataforma que no lo necesita, después de la primera llamada Allegro cambiará a modo encuesta, así, en adelante, tendrá que llamar a esta rutina para obtener la entrada de teclado, sin importar si el driver actual necesita ser leído o no. La funciones keypressed(), readkey() y ureadkey() llaman a poll_keyboard() automáticamente, así que sólo tendra que usar esta función cuando acceda al array key[] y a la variable key_shifts. Devuelve cero con éxito, o un número negativo si hubo algún problema (ej: no hay driver de teclado instalado).

- **keyboard_needs_poll**

```
int keyboard_needs_poll();
```

Devuelve TRUE si el controlador actual de teclado está trabajando en modo muestreo.

- **key**

```
extern volatile char key[KEY_MAX];
```

Array de enteros que indica el estado de cada tecla, ordenado por scancode. Siempre que sea posible se actualizarán estos valores de forma asíncrona, pero si keyboard_needs_poll() devuelve TRUE, deberá llamar manualmente a poll_keyboard() para actualizarlos con el estado actual. Los scancodes están definidos en allegro.h como una serie de constantes KEY_*. Por ejemplo, podría escribir:

- **if (key[KEY_SPACE])**

```
printf("La barra espaciadora está siendo pulsada\n");
```

Tenga en cuenta que se supone que el array representa qué teclas están físicamente apretadas y cuales nó, por lo que semánticamente sólo es de lectura.

- **key_shifts**

```
extern volatile int key_shifts;
```

Máscara de bits que contienen el estado actual de shift/ctrl/alt, de las teclas especiales de Windows y los carácteres de escape de los acentos. Siempre que sea posible se actualizarán estos valores de forma asíncrona, pero si keyboard_needs_poll() devuelve TRUE, deberá llamar manualmente a poll_keyboard() para actualizarlos con el estado actual. Puede contener cualquiera de los bits:

KB_SHIFT_FLAG
KB_CTRL_FLAG
KB_ALT_FLAG
KB_LWIN_FLAG
KB_RWIN_FLAG
KB_MENU_FLAG
KB_SCROLOCK_FLAG
KB_NUMLOCK_FLAG
KB_CAPSLOCK_FLAG
KB_INALTSEQ_FLAG
KB_ACCENT1_FLAG
KB_ACCENT2_FLAG
KB_ACCENT3_FLAG
KB_ACCENT4_FLAG

- **keypressed**

```
int keypressed();
```

Devuelve TRUE si hay teclas esperando en el buffer de entrada. Esto es equivalente a la función kbhit() de la biblioteca libc.

- **readkey**

```
int readkey();
```

Devuelve el siguiente caracter del búffer de teclado en formato ASCII. Si el búffer está vacío espera hasta que se apriete una tecla. El byte de menos peso del valor de retorno contiene el código ASCII de la tecla, y el byte de mayor peso el scancode. El scancode sigue siendo el mismo a pesar del estado de las teclas shift, ctrl y alt, mientras que al código ASCII sí que le afecta la pulsación de shift y ctrl de la manera normal (shift cambia a mayúsculas, ctrl+letra da la posición de la tecla en el alfabeto, ej: ctrl+A = 1, ctrl+B = 2, etc). Apretando alt+key se devuelve sólo el scancode, con el código ASCII cero en el byte de menor peso. Por ejemplo:

```
if ((readkey() & 0xff) == 'd')          // por código ASCII
```

```
        printf("Has pulsado 'd'\n");
if ((readkey() >> 8) == KEY_SPACE)     // por código scancode
        printf("Has pulsado Espacio\n");
if ((readkey() & 0xff) == 3)           // ctrl+letter
        printf("Has pulsado Control+C\n");
if (readkey() == (KEY_X << 8))         // alt+letter
        printf("Has pulsado Alt+X\n");
```

Esta función no puede devolver carácteres mayores que 255. Si necesita leer entradas Unicode use ureadkey() en vez de readkey()

- **ureadkey**

```
int ureadkey(int *scancode);
```

Devuelve el siguiente caracter del búffer de teclado en formato Unicode. Si el búffer está vacío se espera hasta que se presione una tecla. El valor de retorno contiene el valor Unicode de la tecla, y si no es NULL, en el argumento se iniciará con el scancode. Al contrario que readkey() esta función es capaz de devolver carácteres mayores que 255.

- **scancode_to_ascii**

```
int scancode_to_ascii(int scancode);
```

Convierte el scancode dado a un caracter ASCII para esa tecla, devolviendo el resultado de apretar esa tecla sin shift ni capslock, o cero si la tecla no es un caracter generable o no se puede ser traducido.

- **simulate_keypress**

```
void simulate_keypress(int key);
```

Introduce una tecla en el buffer del teclado, como si el usuario la hubiese pulsado. El parámetro está el mismo formato que el devuelto por readkey().

- **simulate_ukeypress**

```
void simulate_ukeypress(int key, int scancode);
```

Introduce una tecla en el búffer de teclado, como si el usuario la hubiese pulsado. El parámetro está en el mismo formato devuelto por ureadkey()

- **keyboard_callback**

```
extern int (*keyboard_callback)(int key);
```

Si se activa, esta función es será llamada por el controlador del teclado en respuesta a cualquier tecla. Se le pasará una copia del valor que se va a

añadir al buffer de entrada, y la función puede devolver este valor sin modificar, devolver cero para que la tecla sea ignorada, o devolver un valor modificado que cambiará lo que readkey() va a devolver después. Esta rutina se ejecuta en contexto de interrupción, por lo que debe estar bloqueada (locked) en memoria.

- **keyboard_ucallback**

```
extern int (*keyboard_ucallback)(int key, int *scancode);
```

Versión Unicode de keyboard_callback(). Si se activa, esta función es llamada por el controlador de teclado en respuesta a cualquier pulsación de tecla. Se le pasa el valor del caracter y el scancode que serán añadidos al búffer de entrada, puede modificar el valor del scancode, y devuelve un código de tecla nuevo o modificado. Si cambia el scancode a cero y devuelve un cero la tecla será ignorada. Esta rutina se ejecuta en un contexto de interrupción, por lo que debe ser bloqueada (locked) en memoria.

- **keyboard_lowlevel_callback**

```
extern void (*keyboard_lowlevel_callback)(int scancode);
```

Si se, activa esta función es llamada por el controlador de teclado en respuesta a cada evento de teclado, tanto cuando se pulsa como cuando se suelta. Se le pasará un byte de scancode puro, con el bit de más peso desactivado si la tecla ha sido pulsada o activado si ha sido soltada. Esta rutina se ejecuta en un contexto de interrupción, así que debe estar bloqueada (locked) en memoria.

- **set_leds**

```
void set_leds(int leds);
```

Modifica el estado de los indicadores LED del teclado. El parámetro es una máscara de bits conteniendo cualquiera de los valores KB_SCROLOCK_FLAG, KB_NUMLOCK_FLAG, y KB_CAPSLOCK_FLAG, o -1 para recuperar el comportamiento normal.

- **set_keyboard_rate**

```
void set_keyboard_rate(int delay, int repeat);
```

Inicializa la frecuendia de repetición del teclado. Los tiempos se dan en milisegundos. Pasar cero desactivará la repetición de teclas.

- **clear_keybuf**

```
void clear_keybuf();
```

Limpia el búffer de teclado.

- **three_finger_flag**

```
extern int three_finger_flag;
```

El controlador de teclado de djgpp proporciona una secuencia de 'salida de emergencia' que puede usar para salir de su programa. Si está ejecutando bajo DOS será la combinación ctrl+alt+del. La mayoría de SSOO multitarea capturarán esta combinación antes de que llegue al controlador de Allegro, en cuyo caso puede usar la combinación alternativa ctrl+alt+fin. Si quiere desactivar este comportamiento en su programa ajuste esta variable a FALSE.

- **key_led_flag**

```
extern int key_led_flag;
```

Por defecto las teclas BloqMayús, BloqNum y BloqDesp activan los LEDS del teclado cuando son presionadas. Si las quiere utilizar en su juego (Ej. BloqMayús para disparar) este comportamiento no es deseable, por lo que puede poner a cero esta variable para evitar que los LEDS sean actualizados.

RUTINAS DE JOYSTICK

- **install_joystick**

int install_joystick(int type);

Inicializa el joystick y calibra el valor de la posición central. El parámetro typo debería ser, normalmente, JOY_TYPE_AUTODETECT, o mirar la documentación específica de la plataforma para tener una lista de los drivers disponibles. Debe llamar a esta rutina antes de usar cualquier otra función del joystick, y se debería asegurar que el joystick está en la posición central en ese momento. Devuelve cero si no ha habido problemas. Tan pronto como haya instalado el módulo de joystick, ya será capaz de leer el estado de los botones y la información digital(on/off) de la dirección, que puede ser suficiente para algunos juegos. Si quiere obtener una entrada totalmente analógica necesitará usar las funciones de calibrate_joystick() para medir el rango exacto de las entradas: lea más abajo.

- **remove_joystick**

void remove_joystick();

Quita el controlador de joystick. Normalmente no necesita llamar a esta rutina, porque allegro_exit() lo hará por usted.

- **poll_joystick**

int poll_joystick();

El joystick no funciona por interrupciones, así que necesitará llamar a esta función una y otra vez para actualizar los valores globales de posición. Devuelve cero con éxito o un número negativo si hubo problemas (normalmente porque no había driver de joystick instalado).

- **num_joysticks**

extern int num_joysticks;

Variables globales que indican el número de joysticks activos. Los controlador actuales soportan un máximo de cuatro dispositivos.

- **joy**

extern JOYSTICK_INFO joy[n];

Array global de información de estado del joystick, que es actualizado por la función poll_joystick(). Sólo el primer elemento num_joysticks tendrá información útil. La estructura JOYSTICK_INFO está definida así:

```
typedef struct JOYSTICK_INFO
{
   int flags;                          -estado de este joystick
   int num_sticks;                     -¿cuántos joysticks activos?
   int num_buttons;                    -¿cuántos botones?
   JOYSTICK_STICK_INFO stick[n];       -información de estado del stick
   JOYSTICK_BUTTON_INFO button[n];     -nformación de estado de los
                                        botones
} JOYSTICK_INFO;
```

El estado de los botones es almacenado en la estructura:

```
typedef struct JOYSTICK_BUTTON_INFO
{
   int b;          -estado del botón on/off
   char *name;     -descripción de este botón
} JOYSTICK_BUTTON_INFO;
```

Puede enseñar los nombres de los botones como parte de una pantalla de configuración en la que el usuario elige qué función desempeñará cada botón en su juego, pero en situaciones simples puede asumir con seguridad que los dos primeros elementos del array button serán siempre los controles principales de disparo.

Cada joystick proveerá una o más entradas stick, de varios tipos. Estas pueden ser controles digitales que tienen siempre un valor específico (ej. un gamepad, el sombrero del Flightstick Pro o Wingman Extreme, o un joystick normal que todavía no ha sido calibrado), o pueden ser entradas analógicas con un rango suave de movimiento. Las palancas pueden tener un número diferente de ejes, por ejemplo un controlador direccional normal tiene dos, pero el mando de gases del Flightstick Pro sólo tiene un eje, y es posible que el sistema pueda ser expandido en el futuro para soportar controladores 3d. La entrada de la palanca está descrita por la estructura:

```
typedef struct JOYSTICK_STICK_INFO
{
   int flags;                          -variable de estado
   int num_axis;                       -¿cuántos ejes tenemos?
   JOYSTICK_AXIS_INFO axis[n];         -información de estado del eje
   char *name;                         -descripción de este stick
} JOYSTICK_STICK_INFO;
```

Un solo joystick puede proveer diferentes entradas de joystick, pero puede asumir con seguridad que el primer elemento del array stick será el controlador principal de dirección.

La información sobre los ejes del mando está almacenada en la subestructura:

```
typedef struct JOYSTICK_AXIS_INFO
{
      int pos;              -posición analógica del eje
      int d1, d2;           -posición digital del eje
      char *name;           -descripción de este eje
} JOYSTICK_AXIS_INFO;
```

Esto le da tanto entrada digital en el campo pos (que va de 128 a 128 o de 0 a 255, dependiendo del tipo de control) como valores digitales en los campos d1 y d2. Por ejemplo, cuando describe la posición del eje X, el campo pos contendrá la posición horizontal de joystick, d1 será activado si mueve a la izquierda, y d2 será activado si mueve a la derecha. Allegro rellenará todos estos valores sin importar el tipo de joystick que esté conectado, emulando el campo pos para joysticks digitales poniéndolo a las posiciones mínima, central y máxima, y emulando los valores d1 y d2 con joysticks analógicos comparando la posición actual con el punto central.

La variable flags de la estructura joystick puede ser cualquier combinación de los siguientes bits:

JOYFLAG_DIGITAL	-Este control tiene entrada digital.
JOYFLAG_ANALOGUE	-Este control tiene entrada analógica.
JOYFLAG_CALIB_DIGITAL	-Este control será capaz de proveer entrada digital una vez sea calibrado, pero ahora no lo hace.
JOYFLAG_CALIB_ANALOGUE	-Este control será capaz de proveer entrada analógica una vez sea calibrado, pero ahora no lo hace.
JOYFLAG_CALIBRATE	-Indica que este control debe ser calibrado.

Muchos dispositivos requieren múltiples pasos de calibración, por lo que puede llamar la función calibrate_joystick() desde un bucle hasta que este bit desaparezca.

JOYFLAG_SIGNED	-Indica que la posición analógica está en formato con signo, que va de -128 a 128. Este es el caso de todos los controles direccionales 2d.

JOYFLAG_UNSIGNED -Indica que la posición analógica está en formato sin signo, que va de 0 a 255. Este es el caso de todos los mandos de gases 1d.

Nota para la gente que escribe diferente: en caso que no quiera escribir "analogue", hay varios #defines en allegro.h que le permitirán escribir "analog" sin problemas.

- **calibrate_joystick_name**

```
char *calibrate_joystick_name(int n);
```

Devuelve una descripción textual del siguiente tipo de calibración que será hecha en el joystick especificado, o NULL si no hace falta más calibración.

- **calibrate_joystick**

```
int calibrate_joystick(int n);
```

La mayoría de los joysticks deben ser calibrados antes de poder ser usados de forma analógica. Esta función realiza la siguiente operación en la serie de calibración para el stick especificado, asumiendo que el joystick ha sido posicionado de la forma descrita por la llamada previa a calibrate_joystick_name(), devolviendo cero con éxito. Por ejemplo, una rutina simple para calibrar los joysticks podría ser así:

```
int i;
for (i=0; i<;num_joysticks; i++)
{
	while (joy[i].flags & JOYFLAG_CALIBRATE)
	{
		char *msg = calibrate_joystick_name(i);
		printf("%s, y pulsa una tecla\n", msg);
		readkey();
		if (calibrate_joystick(i) != 0)
		{
			printf("ioops!\n"); exit(1);
		}
	}
}
```

- **save_joystick_data**

```
int save_joystick_data(const char *filename);
```

Después de todos los dolores de cabeza al calibrar el joystick, no querrá que el pobre usuario tenga que repetir el proceso cada vez que ejecuta su programa.

Llame esta función para salvar los datos de calibración del joystick en un fichero de configuración especificado, que puede ser leído por load_joystick_data(). Pase NULL como filename para escribir los datos en el fichero de configuración seleccionado actualmente. Devuelve cero si no hubo problemas.

- **load_joystick_data**

```
int load_joystick_data(const char *filename);
```

Recupera los datos de calibrado previamente almacenados por save_joystick_data() o la utilidad setup. Esto ajusta todos los aspectos del código de joystick: ni si quiera debe llamar initialise_joystick() si está usando esta función. Pasa NULL como filename para leer los datos del fichero de configuración seleccionado actualmente. Devuelve cero si no hubo problemas: si falla, el estado del joystick queda indefinido y debe reiniciarlo desde el comienzo.

- **initialise_joystick**

```
int initialise_joystick();
```

Deprecado. Use install_joystick() en su lugar.

- **set_color_depth**

void set_color_depth(int depth);

Especifica el formato gráfico que será usado en las siguientes llamadas a set_gfx_mode() y create_bitmap(). Las profundidades válidas son 8 (por defecto), 15, 16, 24 y 32 bits.

- **request_refresh_rate**

void request_refresh_rate(int rate);

Solicita que en la siguiente llamada a set_gfx_mode() se intente usar la velocidad de refresco especificada, si es posible. No todos los controladores son capaces de esto, e incluso cuando pueden, no todas las velocidades serán posibles en cualquier hardware, por lo que el resultado puede diferir de lo que haya pedido. Tras la llamada a set_gfx_mode(), puede usar get_refresh_rate() para saber qué velocidad de refresco fue seleccionada. Por el momento, sólo el driver DOS VESA 3.0, X DGA 2.0 y algunos drivers de DirectX soportan esta función. La velocidad es especificada en Hz, ej: 60, 70. Para volver a la velocidad por defecto, pase el valor cero.

- **get_refresh_rate**

int get_refresh_rate(void);

Devuelve la velocidad de refresco actual, si es conocida (no todos los controladores pueden devolver esta información). Devuelve cero si la velocidad de refresco actual es desconocida.

- **get_gfx_mode_list**

GFX_MODE_LIST *get_gfx_mode_list(int card);

Intenta crear una lista de todos los modos de vídeo soportados por un driver gráfico GFX determinado. Esta función devuelve un puntero a una lista de estructuras del tipo GFX_MODE_LIST que estan definidas como:

```
typedef struct GFX_MODE_LIST
{
	int num_modes;
	GFX_MODE *mode;
```

```
} GFX_MODE_LIST;
```

Si esta función devuelve NULL, significa que la llamada no tuvo éxito. El puntero mode apunta a la verdadera lista de modos de vídeo.

```
typedef struct GFX_MODE
{
	int width, height, bpp;
} GFX_MODE;
```

Esta lista termina con un elemento { 0, 0, 0 }

- **get_gfx_mode_list**

```
GFX_MODE_LIST *get_gfx_mode_list(int card);
```

Intenta crear una lista de todos los modos de vídeo soportados por un driver gráfico GFX determinado. Esta función devuelve un puntero a una lista de estructuras del tipo GFX_MODE_LIST que estan definidas como:

```
typedef struct GFX_MODE_LIST
{
	int num_modes;
	GFX_MODE *mode;
} FX_MODE_LIST;
```

Si esta función devuelve NULL, significa que la llamada no tuvo éxito. El puntero mode apunta a la verdadera lista de modos de vídeo.

```
typedef struct GFX_MODE
{
	int width, height, bpp;
} GFX_MODE;
```

Esta lista termina con un elemento { 0, 0, 0 }

- **destroy_gfx_mode_list**

```
void destroy_gfx_mode_list(GFX_MODE_LIST *mode_list);
```

Borra de la memoria la lista de modos creada por get_gfx_mode_list().

- **set_gfx_mode**

```
int set_gfx_mode(int card, int w, int h, int v_w, int v_h);
```

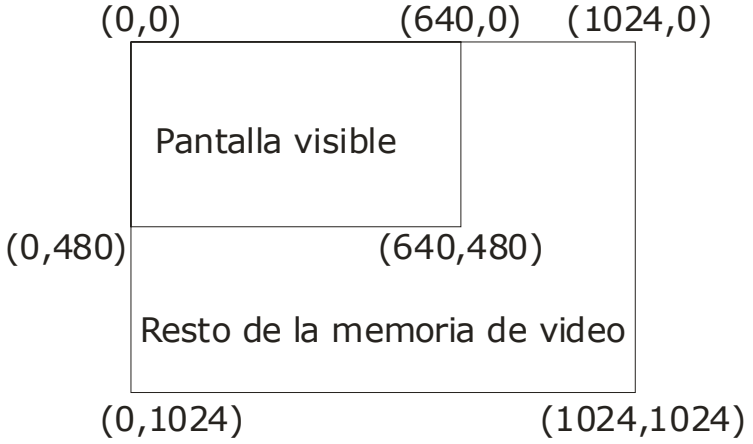
Cambia a modo gráfico. El parámetro card normalmente debería ser:

GFX_AUTODETECT
GFX_AUTODETECT_FULLSCREEN
GFX_AUTODETECT_WINDOWED

O puede mirar la documentación específica de su plataforma para tener una lista los drivers disponibles. Los parámetros w y h especifican que resolución de pantalla quiere.

Los parámetros v_w y v_h especifican el tamaño mínimo de la pantalla virtual, en caso de que necesite una pantalla virtual grande para hacer scroll por hardware o intercambio de páginas. Debería ponerlos a cero si no le importa la pantalla virtual. Las pantallas virtuales crean mucha confusión, pero en realidad son muy simples. Aviso: ahora viene una explicación condescendiente, por lo que quizás quiera saltarse el resto del párrafo :-) Imagínese la memoria de vídeo como una pieza rectangular de papel que es vista a través de un pequeño agujero (su monitor) hecho sobre cartón. Ya que el papel es más grande que el agujero, sólo puede ver una parte de él cada vez, pero moviendo el cartón puede alterar qué porción está visualizando. Puede dejar el agujero en una posición e ignorar el resto de la memoria de vídeo no visible, pero puede conseguir una gran variedad de útiles efectos desplazando la ventana de visualización, o dibujando en una parte de la memoria de vídeo no visible, y entonces intercambiar páginas para visualizar la nueva pantalla.

Por ejemplo, puede seleccionar el modo 640x480 en el cual el monitor actúa como una ventana sobre la pantalla virtual de 1024x1024, y entonces mover la pantalla visible dentro del área grande. Inicialmente, la parte visible está posicionada en la parte superior izquierda de la memoria de vídeo. Esta configuración sería así:



¿Qué es eso? ¿Está viendo esto con una fuente proporcional? Jejeje.

Cuando llama set_gfx_mode(), los parámetros v_w y v_h representan el tamaño mínimo de la pantalla virtual que es aceptable para su programa. El rango de tamaños posibles es normalmente muy restringido, y es posible que Allegro acabe creando una pantalla virtual más grande que la que solicitó. En una SVGA con una mega de vram cuente con conseguir una pantalla virtual de 1024x1024 (256 colores) o 1024x512 (15 o 16 bpp), y con 512k de ram puede conseguir 1024x512 (256 colores). Otros tamaños pueden ser y no ser posibles: no asuma que vayan a funcionar. En modo-X la anchura virtual puede ser cualquier múltiplo de 8 mayor o igual a la anchura de la pantalla, y la altura virtual se ajustará de la misma forma (la VGA tiene 256k de vram, por lo que la altura virtual será 256*1024/virtual_width).

Después de seleccionar el modo gráfico, los tamaños físicos y virtuales de la pantalla pueden obtenerse mediante las macros SCREEN_W, SCREEN_H, VIRTUAL_W, y VIRTUAL_H.

Si Allegro no es capaz de seleccionar un modo apropiado, set_gfx_mode() devuelve un modo negativo y almacena la descripción del problema en allegro_error. De otro modo devuelve cero.

Como caso especial, si usa el código de driver mágico GFX_SAFE, Allegro garantizará que el modo siempre será establecido correctamente. Intentará seleccionar la resolución que pidió, y si falla, usará aquella resolución que sea fiable en la plataforma actual (esto es modo VGA a 320x200 bajo DOS, una resolución de 640x480 bajo Windows, la resolución actual del framebuffer bajo Linux si está soportada, etc). Si no puede establecer ningún modo gráfico de ninguna manera, devolverá un valor negativo indicando que debería abortar inmediatamente su programa, posiblemente tras informar al usuario de este hecho mediante allegro_message. Este driver falso es útil para situaciones en las que quiere establecer un modo gráfico que funcione, y no puede perder el tiempo probando diferentes resoluciones y haciendo la comprobación de errores. Tenga en cuenta, que tras una llamada con éxito a set_gfx_mode con este driver, no puede hacer ninguna asunción sobre el ancho o alto de la pantalla o su profundidad de color: su código deberá tratar con este pequeño detalle.

- **set_display_switch_mode**

```
int set_display_switch_mode(int mode);
```

Establece de qué forma el programa debe manejar el hecho de ser enviado a un segundo plano si el usuario cambia a otro programa. No todos los modos posibles serán soportados por cada driver gráfico en cada plataforma: debe

llamar a esta rutina tras iniciar el modo gráfico y si pide un modo que no es posible, esta rutina devolverá -1. Los modos disponibles son:

· SWITCH_NONE
Desactiva el cambio de modo. Este modo está por defecto en sistemas monotarea como el DOS. Puede ser soportado en otras plataformas, pero debería usarlo con cuidado, porque sus usuarios no se impresionarán si quieren cambiar de programa, ¡pero usted no les deja!

· SWITCH_PAUSE
Pone en pausa el programa mientras esté en segundo plano. La ejecución se restablecerá tan pronto como el usuario vuelva al programa. Este modo está por defecto en la mayoría de los entornos multitarea a pantalla completa, por ejemplo la consola de Linux.

· SWITCH_AMNESIA
Como SWITCH_PAUSE, pero este modo no se preocupa en recordar el contenido de la memoria de vídeo, por lo tanto de la pantalla, y los bitmaps de vídeo que haya creado se eliminarán después de que el usuario cambie de programa y vuelva otra vez. Este no es un modo terriblemente útil pero es el modo por defecto para los modos a pantalla completa bajo Windows porque DirectDraw es demasiado inútil para implementar algo mejor.

· SWITCH_BACKGROUND
El programa seguirá ejecutándose en background, con la variable screen apuntando temporalmente a un buffer de memoria en los modos de vídeo a pantalla completa. Debe tener especial cuidado cuando use este modo, porque ocurrirán cosas malas si el bitmap de pantalla cambia cuando su programa no lo espera (lea más abajo).

· SWITCH_BACKAMNESIA
Como SWITCH_BACKGROUND, pero este modo no se preocupa por acordarse del contenido de la memoria de vídeo (vea SWITCH_AMNESIA). De nuevo, es el único modo soportado por los drivers a pantalla completa de Windows que permite que su programa siga ejecutándose en segundo plano.

Acuérdese de tener mucho cuidado cuando use rutinas gráficas durante un contexto de cambio: siempre deberá llamar acquire_screen() antes de comenzar a dibujar en la pantalla y no la libere hasta que no haya acabado completamente, porque el mecanismo de fijado automático puede no ser suficientemente bueno que funcione mientras el programa se ejecuta en segundo plano o acaba de de pasar al primer plano.

- **set_display_switch_callback**

int set_display_switch_callback(int dir, void (*cb)());

Instala una función de notificación para el cambio de modo que fue previamente seleccionado por set_display_switch_mode(). El parámetro direction puede ser SWITCH_IN o SWITCH_OUT, dependiendo de si quiere ser avisado cuando se deje su programa o cuando se vuelva a él. A veces puede instalar funciones para las dos direcciones, pero no todas las plataformas las soportan, así esta función puede devolver -1 si su petición es imposible. Puede instalar diferentes funciones de cambio de modo al mismo tiempo.

- **remove_display_switch_callback**

void remove_display_switch_callback(void (*cb)());

Elimina una función de notificación que fue previamente instalada mediante set_display_switch_callback(). Todas las funciones serán eliminadas automáticamente cuando llame a set_display_switch_mode().

- **get_display_switch_mode**

int get_display_switch_mode();

Devuelve el modo de cambio de pantalla, en el mismo formato que se pasa a set_display_switch_mode().

- **gfx_capabilities**

extern int gfx_capabilities;

Campo de bits describiendo las capacidades del controlador gráfico y el hardware de video actuales. Puede contener cualquiera de los siguientes bits:

· GFX_CAN_SCROLL:
Indica que la función scroll_screen() puede ser usada con este controlador.

· GFX_CAN_TRIPLE_BUFFER:
Indica que las funciones request_scroll() y poll_scroll() se pueden usar con este driver. Si este bit no está activado es posible que la función enable_triple_buffer() sea capaz de activarlo.

· GFX_HW_CURSOR:
Indica que un puntero de ratón por hardware está siendo usado. Cuando este bit esté activado, puede dibujar sin problemas en la pantalla sin tener que ocultar antes el puntero del ratón. Tenga en cuenta que no todo cursor gráfico puede ser implementado via hardware: en particular, VBE/AF sólo soporta imágenes de 2 colores y de hasta 32x32 pixels, donde el segundo color es un inverso exacto del primero. Esto significa que Allegro puede necesitar alternar

entre cursores por hardware o software en cualquier punto durante la ejecución de su programa, por lo que no debe asumir que este bit será constante durante largos periodos de tiempo. Sólo le dice si un cursor hardware está siendo usado ahora, y puede cambiar cuando oculte/enseñe el puntero.

· GFX_HW_HLINE:
Indica que la versión opaca normal de la función hline() está implementada usando aceleración por hardware. Esto incrementará el rendimiento, no sólo de hline(), sino también de otras funciones que la usen internamente, por ejemplo circlefill(), triangle(), y floodfill().

· GFX_HW_HLINE_XOR:
Indica que la versión XOR de la función hline(), y cualquier otra que la use, estan implementadas usando aceleración por hardware.

· GFX_HW_HLINE_SOLID_PATTERN:
Indica que los modos sólidos y con patrones de la función hline(), y cualquier otra función que la use, estan implementadas usando aceleración por hardware (lea nota abajo).

· GFX_HW_HLINE_COPY_PATTERN:
Indica que el modo copia de patrón de la función hline(), y cualquier otra función que la use, estan implementadas usando aceleración por hardware (lea nota abajo).

· GFX_HW_FILL:
Indica que la versión opaca de las funciones rectfill(), clear_bitmap() y clear_to_color(), están implementadas usando aceleración por hardware.

· GFX_HW_FILL_XOR:
Indica que la versión XOR de la función rectfill() está implementada usando aceleración por hardware.

· GFX_HW_FILL_SOLID_PATTERN:
Indica que los modos sólidos y con patrones de la función rectfill() están implementados usando aceleración por hardware (lea nota abajo).

· GFX_HW_FILL_COPY_PATTERN:
Indica que el modo copia de patrón de la función rectill() está implementado usando aceleración por hardware (lea nota abajo).

· GFX_HW_LINE:

Indica que la versión opaca de las funciones line() y vline() está implementada usando aceleración por hardware.

· GFX_HW_LINE_XOR:
Indica que la versión XOR de las funciónes line() y vline() está implementada usando aceleración por hardware.

· GFX_HW_TRIANGLE:
Indica que la versión opaca de la función triangle() está implementada usando aceleración por hardware.

· GFX_HW_TRIANGLE_XOR:
Indica que la versión XOR de la función triangle() está implementada usando aceleración por hardware.

· GFX_HW_GLYPH:
Indica que la expansión de carácter monocromo (para dibujo de texto) está implementada usando aceleración hardware.

· GFX_HW_VRAM_BLIT:
Indica que hacer un blit desde una parte de la pantalla a otra está implementado usando aceleración por hardware. Si este bit está activado, hacer blits dentro de la memoria de vídeo será ciertamente el modo más rápido para enseñar una imagen, por lo que sería útil almacenar parte de sus gráficos más usados en una posición oculta de la memoria de vídeo.

· GFX_HW_VRAM_BLIT_MASKED:
Indica que la rutina masked_blit() es capaz de hacer una copia de una parte de vídeo a otra usando aceleración por hardware, y que draw_sprite() usará copia por hardware cuando un sub-bitmap de la pantalla o un bitmap de memoria de vídeo sea la imagen origen. Si este bit está activado, el copiar desde la memoria de vídeo será casi seguramente el modo más rápido para visualizar una imagen, por lo que podría ser rentable almacenar algunos de sus sprites más usados en una porción no visible de la memoria de vídeo.

Aviso: isi esta bit no está activado, masked_blit() y draw_sprite() no funcionarán correctamente cuando los use con la memoria de vídeo como bitmap origen! Sólo puede usar estas funciones para copiar memoria de vídeo si están soportadas por el hardware.

· GFX_HW_MEM_BLIT:
Indica que hacer un blit desde un bitmap de memoria a la pantalla usa aceleración por hardware.

· GFX_HW_MEM_BLIT_MASKED:
Indica que masked_blit() y draw_sprite() usan aceleración por hardware cuando la imagen fuente es un bitmap de memoria, y la imagen destino es la pantalla física.

· GFX_HW_SYS_TO_VRAM_BLIT:
Indica que hacer un blit desde un bitmap de sistema a la pantalla está acelerado por hardware. Note que puede haber alguna aceleración incluso si este bit no está activado, porque los bitmaps de sistema también se pueden beneficiar del blit de memoria normal a la pantalla. Este bit sólo estará activado si los bitmaps de sistema una aceleración mayor que la proporcionada por GFX_HW_MEM_BLIT.

· GFX_HW_SYS_TO_VRAM_BLIT_MASKED:
Indica que las funciones masked_blit() y draw_sprite() están siendo aceleradas por hardware cuando la imagen fuente es un bitmap de sistema y el destino es la pantalla física. Note que puede haber alguna aceleración incluso si este bit no está activado, porque los bitmaps de sistema también se pueden beneficiar del blit de memoria normal a la pantalla. Este bit sólo estará activado si los bitmaps de sistema una aceleración mayor que la proporcionada por GFX_HW_MEM_BLIT_MASKED.

Nota: incluso cuando la información diga que el dibujo con patrón está soportado usando aceleración por hardware, no será posible para cualquier tamaño de patrón. El controlador VBE/AF sólo soporta patrones de hasta 8x8 pixels, y usará la versión original no acelerada por hardware de las rutinas de dibujo siempre que use patrones más grandes.

Nota2: estas características de aceleración por hardware sólo surtirán efecto cuando dibuje directamente sobre la pantalla física, o un sub-bitmap de ésta. La aceleración por hardware es útil sobre todo con una configuración de cambio de páginas o triple buffer, y probablemente no habrá diferencia de rendimiento con el sistema "dibuja en un bitmap de memoria, entonces cópialo a la pantalla".

- **enable_triple_buffer**

```
int enable_triple_buffer();
```

Si el bit GFX_CAN_TRIPLE_BUFFER de la variable gfx_capabilities no está activado, puede intentar activarlo llamando esta función. En particular, si está trabajando en modo-X bajo DOS puro, esta rutina activará el simulador de retrazo temporizado, el cual entonces activará las funciones de triple buffering. Devuelve cero si el triple buffering está activado.

- **scroll_screen**

int scroll_screen(int x, int y);

Intenta hacer un scroll de la pantalla para mostrar una parte diferente de la pantalla virtual (que inicialmente se posicionará en 0,0, que es la esquina superior izquierda). Devuelve cero si ha tenido éxito: puede fallar si el controlador gráfico no soporta scroll por hardware o la pantalla virtual no es lo suficientemente grande. Puede usar esta función para mover la pantalla por un espacio de pantalla virtual grande, o para hacer un intercambio de páginas entre dos áreas de pantalla virtual que no estén solapadas. Tenga en cuenta que para dibujar fuera de la posición original de la pantalla deberá alterar el área de recorte: mire abajo.

El scroll en Modo-X es de fiar y funcionará en cualquier tarjeta. Desafortunadamente la mayoría de las implementaciones VESA sólo pueden hacer scroll horizontal en incrementos de cuatro pixels, así que hacer un scroll suave en modos SVGA es imposible. Es vergonzoso, pero no veo forma de solucionarlo. Un numero significativo de implementaciones VESA parecen tener muchos fallos cuando hay que hacer scroll en modos truecolor, por lo que recomiendo no usar esta rutina en modos truecolor a menos que esté seguro de que Scitech Display Doctor está instalado.

Allegro se ocupará de cualquier sincronización del retrazo vertical cuando hago un scroll de pantalla, así que no necesita llamar a vsync() antes. Esto significa que scroll_screen() tiene los mismos efectos de retraso que vsync().

- **request_scroll**

int request_scroll(int x, int y);

Esta función es usada para el triple buffering. Hace una petición de scroll por hardware a la posición especificada, pero vuelve inmediatamente en vez de esperar un retrazo. El scroll tendrá lugar durante el siguiente retrazo vertical, pero puede seguir ejecutando su código mientras y usar la rutina poll_scroll() para detectar cuando ha ocurrido el cambio por scroll (vea examples/ex3buf.c). El triple buffering sólo es posible en cierto hardware: funcionará en cualquier resolución de modo-X si el simulador de retrazo esta activo(pero no funciona bien bajo win95), y está suportado por los controladores VBE 3.0 y VBE/AF para un número limitado de tarjetas gráficas de alto nivel. Puede mirar el bit GFX_CAN_TRIPLE_BUFFER del campo de bits gfx_capabilities para ver si funcionará con el controlador actual. Esta función devuelve cero si no hubo problemas.

- **poll_scroll**

int poll_scroll();

Esta función es usada con triple buffering. Comprueba el estado de un scroll por hardware iniciado previamente por la rutina request_scroll(), devolviendo no-cero si todavía está esperando a que ocurra, y cero si ya ha ocurrido.

- **show_video_bitmap**

```
int show_video_bitmap(BITMAP *bitmap);
```

Solicita intercambiar la página de pantalla hardware para visualizar el objeto bitmap de vídeo especificado, que debe tener el mismo tamaño que la pantalla física, y debe haber sido obtenido usando la función create_video_bitmap(). Devuelve cero si no hubo problemas. Esta función esperará un retrazo vertical si su tarjeta de vídeo lo requiere, por lo que no hace falta que llame vsync() manualmente.

- **request_video_bitmap**

```
int request_video_bitmap(BITMAP *bitmap);
```

Esta función se usa con triple buffering. Solicita intercambiar la página de pantalla al objeto bitmap de vídeo especificado, pero retorna inmediatamente en vez de esperar el retrazo. El intercambio tendrá lugar con el siguiente retrazo vertical, pero puede mientras puede seguir ejecutando su código y usar la rutina poll_scroll() para detectar cuándo ocurre el intercambio realmente. Triple buffering sólo es posible en determinado hardware: mire los comentarios de request_scroll(). Devuelve cero si no hubo problemas.

OBJETOS BITMAP

Una vez haya seleccionado un modo gráfico, puede dibujar cosas en la pantalla por el bitmap 'screen'. Todas las rutinas de dibujo de Allegro dibujan en estructuras BITMAP, que son áreas de memoria que contienen imágenes rectangulares, almacenadas en arrays de packs de bytes (un byte por pixel en modos de 8 bits, sizeof(short) bytes por pixel en modos de 15 y 16 bits por pixel, 3 bytes por pixel en modos de 24 bits y sizeof(long) bytes por pixel en modos de 32 bits). Puede crear y manipular bitmaps en la memoria RAM, o puede dibujar en el bitmap especial 'screen' que representa la memoria de vídeo de su tarjeta gráfica.

Por ejemplo, para dibujar un pixel en la pantalla escribiría: putpixel(screen, x, y, color);

O para implementar un sistema doble-buffer:

```
BITMAP *bmp = create_bitmap(320, 200);     // crea un bitmap en la RAM
clear_bitmap(bmp);                         // limpia el bitmap
putpixel(bmp, x, y, color);                // dibuja sobre el bitmap
blit(bmp, screen, 0, 0, 0, 0, 320, 200);   // lo copia a la pantalla
```

Mire abajo para saber cómo obtener acceso directo a la memoria de un bitmap.

Allegro soporta varios tipos diferentes de bitmaps:

- El bitmap screen, que representa la memoria de vídeo de su hardware. Debe dibujar sobre él para que su imagen sea visible.

- Bitmaps de memoria, que están situados en la RAM del sistema y pueden ser usados para almacenar gráficos o como espacios de dibujo temporales para sistemas doble buffer. Estos pueden ser obtenidos llamando create_bitmap(), load_pcx(), o al cargar un fichero de datos.
- Sub-bitmaps. Estos comparten memoria de imagen con un bitmap padre (que puede ser la pantalla, un bitmap de memoria u otro sub-bitmap), por lo que dibujar en ellos también modificará al bitmap padre. Pueden tener cualquier tamaño y estar localizados en cualquier parte del bitmap padre. Pueden tener sus propias áreas de recorte, por lo que son útiles para dividir un

bitmap en varias unidades más pequeñas, ej: partir una pantalla virtual grande en múltiples secciones (mire examples/exscroll.c).

- Bitmaps de memoria de vídeo. Estos son creados con la función create_video_bitmap(), y normalmente son implementados como sub-bitmaps del objeto screen.

- Bitmaps de sistema. Se crean mediante la función create_system_bitmap(), y están a caballo entre los bitmaps de memoria y los de video. Viven en la memoria de sistema, así pues no están limitado por la cantidad de ram de video de su tarjeta, pero se guardan en un formato específico de la plataforma que puede activar una aceleración hardware mejor de la que es posible con un bitmap de memoria normal (vea los bits GFX_HW_SYS_TO_VRAM_BLIT y GFX_HW_SYS_TO_VRAM_BLIT_MASKED en gfx_capabilities). A los bitmaps de sistema se debe acceder de la misma manera que a los bitmaps de video, usando las funciones de cambio de banco y las macro bmp_write*(). No todas las plataformas implementan este tipo de bitmap: si no están disponibles create_system_bitmap() funcionará idénticamente igual que create_bitmap().

- **screen**

```
extern BITMAP *screen;
```

Puntero global a un bitmap, de tamaño VIRTUAL_W x VIRTUAL_H. Esto es creado por set_gfx_mode(), y representa la memoria de vídeo de su hardware. Sólo una parte de este bitmap será visible, de tamaño SCREEN_W x SCREEN_H. Normalmente estará en la parte superior izquierda de la pantalla virtual, por lo que puede ignorar el resto de la pantalla virtual si no está interesado en scroll por hardware o intercambio de páginas. Para mover la ventana visible a otras partes del bitmap screen, llama scroll_screen(). Inicialmente el área de recorte será igual de grande que la pantalla física, por lo que si quiere dibujar en la pantalla virtual fuera de este rectángulo, deberá ajustar el área de recorte.

- **create_bitmap**

```
BITMAP *create_bitmap(int width, int height);
```

Crea un bitmap de memoria con tamaño width por height, y devuelve un puntero a él. El bitmap tendrá el área de recorte activada y ajustada al tamaño total del bitmap. La memoria de la imagen no será limpiada, por lo que probablemente tendrá basura: debería limpiar el bitmap antes de usarlo. Esta rutina usa siempre el formato global de profundidad de color especificado al llamar set_color_depth().

- **create_bitmap_ex**

BITMAP *create_bitmap_ex(int color_depth, int width, int height);

Crea un bitmap de una profundidad de color específica (8, 15, 16, 24 o 32 bits por pixel).

- **create_sub_bitmap**

BITMAP *create_sub_bitmap(BITMAP *parent, int x, y, width, height);

Crea un sub-bitmap, es decir, un bitmap que comparte memoria con un bitmap ya existente, pero posiblemente con un tamaño y área de recorte diferentes. Cuando cree un sub-bitmap de la pantalla en modo-X, la posición x debe ser un múltiplo de cuatro. La anchura y altura del sub-bitmap pueden extenderse fuera de los bordes del bitmap padre (serán recortados), pero el punto de origen debe estar en una región del bitmap padre.

- **create_video_bitmap**

BITMAP *create_video_bitmap(int width, int height);

Reserva un bitmap de memoria de vídeo del tamaño especificado, devolviendo un puntero si funcionó, o NULL si hubo algún error (ej: se has quedado sin memoria vram libre). Esto puede ser usado para reservar memoria de vídeo oculta para almacenar gráficos preparados para operaciones aceleradas por hardware, o para crear múltiples páginas de vídeo que luego pueden ser visualizadas con show_video_bitmap(). Los bitmaps de memoria de vídeo son normalmente reservados usando el mismo espacio que el bitmap screen, ya que pueden sobreescribirlo: por lo tanto no es una buena idea usar screen al mismo tiempo que las superficies devueltas por esta función.

- **create_system_bitmap**

BITMAP *create_system_bitmap(int width, int height);

Crea un bitmap en memoria de sistema del tamaño especificado, devolviendo un puntero a él si no hubo problemas o NULL en caso contrario.

- **destroy_bitmap**

void destroy_bitmap(BITMAP *bitmap);

Destruye un bitmap de memoria, sub-bitmap, bitmap de memoria de vídeo o un bitmap de sistema cuando ya no lo necesite.

- **lock_bitmap**

void lock_bitmap(BITMAP *bitmap);

Bajo DOS, fija toda la memoria usada por un bitmap. Normalmente no necesita llamar a esta función a no ser que esté haciendo cosas realmente raras en su programa.

- **bitmap_color_depth**

```
int bitmap_color_depth(BITMAP *bmp);
```

Devuelve la profundidad de color del bitmap especificado (8, 15, 16, 24, o 32).

- **bitmap_mask_color**

```
int bitmap_mask_color(BITMAP *bmp);
```

Devuelve el color de máscara del bitmap especificado (el valor que es ignorado al dibujar sprites). En bitmaps de 256 colores es cero, y en bitmaps truecolor es rosa fucsia (rojo y azul al máximo, verde a cero).

- **is_same_bitmap**

```
int is_same_bitmap(BITMAP *bmp1, BITMAP *bmp2);
```

Devuelve TRUE si ambos bitmaps describen el mismo área de dibujo, ej: sus punteros son iguales, uno es un sub-bitmap del otro, o ambos son sub-bitmaps de un padre común.

- **is_linear_bitmap**

```
int is_linear_bitmap(BITMAP *bmp);
```

Devuelve TRUE si bmp es un bitmap lineal, es decir, es un bitmap de memoria, pantalla en modo 13h o SVGA. En bitmaps lineales puede usar las funciones _putpixel(), _getpixel(), bmp_write_line(), y bmp_read_line().

- **is_planar_bitmap**

```
int is_planar_bitmap(BITMAP *bmp);
```

Devuelve TRUE si bmp es un bitmap de pantalla plano (modo-X o Xtended).

- **is_memory_bitmap**

```
int is_memory_bitmap(BITMAP *bmp);
```

Devuelve TRUE si bmp es un bitmap de memoria, es decir, que fue creado llamando create_bitmap() o cargado de un fichero de datos o una imagen. Los bitmaps de memoria pueden ser accedidos con los punteros de línea de la estructura bitmap, ej: bmp->line[y][x] = color.

- **is_screen_bitmap**

```
int is_screen_bitmap(BITMAP *bmp);
```

Devuelve TRUE si bmp es el bitmap screen, o un sub-bitmap de él.

- **is_video_bitmap**

```
int is_video_bitmap(BITMAP *bmp);
```

Devuelve TRUE si bmp es el bitmap screen, o un bitmap de memoria de video, o un sub-bitmap de alguno de ellos.

- **is_system_bitmap**

```
int is_system_bitmap(BITMAP *bmp);
```

Devuelve TRUE si bmp es un bitmap de sistema, o un sub-bitmap de uno.

- **is_sub_bitmap**

```
int is_sub_bitmap(BITMAP *bmp);
```

Devuelve TRUE si bmp es un sub-bitmap.

- **acquire_bitmap**

```
void acquire_bitmap(BITMAP *bmp);
```

Bloquea el bitmap de memoria de vídeo especificado antes de dibujar en él. Esto no se aplica a los bitmaps de memoria, y sólo afecta a algunas plataformas (Windows lo necesita, DOS no). Estas llamadas no son estrictamente necesarias, porque las rutinas de dibujo adquirirán el bitmap automáticamente antes de acceder a él, pero bloquear una superficie de DirectDraw es muy lento, y obtendrá mucho más rendimiento si adquiere la pantalla sólo una vez antes del inicio de la función de redibujado principal, y la suelta cuando el dibujado ha concluido completamente. Múltiples llamadas de adquisición serán anidadas, y el bitmap sólo será liberado cuando el contador de bloqueo sea cero. Tenga cuidado porque los programas DirectX activan un bloqueo de exclusión mutua (mutex) siempre que una superficie es bloqueada, lo que impide que reciban cualquier mensaje de entrada, iasí que debe asegurarse de liberar todos sus bitmaps antes de usar alguna rutina de temporización, teclado o cualquier otra rutina no gráfica!

- **release_bitmap**

```
void release_bitmap(BITMAP *bmp);
```

Libera un bitmap que fue bloqueado previamente mediante una llamada a acquire_bitmap(). Si el bitmap fue bloqueado varias veces, deberá liberarlo el mismo número de veces para que esté verdaderamente desbloqueado.

- **acquire_screen**

```
void acquire_screen();
```

Sinónimo de acquire_bitmap(screen);

- **release_screen**

```
void release_screen();
```

Sinónimo de release_bitmap(screen);

- **set_clip**

```
void set_clip(BITMAP *bitmap, int x1, int y1, int x2, int y2);
```

Cada bitmap tiene un área de recorte asociada, que es el área de pantalla sobre la que puede dibujar. Nada será dibujado fuera de este espacio. Pase las dos esquinas opuestas del área de recorte: éstas son inclusivas, ej: set_clip(bitmap, 16, 16, 32, 32) le permitirá dibujar en (16, 16) y (32, 32), pero no en (15, 15) o (33, 33). Si x1, y1, x2 e y2 son cero, el área de recorte se desactivará, lo que puede acelerar algunas operaciones de dibujo (normalmente casi nada, aunque cualquier poco ayuda) pero harán que su programa muera horriblemente si intenta dibujar fuera de los bordes del bitmap.

CARGANDO IMÁGENES

Aviso: cuando use imágenes truecolor, siempre debe ajustar el modo gráfico antes de cargar cualquier dato. De otro modo el formato de pixel (RGB o BGR) será desconocido, y el fichero podría ser convertido erróneamente.

- **load_bitmap**

```
BITMAP *load_bitmap(const char *filename, RGB *pal);
```

Carga un bitmap desde un fichero, devolviendo un puntero al bitmap y almacenando los datos de la paleta en el lugar especificado, que debería ser un array de 256 estructuras RGB. Es responsable de destruir el bitmap cuando ya no lo necesite. Devuelve NULL si hubo errores. Por ahora esta función soporta ficheros BMP, LBM, PCX y TGA, determinando el tipo por la extensión del fichero. Si el fichero contiene una imagen truecolor, debe ajustar el modo de vídeo o llamar set_color_conversion() antes de cargarlo.

- **load_bmp**

```
BITMAP *load_bmp(const char *filename, RGB *pal);
```

Carga un fichero Windows u OS/2 de 256 colores o 24 bits truecolor.

- **load_lbm**

```
BITMAP *load_lbm(const char *filename, RGB *pal);
```

Carga un fichero IFF ILBM/PBM de 256 colores.

- **load_pcx**

```
BITMAP *load_pcx(const char *filename, RGB *pal);
```

Carga un fichero PCX de 256 colores o 24 bits truecolor.

- **load_tga**

```
BITMAP *load_tga(const char *filename, RGB *pal);
```

Carga un fichero TGA de 256 colores, 15 bits hicolor, 24 bits truecolor o 32 bits truecolor + canal alpha.

- **save_bitmap**

```
int save_bitmap(const char *filename, BITMAP *bmp, const RGB *pal);
```

Escribe un bitmap en un fichero, usando la paleta especificada, que debería ser un array de 256 estructuras RGB. Devuelve distinto de cero si hubo errores. El formato de salida es determinado por la extensión del fichero: actualmente esta función soporta ficheros BMP, PCX o TGA. Una de las cosas con las que tener cuidado: si usa esto para volcar la pantalla en un fichero, puede acabar con una imagen más grande de lo esperado, ya que Allegro suele crear pantallas virtuales más grandes que la pantalla visible. Puede solucionar esto usando un sub-bitmap que especifica la parte de la pantalla que desea salvar, ejemplo:

```
BITMAP *bmp;
PALETTE pal;
get_palette(pal);
bmp = create_sub_bitmap(screen, 0, 0, SCREEN_W,SCREEN_H);
save_bitmap("pantalla.pcx", bmp, pal);
destroy_bitmap(bmp);
```

- **save_bmp**

```
int save_bmp(const char *filename, BITMAP *bmp, RGB *pal);
```

Escribe un bitmap en un fichero BMP de 256 colores o 24 bits truecolor.

- **save_pcx**

```
int save_pcx(const char *filename, BITMAP *bmp, RGB *pal);
```

Escribe un bitmap en un fichero PCX de 256 colores o 24 bits truecolor.

- **save_tga**

```
int save_tga(const char *filename, BITMAP *bmp, RGB *pal);
```

Escribe un bitmap en un fichero TGA de 256 colores, 15 bits hicolor, 24 bits truecolor o 32 bits truecolor + canal alpha.

- **register_bitmap_file_type**

```
void register_bitmap_file_type(const char *ext, BITMAP *(*load)(const char *filename, RGB *pal), int (*save)(const char *filename, BITMAP *bmp, const RGB *pal));
```

Informa a las funciones load_bitmap() y save_bitmap() de un nuevo tipo de fichero, dando rutinas para leer o escribir imágenes en este formato (cualquier función puede ser NULL).

- **set_color_conversion**

void set_color_conversion(int mode);

Especifica cómo convertir imágenes entre diferentes profundidades de color cuando se leen gráficos desde archivos externos o ficheros de datos. El modo es una máscara de bits especificando qué tipos de conversión se permiten. Si se activa el bit apropiado, los datos serán convertidos al formato actual de pixel (seleccionado llamando a la función set_color_depth()), si nó, se dejarán en el mismo formato que tenían en el fichero, debiendo entonces convertir manualmente el gráfico antes de poder mostrarlo. El modo por defectos es la conversión total, así que todas las imágenes se cargarán en el modo apropiado para el modo de vídeo actual. Los bits válidos son:

COLORCONV_NONE	// desactiva las conversiones entre formatos
COLORCONV_8_TO_15	// expande 8 bits a 15
bitsCOLORCONV_8_TO_16	// expande 8 bits a 16
bitsCOLORCONV_8_TO_24	// expande 8 bits a 24
bitsCOLORCONV_8_TO_32	// expande 8 bits a 32
bitsCOLORCONV_15_TO_8	// reduce 15 bits a 8
bitsCOLORCONV_15_TO_16	// expande 15 bits a 16
bitsCOLORCONV_15_TO_24	// expande 15 bits a 24
bitsCOLORCONV_15_TO_32	// expande 15 bits a 32
bitsCOLORCONV_16_TO_8	// reduce 16 bits a 8
bitsCOLORCONV_16_TO_15	// reduce 16 bits a 15
bitsCOLORCONV_16_TO_24	// expande 16 bits a 24
bitsCOLORCONV_16_TO_32	// expande 16 bits a 32
bitsCOLORCONV_24_TO_8	// reduce 24 bits a 8
bitsCOLORCONV_24_TO_15	// reduce 24 bits a 15
bitsCOLORCONV_24_TO_16	// reduce 24 bits a 16
bitsCOLORCONV_24_TO_32	// expande 24 bits a 32
bitsCOLORCONV_32_TO_8	// reduce 32 bits en RGB a 8
bitsCOLORCONV_32_TO_15	// reduce 32 bits en RGB a 15
bitsCOLORCONV_32_TO_16	// reduce 32 bits en RGB a 16
bitsCOLORCONV_32_TO_24	// reduce 32 bits en RGB a 24
bitsCOLORCONV_32A_TO_8	// reduce 32 bits en RGBA a 8
bitsCOLORCONV_32A_TO_15	// reduce 32 bits en RGBA a 15
bitsCOLORCONV_32A_TO_16	// reduce 32 bits en RGBA a 16
bitsCOLORCONV_32A_TO_24	// reduce 32 bits en RGBA a 24
bitsCOLORCONV_DITHER_PAL	// difumina al reducir a 8
bitsCOLORCONV_DITHER_HI	// difumina al reducir a
hicolorCOLORCONV_KEEP_TRANS	// mantiene la transparencia original

Por conveniencia, las siguientes macros pueden usarse para seleccionar combinaciones comunes de los bits anteriores.

COLORCONV_EXPAND_256	// expande 256 colores a hi/truecolor
COLORCONV_REDUCE_TO_256	// reduce hi/truecolor a 256 colors
COLORCONV_EXPAND_15_TO_16	// expande hicolor de 15 bit a 16 bits
COLORCONV_REDUCE_16_TO_15	// reduce hicolor de 16 bits a 15 bits
COLORCONV_EXPAND_HI_TO_TRUE	// expande 15/16 bits a 24/32 bits
COLORCONV_REDUCE_TRUE_TO_HI	// reduce 24/32 bits a 15/16 bits
COLORCONV_24_EQUALS_32	// convierte entre 24 and 32 bits
COLORCONV_TOTAL	// todo al formato actual
COLORCONV_PARTIAL	// convierte 15<>16 y 24<->32
COLORCONV_MOST	// todas excepto hi/truecolor <-> 256

Si activa el bit COLORCONV_DITHER, el difuminado se efectuará siempre que los gráficos truecolor se conviertan a formato hicolor o modo con paleta, incluyendo la función blit(), y cualquier rutina de conversión automática que se ejecute cuando lea gráficos del disco. Esto puede producir resultados visuales más atractivos, pero obviamente es mucho más lento que una conversión directa.

Si intenta usar bitmaps convertidos con funciones como masked_blit() o draw_sprite(), debería especificar el bit COLORCONV_KEEP_TRANS. Se asegurará de que las áreas de máscara del bitmap antes y después de la conversión se mantendrán iguales, convirtiendo los colores transparentes de un modo a otro y ajustando aquellos colores que tras la conversión podrían ser reconvertidos en transparentes. Este bit afecta a cualquier llamada blit() entre formatos de pixel distintos y a cualquier conversión automática.

RUTINAS DE PALETA

Todas las funciones de dibujo de Allegro usan parámetros en enteros para representar colores. En las resoluciones truecolor estos números codifican el color directamente como una colección de bits rojos, verdes y azules, pero en el modo normal de 256 colores, los valores son tratados como índices de la paleta actual, que es una tabla que contiene las intensidades de rojo, verde y azul de cada uno de los 256 colores posibles.

La paleta se almacena con estructuras RGB, que contienen intensidades de rojo, verde y azul en el formato hardware de la VGA, que van de 0 a 63, y son definidas así:

```
typedef struct RGB
{
      unsigned char r, g, b;
} RGB;
```

Por ejemplo:

```
RGB negro  = { 0,  0,  0  };
RGB blanco = { 63, 63, 63 };
RGB verde  = { 0,  63, 0  };
RGB gris = { 32, 32, 32 };
```

El tipo PALETTE es definido como un array de 256 estructuras RGB.

Puede notar que gran parte del código de Allegro escribe 'palette' como 'pallete'. Esto es porque los ficheros de cabecera de mi viejo compilador Mark Williams del Atari lo escribían con dos l's, y estoy acostumbrado a eso. Allegro aceptará sin problemas ambas escrituras, debido a algunos #defines en allegro.h.

- **vsync**

```
void vsync();
```

Espera a que empiece un retrazo vertical. El retrazo ocurre cuando el rayo de electrones de su monitor ha llegado a la parte inferior de la pantalla y está volviendo arriba para hacer otro barrido. Durante este corto periodo de tiempo la tarjeta de vídeo no manda datos al monitor, por lo que puede hacer cosas

que de otra forma no podría, como alterar la paleta sin causar parpadeo (nieve). Sin embargo Allegro esperará automáticamente el retrazo vertical antes de alterar la paleta o hacer scroll por hardware, por lo que normalmente no debe preocuparse por esta función.

- **set_color**

```
void set_color(int index, const RGB *p);
```

Cambia la entrada de la paleta especificada al triplete RGB dado. A diferencia de otras funciones de paleta, esto no hace sincronización con el retrazo, por lo que debería llamar vsync() antes para evitar problemas de nieve.

- **_set_color**

```
void _set_color(int index, const RGB *p);
```

Esta es una versión inline de set_color(), que puede usar en la función callback del simulador de retrazo vertical. Sólo debería ser usada en VGA modo 13h y modo-X, porque algunos de las recientes SVGAs no son compatibles con la VGA (set_color() y set_palette() usarán llamadas VESA en estas tarjetas, pero _set_color() no sabrá nada de eso).

- **set_palette**

```
void set_palette(const PALETTE p);
```

Ajusta la paleta entera de 256 colores. Debe pasar un array de 256 estructuras RGB. A diferencia de set_color(), no hace falta llamar vsync() antes de esta función.

- **set_palette_range**

```
void set_palette_range(const PALETTE p, int from, int to, int vsync);
```

Ajusta las entradas de la paleta desde from hasta to (inclusivos: pase 0 y 255 para ajustar la paleta entera). Si vsync está activado, espera un retrazo vertical, de otro modo cambia los colores inmediatamente.

- **get_color**

```
void get_color(int index, RGB *p);
```

Recupera la entrada de la paleta especificada.

- **get_palette**

```
void get_palette(PALETTE p);
```

Recupera la paleta entera de 256 colores. Debe proveer un array de 256 estructuras RGB para almacenar ahí los colores.

- **get_palette_range**

void get_palette_range(PALETTE p, int from, int to);

Recupera las entradas de la paleta desde from hasta to (inclusivos: pase 0 y 255 para recuperar la paleta entera).

- **fade_interpolate**

void fade_interpolate(const PALETTE source, const PALETTE dest, PALETTE output, int pos, int from, to);

Calcula una paleta temporal en un sitio entre source y dest, devolviéndola en el parámetro output. La posición entre los dos extremos es especificada por el valor pos: 0 devuelve una copia exacta de source, 64 devuelve dest, 32 devuelve una paleta a medio camino entre las dos, etc. Esta rutina sólo afecta a los colores desde from hasta to (inclusivos: pase 0 y 255 para interpolar la paleta entera).

- **fade_from_range**

void fade_from_range(const PALETTE source, const PALETTE dest, int speed, int from, to);

Funde gradualmente una parte de la paleta desde la paleta source hasta la paleta dest. La velocidad va de 1 (lento) a 64 (instantáneo). Esta rutina sólo afecta los colores desde from hasta to (inclusivos: pase 0 y 255 para fundir la paleta entera).

- **fade_in_range**

void fade_in_range(const PALETTE p, int speed, int from, to);

Funde gradualmente una parte de la paleta desde una pantalla negra hasta la paleta especificada. La velocidad va de 1 (lento) a 64 (instantáneo). Esta rutina sólo afecta los colores desde from hasta to (inclusivos: pase 0 y 255 para fundir la paleta entera).

- **fade_out_range**

void fade_out_range(int speed, int from, to);

Funde gradualmente una parte de la paleta desde la paleta actual hasta una pantalla negra. La velocidad va de 1 (lento) a 64 (instantáneo). Esta rutina sólo afecta los colores desde from hasta to (inclusivos: pase 0 y 255 para fundir la paleta entera).

- **fade_from**

void fade_from(const PALETTE source, const PALETTE dest, int speed);

Funde gradualmente desde la paleta source hasta la paleta dest. La velocidad va de 1 (lento) a 64 (instantáneo).

- **fade_in**

void fade_in(const PALETTE p, int speed);

Funde gradualmente desde una pantalla negra a la paleta especificada. La velocidad va de 1 (lento) a 64 (instantáneo).

- **fade_out**

void fade_out(int speed);

Funde gradualmente la paleta actual hasta una pantalla negra. La velocidad va de 1 (lento) a 64 (instantáneo).

- **select_palette**

void select_palette(const PALLETE p);

Rutina fea que puede usar en algunas situaciones peculiares cuando necesita convertir entre formatos de imagen con paleta a truecolor. Ajusta la tabla de la paleta interna de la misma forma que la función set_palette(), para que la conversión use la paleta especificada, pero sin afectar de ningún modo al hardware de visualización. La paleta antigua es almacenada en un buffer interno, y puede ser recuperada llamando unselect_palette().

- **unselect_palette**

void unselect_palette();

Recupera la tabla de la paleta que estaba en uso antes de la última llamada a select_palette().

- **generate_332_palette**

void generate_332_palette(PALETTE pal);

Construye una paleta truecolor falsa, usando tres bits para el rojo y el verde y dos para el azul. La función load_bitmap() devuelve esto si el fichero no contiene ninguna paleta (ej. cuando lee un bitmap truecolor).

- **generate_optimized_palette**

int generate_optimized_palette(BITMAP *bmp, PALETTE pal, const char rsvd[256]);

Genera una paleta de 256 colores óptima para hacer una versión reducida, en cuanto a color, de la imagen truecolor especificada. El parámetro rsvd apunta a una tabla que indica qué colores se le permite modificar a la función: cero para colores libres que pueden ser asignados como el optimizador quiera, valores negativos para colores reservados que no pueden usarse, y valores positivos para entradas fijas de la paleta que no deben cambiarse, pero que se pueden usar en la optimización.

- **default_palette**

extern PALETTE default_palette;

La paleta por defecto de la BIOS IBM. Se seleccionará automáticamente cuando active un nuevo modo gráfico.

- **black_palette**

extern PALETTE black_palette;

Una paleta que contiene colores negros sólidos, usada por las rutinas de fundidos.

- **desktop_palette**

extern PALETTE desktop_palette;

La paleta usada por el escritorio de baja resolución del Atari ST. No estoy seguro por qué esto sigue aquí, excepto porque los programas grabber y test la usan. Es probablemente el único código heredado del Atari que queda en Allegro, y sería una pena quitarlo :-)

FORMATOS DE PIXEL TRUECOLOR

En los modos de vídeo truecolor, los componentes rojo, verde y azul de cada pixel son empaquetados directamente en el valor del color, en vez de ser un índice a una tabla de colores. En los modos de 15 bits hay 5 bits para cada color, en modos de 16 bits hay 5 bits para rojo y azul y seis para verde, y los modos de 24 y 32 bits usan ambos 8 bits para cada color (los pixels de 32 bits simplemente tienen un byte extra para alinear los datos). El mapa de estos componentes puede variar entre ordenadores, pero generalmente será RGB o BGR. ¡Ya que el mapa no es conocido hasta que seleccione el modo de vídeo que vaya a usar, debe llamar set_gfx_mode() antes de usar cualquiera de las siguientes rutinas!

- **makecol8**

int makecol8(int r, int g, int b); int makecol15(int r, int g, int b); int makecol16(int r, int g, int b); int makecol24(int r, int g, int b); int makecol32(int r, int g, int b);

Estas rutinas convierten los colores desde un formato independiente del hardware (rojo, verde y azul que van de 0 a 255) a varios formatos de pixel dependientes del hardware. Convertir entre formatos de 15, 16, 24 o 32 bits sólo requiere algunos desplazamientos de bits, por lo que es eficiente. Convertir hacia un color de 8 bits requiere buscar la paleta para encontrar el color más parecido, algo que es muy lento a no ser que haya creado un mapa RGB (mire abajo).

- **makeacol32**

int makeacol32(int r, int g, int b, int a);

Convierte un color RGBA en un formato de pixel de pantalla de 32 bits, que incluye un valor alfa (de transparencia). No hay versiones de esta rutina para otras profundidades de color, porque sólamente el formato de 32 bits tiene espacio suficiente para guardar un canal alfa útil. Debería usar colores en formato RGBA sólamente como entrada para draw_trans_sprite() o draw_trans_rle_sprite() después de llamar a set_alpha_blender(), en vez de dibujarlos directamente en la pantalla.

- **makecol**

int makecol(int r, int g, int b);

Convierte colores desde un formato independiente del hardware (rojo, verde y azul que van de 0 a 255) al formato de pixel requerido por el modo de vídeo actual, llamando a las funciones previas makecol de 8, 15, 16, 24, o 32 bits según convenga.

- **makecol_depth**

int makecol_depth(int color_depth, int r, int g, int b);

Convierte colores desde un formato independiente del hardware (rojo, verde y azul que van de 0 a 255) al formato de pixel requerido por la profundidad de color especificada.

- **makeacol**

int makeacol(int r, int g, int b, int a); int makeacol_depth(int color_depth, int r, int g, int b, int a);

Convierte colores RGBA en un formato de pixel dependiente de la pantalla. Si está en un modo inferior al de 32 bits, esto es lo mismo que llamar a makecol() o makecol_depth(), pero al usar esta rutine es posible crear valores de color de 32 bits que contienen un canal alpha verdadero de 8 bits junto con los componentes rogo, verde y azul. Sólo debería usar colores en formato RGBA como dato de entrada para draw_trans_sprite(), o draw_trans_rle_sprite() tras haber llamado a set_alpha_blender(), en vez de dibujarlos directamente en la pantalla.

- **makecol15_dither**

int makecol15_dither(int r, int g, int b, int x, int y);int makecol16_dither(int r, int g, int b, int x, int y);

Dados dos valores y coordenadas de pixel, calcula una valor RGB difuminado de 15 o 16 bits. Esto puede producir mejores resultados al reducir imágenes de truecolor a hicolor. Aparte de llamar estas funciones diréctamente, el difuminado hicolor puede ser activado automáticamente al cargar gráficos llamando la función set_color_conversion(), por ejemplo set_color_conversion(COLORCONV_REDUCE_TRUE_TO_HI|COLORCONV_DITHER).

- **getr8**

int getr8(int c); int getg8(int c); int getb8(int c); int getr15(int c); int getg15(int c); int getb15(int c); int getr16(int c); int getg16(int c); int getb16(int c); int getr24(int c); int getg24(int c); int getb24(int c); int getr32(int c); int getg32(int c); int getb32(int c);

Dado un color en un formato dependiente del hardware, estas funciones extraen uno de los componentes rojo, verde o azul (de 0 a 255).

- **geta32**

int geta32(int c);

Dado un color en un formato de píxel en 32 bits, esta función extrae el componente alfa (de 0 a 255).

- **getr**

int getr(int c); int getg(int c); int getb(int c); int geta(int c);

Dado un color en el formato usado por el modo de vídeo actual, estas funciones extraen uno de los componentes rojo, verde, azul o alfa (de 0 a 255), llamando a las funciones previas get de 8, 15, 16, 24 o 32 bits según convenga. La parte alfa sólo tiene sentido para píxels de 32 bits.

- **getr_depth**

int getr_depth(int color_depth, int c); int getg_depth(int color_depth, int c); int getb_depth(int color_depth, int c); int geta_depth(int color_depth, int c);

Dado un color en el formato usado por la profundidad de color especificada, estas funciones extraen un componente rojo, verde, azul o alfa (de 0 a 255). La parte alfa sólo tiene sentido para píxels de 32 bits.

- **palette_color**

extern int palette_color[256];

Tabla de mapa de la paleta que convierte un color (0-255) en el formato de pixel que está siendo usado por el modo de vídeo. En un modo de 256 colores esto simplemente apunta al índice del array. En modos truecolor mira el valor especificado en la paleta actual, y lo convierte a ese valor RGB en el formato de pixel empaquetado apropiado.

MASK_COLOR_8
#define MASK_COLOR_8 0
#define MASK_COLOR_15 (5.5.5 pink)
#define MASK_COLOR_16 (5.6.5 pink) #define MASK_COLOR_24 (8.8.8 pink)
#define MASK_COLOR_32 (8.8.8 pink)

Constantes que representan los colores usados para enmascarar los pixels transparentes de los sprites para cada profundidad de color. En resoluciones de 256 colores es cero, y en modos truecolor es rosa fucsia (rojo y azul al máximo, verde a cero).

Excepto _putpixel(), todas estas rutinas son afectadas por el modo actual de dibujo y el área de recorte del bitmap destino.

- **putpixel**

void putpixel(BITMAP *bmp, int x, int y, int color);

Escribe un pixel en la posición especificada del bitmap, usando el modo de dibujo actual y el área de recorte del bitmap.

- **_putpixel**

void _putpixel(BITMAP *bmp, int x, int y, int color); void _putpixel15(BITMAP *bmp, int x, int y, int color); void _putpixel16(BITMAP *bmp, int x, int y, int color); void _putpixel24(BITMAP *bmp, int x, int y, int color); void _putpixel32(BITMAP *bmp, int x, int y, int color);

Como el putpixel() normal, pero mucho más rápidas porque están implementadas como funciones de ensamblador en línea para profundidades de color específicas. No funcionarán en modos gráficos de tipo Modo-X, no soportan áreas de recorte (ise bloquearán si intenta dibujar fuera del bitmap!), e ignoran el modo de dibujo.

- **getpixel**

int getpixel(BITMAP *bmp, int x, int y);

Lee el pixel del punto x, y en el bitmap. Devuelve -1 si el punto está fuera del bitmap.

- **_getpixel**

int _getpixel(BITMAP *bmp, int x, int y); int _getpixel15(BITMAP *bmp, int x, int y); int _getpixel16(BITMAP *bmp, int x, int y); int _getpixel24(BITMAP *bmp, int x, int y); int _getpixel32(BITMAP *bmp, int x, int y);

Versiones más rápidas de getpixel() para profundidades de color específicas. No funcionarán en modo-X y no soportan áreas de recorte, así que debe estar seguro que el punto está dentro del bitmap.

- **vline**

void vline(BITMAP *bmp, int x, int y1, int y2, int color);

Dibuja una línea vertical en el bitmap, desde (x, y1) hasta (x, y2).

- **hline**

```
void hline(BITMAP *bmp, int x1, int y, int x2, int color);
```

Dibuja una línea horizontal en el bitmap, desde (x1, y) hasta (x2, y).

- **do_line**

```
void do_line(BITMAP *bmp, int x1, y1, x2, y2, int d, void(*proc)());
```

Calcula todos los puntos de una línea desde el punto (x1, y1) hasta (x2, y2), llamando la función proc para cada pixel. A ésta función se le pasa una copia del parámetro bmp, la posición x e y, y el parámetro d, por lo que puede llamar la función putpixel().

- **line**

```
void line(BITMAP *bmp, int x1, int y1, int x2, int y2, int color);
```

Dibuja una línea en el bitmap, desde (x1, y1) hasta (x2, y2).

- **triangle**

```
void triangle(BITMAP *bmp, int x1, y1, x2, y2, x3, y3, int color);
```

Dibuja un triángulo relleno entre los tres puntos.

- **polygon**

```
void polygon(BITMAP *bmp, int vertices, int *points, int color);
```

Dibuja un polígono relleno con un número arbitrario de vértices. Pase el número de vértices y un array que contenga series de puntos x e y (hasta un total del valor vertices*2).

- **rect**

```
void rect(BITMAP *bmp, int x1, int y1, int x2, int y2, int color);
```

Dibuja los bordes de un rectángulo con los dos puntos dados como esquinas opuestas.

- **rectfill**

```
void rectfill(BITMAP *bmp, int x1, int y1, int x2, int y2, int color);
```

Dibuja un rectángulo sólido con los dos puntos dados como esquinas opuestas.

- **do_circle**

```
void do_circle(BITMAP *bmp, int x, int y, int radius, int d, void (*proc)());
```

Calcula todos los puntos de un círculo alrededor de (x, y) con el radio r, llamando a la función proc para cada pixel. A ésta función se le pasa una copia del parámetro bmp, la posición x e y, y el parámetro d, por lo que puede llamar la función putpixel().

- **circle**

```
void circle(BITMAP *bmp, int x, int y, int radius, int color);
```

Dibuja un círculo con el centro y radio especificados.

- **circlefill**

```
void circlefill(BITMAP *bmp, int x, int y, int radius, int color);
```

Dibuja un círculo relleno con el centro y radio especificados.

- **do_ellipse**

```
void do_ellipse(BITMAP *bmp, int x, y, int rx, ry, int d, void (*proc)());
```

Calcula todos los puntos de una elipse alrededor de (x, y) con el radio rx y ry, llamando a la función proc por cada pixel. A ésta función se le pasa una copia del parámetro bmp, la posición x e y, y el parámetro d, por lo que puede llamar la función putpixel().

- **ellipse**

```
void ellipse(BITMAP *bmp, int x, int y, int rx, int ry, int color);
```

Dibuja una elipse con el centro y radio especificados.

- **ellipsefill**

```
void ellipsefill(BITMAP *bmp, int cx, int cy, int rx, int ry, int color);
```

Dibuja una elipse rellena con el centro y radio especificados.

- **do_arc**

```
void do_arc(BITMAP *bmp, int x, y, fixed a1, a2, int r, d, void (*proc)());
```

Calcula todos los puntos en un arco circular alrededor del punto (x, y) con radio r, llamando a la función proc por cada uno de ellos. A ésta se le pasará una copia del parámetro bmp, la posición x e y, y una copia del parámetro d, por lo que puede usar putpixel(). El arco será pintado en sentido antihorario empezando desde el ángulo a1 y terminando en a2. Estos valores deben ser

especificados en fromato de punto fijo 16.16, siendo 256 un círculo total, 64 un ángulo recto, etc. Cero comienza a la derecha del punto central, y valores mayores rotan en dirección antihoraria desde ahí.

- **arc**

void arc(BITMAP *bmp, int x, y, fixed ang1, ang2, int r, int color);

Dibuja un arco circular con centro radio r y centro x, y, en dirección antihoraria empezando desde el ángulo a1 y terminando en a2. Estos valores deben ser especificados en fromato de punto fijo 16.16, siendo 256 un círculo total, 64 un ángulo recto, etc. Cero comienza a la derecha del punto central, y valores mayores rotan en dirección antihoraria desde ahí.

- **calc_spline**

void calc_spline(int points[8], int npts, int *x, int *y);

Calcula una serie de puntos npts a lo largo de una curva bezier, almacenándolos en los arrays x e y. La curva bezier es especificada por los cuatro puntos de control x/y del array points: points[0] y points[1] contienen las coordenadas del primer punto de control, points[2] y points[3] son el segundo punto de control, etc. Los puntos de control 0 y 3 son los extremos de la curva, y los puntos 1 y 2 son las guías. La curva probablemente no pasará por los puntos 1 y 2, pero estos afectan a la forma de la curva entre los puntos 0 y 3 (las líneas p0-p1 y p2-p3 son tangentes de la curva bezier). La forma más fácil de imaginárselo es pensar que la curva empieza en p0 en la dirección de p1, pero gira de tal forma que llega a p3 desde la dirección de p2. Además de su rol de primitivas gráficas, las curvas bezier pueden ser útiles para construir caminos alrededor de una serie de puntos de control, como en exspline.c.

- **spline**

void spline(BITMAP *bmp, int points[8], int color);

Dibuja una curva bezier usando los cuatro puntos de control especificados en el array points.

- **floodfill**

void floodfill(BITMAP *bmp, int x, int y, int color);

Rellena un área cerrada, comenzando en el punto (x, y), con el color especificado.

Todas estas rutinas son afectadas por el área de recorte del bitmap destino.

- **clear_bitmap**

void clear_bitmap(BITMAP *bitmap);

Limpia el bitmap con el color 0.

- **clear**

void clear(BITMAP *bitmap);

Un alias de clear_bitmap(), existe para mantener compatibilidad hacia atrás. Está implementado como una función estática inline. El alias puede ser desactivado definiendo el símbolo de preprocesador ALLEGRO_NO_CLEAR_BITMAP_ALIAS antes de incluir allegro.h:

```
#define ALLEGRO_NO_CLEAR_BITMAP_ALIAS
#include <allegro.h>
```

- **clear_to_color**

void clear_to_color(BITMAP *bitmap, int color);

Limpia el bitmap con el color especificado.

- **blit**

void blit(BITMAP *source, BITMAP *dest, int source_x, int source_y, int dest_x, int dest_y, int width, int height);

Copia un área rectangular del bitmap origen en el bitmap destino. Los parámetros source_x y source_y son de la esquina superior izquierda del área a copiar del bitmap origen, y dest_x y dest_y es la posición correspondiente en el bitmap destino. Esta rutina respeta el área de recorte del destino, y también habrá recorte si intenta copiar áreas que quedan fuera del bitmap origen.

Puede hacer un blit entre cualquier parte de dos bitmaps, incluso si las dos áreas se superponen (ejemplo: source y dest son el mismo bitmap, o uno es un sub-bitmap del otro). Debería tener en cuenta, sin embargo, que muchas tarjetas SVGA no tienen bancos de lectura/escritura separados, lo que significa que hacer un blit de una parte de la pantalla a otra requiere el uso de un

bitmap de memoria temporal, y es por ello extremadamente lento. Como regla general debería evitar hacer blits de la pantalla sobre sí misma en modos SVGA.

Sin embargo, hacer un blit en modo-X de una parte de la pantalla a otro lado puede ser significantemente más rápido que desde la memoria hacia la pantalla, siempre y cuando el origen y el destino estén correctamente alineados el uno con el otro. Hacer una copia entre áreas de la pantalla que se superponen es lento, pero si las áreas no se superponen, y si tienen la misma alineación de planos (es decir: (source_x%4) == (dest_x%4)), entonces se pueden usar los registros de la VGA para realizar una rápida transferencia de datos. Para tomar ventaja de esto, en modo-X normalmente se almacenan los gráficos en una parte oculta de la memoria de vídeo (usando una pantalla virtual grande), y se hacen blits desde allí a la parte visible de la pantalla.

Si el bit GFX_HW_VRAM_BLIT está activado en la variable gfx_capabilities, el controlador actual soporta blits de una parte de la pantalla a otra usando aceleración por hardware. Esto es extremadamente rápido, por lo que si este bit está activado, sería útil almacenar parte de sus gráficos más frequentemente usados en una porción oculta de la memoria de vídeo.

Al contrario que la mayoría de las rutinas gráficas, blit() permite que los bitmaps de origen y destino sean de diferentes profundidades de color, por lo que se puede usar para convertir imágenes de un formato de pixel a otro.

- **masked_blit**

```
void masked_blit(BITMAP *source, BITMAP *dest, int source_x, int source_y, int dest_x, int dest_y, int width, int height);
```

Como blit(), pero salta los pixels transparentes (cero en modos de 256 colores, y rosa fucsia para modos truecolor). La imagen origen debe ser un bitmap de memoria o un sub-bitmap, y las regiones de origen y destino no pueden superponerse.

Si el bit GFX_HW_VRAM_BLIT_MASKED está activado en la variable gfx_capabilities, el controlador actual soporta blits de una parte de la pantalla a otra usando aceleración por hardware. Esto es extremadamente rápido, por lo que si este bit está activado, sería útil almacenar parte de sus gráficos más frequentemente usados en una porción oculta de la memoria de vídeo.

Atención: isi el bit de aceleración por hardware no está activado, masked_blit() no funcionará correctamente cuando la imagen origen sea la memoria de vídeo, y el gráfico a dibujar

siempre tiene que ser un bitmap de memoria!

- **stretch_blit**

void stretch_blit(BITMAP *source, BITMAP *dest, int source_x, source_y, source_width, source_height, int dest_x, dest_y, dest_width, dest_height);

Como blit(), excepto que puede escalar imágenes de tal forma que las áreas de origen y destino no tienen que tener el mismo tamaño. Esta rutina no realiza tantas comprobaciones de seguridad como blit(): en particular debe tener cuidado de no copiar desde áreas fuera del bitmap origen, y no puede hacer la copia entre áreas que se superponen, y el bitmap origen y destino no pueden ser el mismo. Además, el origen debe ser un bitmap de memoria o sub-bitmap, no la pantalla hardware.

- **masked_stretch_blit**

void masked_stretch_blit(BITMAP *source, BITMAP *dest, int source_x, source_y, source_w, source_h, int dest_x, dest_y, dest_w, dest_h);

Como stretch_blit(), pero se salta pixels transparentes, que están marcados con un 0 en modos de 256 colores o magenta fara datos en truecolor (rojo y azul al máximo y el verde a cero). Las regiones origen (source) y destino (destination) no deben solaparse.

- **draw_sprite**

void draw_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y);

Dibuja una copia del bitmap sprite en el bitmap destino en la posición especificada. Eso es casi igual que blit(sprite, bmp, 0, 0, x, y, sprite->w, sprite->h), pero usa un modo de dibujado con máscara en el cual los pixels transparentes (cero en modos de 256 colores, rosa fucsia para modos truecolor) son ignorados, por lo que la imagen de fondo se podrá ver por las partes enmascaradas del sprite. El sprite debe ser un bitmap de memoria, no la pantalla o un subbitmap. El destino puede ser cualquier bitmap.

Si el bit GFX_HW_VRAM_BLIT_MASKED está activado en la variable gfx_capabilities, el controlador actual soporta blits de una parte de la pantalla a otra usando aceleración por hardware. Esto es extremadamente rápido, por lo que si este bit está activado, sería útil almacenar parte de sus gráficos más frequentemente usados en una porción oculta de la memoria de vídeo.

Atención: isi el bit de aceleración por hardware no está activado, draw_sprite() no funcionará correctamente cuando la imagen origen sea la memoria de vídeo, y el gráfico a dibujar siempre tiene que ser un bitmap de memoria!

A pesar de no soportar generalmente gráficos de diferentes profundidades de color, como caso especial puede usar esta función para dibujar imágenes origen de 256 colores en bitmaps destino truecolor, por lo que puede usar efectos de paleta en sprites específicos dentro de un programa truecolor.

- **draw_sprite_v_flip**

void draw_sprite_v_flip(BITMAP *bmp, BITMAP *sprite, int x, int y); void draw_sprite_h_flip(BITMAP *bmp, BITMAP *sprite, int x, int y); void draw_sprite_vh_flip(BITMAP *bmp, BITMAP *sprite, int x, int y);

Estas son como draw_sprite(), pero invierten la imagen sobre el eje vertical, horizontal o ambos. Esto produce imágenes espejo exactas, que no es lo mismo que rotar el sprite (y además esto es más rápido que la rutina de rotación). El sprite debe ser un bitmap de memoria.

- **draw_trans_sprite**

void draw_trans_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y);

Usa la tabla global color_map o las funciones de fundido truecolor para sobreimprimir un sprite encima de una imagen existente. Esto sólo debe usarse si antes se ha creado la tabla de colores (para modos de 256 colores) o las funciones de fundido (para modos truecolor). Ya que tiene que leer al igual que escribir en la memoria del bitmap, el dibujado translúcido es muy lento si dibuja directamente en la memoria de vídeo, así que siempre que sea posibe debería usar un bitmap de memoria. El bitmap y el sprite deben, normalmente, tener la misma profundidad de color, pero como caso especial puede dibujar sprites en formato RGBA de 32 bits en cualquier bitmap hicolor o truecolor, siempre y cuando llame primero a set_alpha_blender(), y puede dibujar imágenes con 8 bits de alfa en un destino en RGBA de 32 bits, mientras llame antes a set_write_alpha_blender().

- **draw_lit_sprite**

void draw_lit_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, int color);

Entinta el sprite hacia el color o nivel de luz especificado, usando la tabla global color_map, y dibuja la imagen resultante en el bitmap destino. Esto sólo se puede usar si antes ha creado una tabla de colores (para modos de 256 colores) o un mapa de fundido (para modos truecolor).

- **draw_gouraud_sprite**

void draw_gouraud_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, int c1, int c2, int c3, int c4);

Entinta el sprite al color o nivel de luz especificado, interpolando los cuatro colores de la esquina sobre la imagen. Esto sólo se puede usar si antes ha creado una tabla de colores (para modos de 256 colores) o un mapa de fundido (para modos truecolor).

- **draw_character**

```
void draw_character(BITMAP *bmp, BITMAP *sprite, int x, int y, int color);
```

Dibuja una copia del sprite en el bitmap destino en la posición especificada, dibujando los pixels transparentes (cero en modos de 256 colores, rosa fucsia en modos truecolor) con el modo de texto actual (ignorándolos si el modo de texto es -1, de otra forma los dibuja en el color de fondo del texto), y ajustando el resto de los pixels al color especificado. El sprite debe ser una imagen de 8 bits, incluso si el destino es un bitmap truecolor.

- **rotate_sprite**

```
void rotate_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, fixed angle);
```

Dibuja el sprite en el bitmap destino. Será colocado usando su esquina superior izquierda como la posición especificada, y entonces será rotado el ángulo especificado sobre su centro. El ángulo es un número de punto fijo 16.16 con el mismo formato usado por las funciones trigonométricas de punto fijo, siendo 256 un círculo completo, 64 un ángulo recto, etc. Todas las funciones de rotación pueden usar diferentes tipos de bitmaps de origen y destino, incluso bitmaps de pantalla o con una profundidad de color distinta.

- **rotate_sprite_v_flip**

```
void rotate_sprite_v_flip(BITMAP *bmp, BITMAP *sprite, int x, int y, fixed angle);
```

Como rotate_sprite, pero también invierte la imágen verticalmente. Para invertir la imagen horizontalmente, use esta rutina pero añada fixtoi(128) al ángulo. Para invertir la imágen en ambos ejes, use rotate_sprite() y añada fixtoi(128) a su ángulo.

- **rotate_scaled_sprite**

```
void rotate_scaled_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, fixed angle, fixed scale);
```

Como rotate_sprite(), pero modifica el tamaño de la imagen a la vez que la rota.

- **rotate_scaled_sprite_v_flip**

void rotate_scaled_sprite_v_flip(BITMAP *bmp, BITMAP *sprite, int x, int y, fixed angle, fixed scale)

Dibuja el sprite, de forma similar a rotate_scaled_sprite(), excepto que primero invierte la imagen verticalmente.

- **pivot_sprite**

void pivot_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, int cx, int cy, fixed angle);

Como rotate_sprite(), pero toma el punto del sprite dado por (cx, cy) como (x, y) en el bitmap, y entonces lo rota sobre este punto.

- **pivot_sprite_v_flip**

void pivot_sprite_v_flip(BITMAP *bmp, BITMAP *sprite, int x, int y, int cx, int cy, fixed angle);

Como rotate_sprite_v_flip(), pero toma el punto del sprite dado por (cx, cy) como (x, y) en el bitmap, y entonces lo rota sobre este punto.

- **pivot_scaled_sprite**

void pivot_scaled_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, int cx, int cy, fixed angle, fixed scale));

Como rotate_scaled_sprite(), pero toma el punto del sprite dado por (cx, cy) como (x, y) en el bitmap, y entonces lo rota y escala sobre este punto.

- **pivot_scaled_sprite_v_flip**

void pivot_scaled_sprite_v_flip(BITMAP *bmp, BITMAP *sprite, int x, int y, fixed angle, fixed scale);

Como rotate_scaled_sprite_v_flip(), pero toma el punto del sprite dado por (cx, cy) como (x, y) en el bitmap, y entonces lo rota y escala sobre este punto.

- **stretch_sprite**

void stretch_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, int w, int h);

Dibuja un sprite en el bitmap en la posición especificada, cambiando el tamaño al ancho y alto especificado. La diferencia entre stretch_sprite() y stretch_blit() es que stretch_sprite() enmascara los pixels transparentes, que son cero en modos de 256 colores, y rosa fucsia en modos truecolor (rojo y azul al máximo, verde a cero).

SPRITES RLE

Ya que los bitmaps pueden ser usados de tantas maneras diferentes, la estructura bitmap es bastante complicada y contiene muchos datos. En muchas ocasiones, sin embargo, se encontrará almacenando imágenes que sólo son copiadas en la pantalla, en vez de pintar en ellas o usarlas como patrones de relleno, etc. Si este es el caso, sería mejor que usase estas imágenes en estructuras RLE_SPRITE o COMPILED_SPRITE en vez de bitmaps normales.

Los sprites RLE almacenan la imagen en un formato simple runlength, donde los pixels cero repetidos son sustituidos por un contador de longitud, y las series de pixels, que no son cero, son precedidos por un contador que da la longitud del recorrido sólido. Los sprites RLE son normalmente más pequeños que los bitmaps, tanto por la compresión run length como porque evitan la mayoría de sobrecarga de la estructura bitmap. También son normalmente más rápidos que bitmaps normales, porque en vez de tener que comparar cada pixel individual con cero para determinar si hay que dibujarlo, es posible saltarse una serie de ceros con una simple suma, o copiar una serie larga de pixels que no son cero con rápidas instrucciones de cadena.

Sin embargo no es oro todo lo que reluce, y hay una falta de flexibilidad con los sprites RLE. No puede dibujar en ellos, y no puede invertirlos, rotarlos o modificar su tamaño. De hecho, lo único que puede hacer con ellos es copiarlos en un bitmap con la función draw_rle_sprite(), que es equivalente a usar draw_sprite() con un bitmap normal. Puede convertir bitmaps en sprites RLE en tiempo real, o puede crear estructuras de sprites RLE en los ficheros de datos con el grabber, creando un nuevo objeto de tipo 'RLE sprite'.

- **get_rle_sprite**

```
RLE_SPRITE *get_rle_sprite(BITMAP *bitmap);
```

Crea un sprite RLE basándose en el bitmap especificado (que debe ser un bitmap de memoria).

- **destroy_rle_sprite**

```
void destroy_rle_sprite(RLE_SPRITE *sprite);
```

Destruye una estructura de sprite RLE previamente creada por get_rle_sprite().

- **draw_rle_sprite**

```
void draw_rle_sprite(BITMAP *bmp, const RLE_SPRITE *sprite, int x, int y);
```

Dibuja un sprite RLE en el bitmap en la posición especificada.

- **draw_trans_rle_sprite**

void draw_trans_rle_sprite(BITMAP *bmp, const RLE_SPRITE *sprite, int x, int y);

Versión translúcida de draw_rle_sprite(). Vea la descripción de draw_trans_sprite(). Sólo debe ser usado después de que haya creado la tabla de color (para modos de 256 colores) o funciones de fundido (para modos truecolor). El bitmap y el sprite deben, normalmente, tener la misma profundidad de color, pero como caso especial puede dibujar sprites en formato RGBA de 32 bits en cualquier bitmap hicolor o truecolor, siempre y cuando llame a set_alpha_blender() antes.

- **draw_lit_rle_sprite**

void draw_lit_rle_sprite(BITMAP *bmp, const RLE_SPRITE *sprite, int x, y, color);

Versión teñida de draw_rle_sprite(). Lea la descripción de draw_lit_sprite(). Esta sólo debe usarse después de que haya construído la tabla de color (para modos de 256 colores) o funciones de fundido (para modos truecolor).

SPRITES COMPILADOS

Los sprites compilados se almacenan como instrucciones de código máquina que dibujan una imagen específica sobre un bitmap, usando instrucciones mov con valores de datos inmediatos. Esto es el método más rápido para dibujar una imagen con máscara: en mi máquina, dibujar sprites compilados es unas cinco veces más rápido que usando draw_sprite() con bitmaps normales. Los sprites compilados son grandes, por lo que si hay poca memoria, debería usar sprites RLE, y lo que puede hacer con ellos esta incluso más restringido que con sprites RLE, porque no soportan áreas de recorte. Si intenta dibujar uno fuera de los bordes de un bitmap, corromperá memoria y probablemente se cargará el sistema. Puede convertir bitmaps en sprites compilados en tiempo real, o puede crear complicadas estructuras de sprites compilados en los ficheros de datos con el grabber creando un nuevo objeto de tipo 'Compiled sprite' o 'Compiled x-sprite'.

- **get_compiled_sprite**

COMPILED_SPRITE *get_compiled_sprite(BITMAP *bitmap, int planar);

Crea un sprite compilado basándose en el bitmap especificado (que debe ser un bitmap de memoria). Los sprites compilados son dependientes de los dispositivos, por lo que debe especificar si compilarlo en un formato lineal o planar. Pase FALSE como segundo parámetro si va a dibujar en bitmaps de

memoria o bitmaps de pantalla de modo 13h y SVGA, y pase TRUE si va a dibujarlos en bitmaps de pantalla modo-X o modo Xtended.

- **destroy_compiled_sprite**

void destroy_compiled_sprite(COMPILED_SPRITE *sprite);

Destruye una estructura de un sprite compilado previamente creado por get_compiled_sprite().

- **draw_compiled_sprite**

void draw_compiled_sprite(BITMAP *bmp, const COMPILED_SPRITE *sprite, int x, int y);

Dibuja un sprite compilado en un bitmap en la posición especificada. El sprite debe ser compilado según el tipo correcto del bitmap (lineal o planar). Esta función no soporta áreas de recorte.

Ayuda: si el no poder recortar sprites compilados es un problema, un buen truco es crear una superficie de trabajo (bitmap de memoria, pantalla virtual en modo-X o lo que sea) un poco más grande de lo que necesite, y usar el centro como su pantalla. De esta forma puede dibujar por afuera de los bordes sin ningún problema.

SALIDA DE TEXTO

Allegro proporciona rutinas de salida de texto que funcionan tanto con fuentes monocromo y color, que pueden contener cualquier número de rangos de carácteres Unicode. El programa grabber puede crear fuentes desde conjuntos de carácteres dibujados en un fichero bitmap (mire grabber.txt para más información), y también puede importar ficheros de fuentes en formato GRX o BIOS. La estructura fuente contiene un número de engaches (hooks) que pueden usarse para extenderla con su propio código de dibujo: vea la definición en allegro.h para conocer los detalles.

- **font**

```
extern FONT *font;
```

Una fuente simple con un tamaño fijo de 8x8 (del modo 13h de la BIOS). Si quiere alterar la fuente usada por las rutinas GUI, cambie esto para que apunte a una de sus fuentes. Esta función contiene los rangos estándar de carácteres ASCII (U+20 a U+7F), Latin-1 (U+A1 a U+FF) y Latin Extended-A (U+0100 a U+017F).

- **allegro_404_char**

```
extern int allegro_404_char;
```

Cuando Allegro no puede encontrar el gráfico que necesita una fuente, en su lugar dibujará el carácter indicado por allegro_404_char. Por defecto éste es el símbolo del acento circumflejo, '^'.

- **text_mode**

```
int text_mode(int mode);
```

Ajusta el modo en el que se dibujará el texto. Devuelve el valor previo. Si el modo es cero o positivo, la salida de texto será opaca y el fondo de los carácteres serán ajustados al color #mode. Si mode es negativo, el texto será dibujado de forma transparente (es decir, el fondo de los carácteres no será alterado). Por defecto este modo es cero.

- **textout**

```
void textout(BITMAP *bmp, const FONT *f, const char *s, int x, y, int color);
```

Escribe la cadena s en el bitmap en la posición x, y, usando el modo de texto actual y el color de fondo y fuente especificado. Si el color es -1 y se esta usando una fuente en color, se dibujará usando los colores de el bitmap de fuentes original (el que importó en el programa grabber), lo que permite salida de texto multicolor.

- **textout_centre**

void textout_centre(BITMAP *bmp, const FONT *f, const unsigned char *s, int x, y, color);

Como textout(), pero interpreta la coordenada x como el centro y no como el margen izquierdo de la cadena.

- **textout_right**

void textout_right(BITMAP *bmp, const FONT *f, const char *s, int x, y, color);

Como textout(), pero interpreta la coordenada x como la parte derecha de la cadena de texto.

- **textout_justify**

void textout_justify(BITMAP *bmp, const FONT *f, const unsigned char *s, int x1, int x2, int y, int diff, int color);

Dibuja texto justificado dentro de la región x1-x2. Si la cantidad de espacio reservado es mayor que el valor diff, se dará por vencido y dibujará un texto justificado por la izquierda.

- **textprintf**

void textprintf(BITMAP *bmp, const FONT *f, int x, y, color, const char *fmt, ...);

Salida de texto formateada, usando un formato de cadena al estilo de printf().

- **textprintf_centre**

void textprintf_centre(BITMAP *bmp, const FONT *f, int x, y, color, const char *fmt, ...);

Como textprintf(), pero interpreta la coordenada x como el centro y no como el margen izquierdo de la cadena.

- **textprintf_right**

void textprintf_right(BITMAP *bmp, FONT *f, int x, y, color, const char *fmt,...);

Como textprintf(), pero interpreta la coordenada x como la parte derecha de la cadena.

- **textprintf_justify**

void textprintf_justify(BITMAP *bmp, const FONT *f, int x1, int x2, int y, int diff, int color, const char *fmt, ...);

Como textout_justify, pero usando una cadena de formato printf.

- **text_length**

int text_length(const FONT *f, const unsigned char *str);

Devuelve la longitud (en pixels) de la cadena en la fuente especificada.

- **text_height**

int text_height(const FONT *f)

Devuelve la altura (en pixels) de la fuente especificada.

- **destroy_font**

void destroy_font(FONT *f);

Libera la memoria usada por una estructura de fuente.

RENDERIZACIÓN DE POLÍGONOS

- **polygon3d**

void polygon3d(BITMAP *bmp, int type, BITMAP *texture, int vc, V3D *vtx[]);
void polygon3d_f(BITMAP *bmp, int type, BITMAP *texture, int vc, V3D_f *vtx[]);

Dibuja polígonos 3d en el bitmap especificado, usando el modo de render especificado. A diferencia de la función polygon(), estas rutinas no soportan figuras cóncavas o con intersecciones, y no pueden dibujar sobre bitmaps de pantalla en modo-X (si quiere escribir código en modo-X, dibuja en un bitmap de memoria y cópialo a la pantalla). El ancho y alto del bitmap de la textura debe ser un múltiplo de dos, pero puede ser diferente, ejemplo: una textura 64x16 está bien, pero una de 17x3 no. El parámetro que cuenta los vértices (vc) debe ser seguido por un array que contiene el número apropiado de punteros a estructuras vertex: polygon3d() usa la estructura de punto fijo V3D, mientras que polygon3d_f() usa la estructura coma flotante V3D_f. Estas son definidas así:

```
typedef struct V3D
{
      fixed x, y, z;          -posición
      fixed u, v;             -coordenadas de la textura
      int c;                  -color
} V3D;

typedef struct V3D_f
{
      float x, y, z;      -posición
      float u, v;         -coordenadas de la textura
      int c;              -color
} V3D_f;
```

El cómo se almacenan los datos de los vértices depende del modo de render: Los valores x e y especifican la posición del vértice en coordenadas de pantalla 2d.

El valor z sólo es necesario cuando use corrección de perspectiva en las texturas, y especifica la profundidad del punto usando coordenadas del mundo

3d.

Las coordenadas u y v sólo son necesarias cuando use texturas, y especifican la posición del punto en el bitmap de la textura que se corresponde con el vértice indicado. El plano de la textura es un plano infinito con el bitmap repetido sobre toda la superficie, y la imagen del polígono resultante en este plano rellenará el polígono cuando se dibuje en pantalla.

Nos referimos a pixels en el plano de la textura como texels. Cada texel es un bloque, nó sólo un punto, y los números enteros de u y v se refieren a la esquina superior izquierda del texel. Esto tiene varias implicaciones. Si quiere dibujar un polígono rectangular y aplicar una textura de 32x32 sobre él, debe usar las coordenadas de textura (0,0), (0,32), (32,32) y (32,0), asumiendo que los vértices son especificados en órden antihorario. La textura será aplicada perfectamente sobre el polígono. No obstante, note que si ajustamos u=32, la última columna de texels que se verán en la pantalla serán los que están en u=31, y lo mismo ocurre para v. Esto es porque las coordenadas se refieren a la esquina superior izquierda de los texels. En efecto, las coordenadas de textura por la derecha y por abajo son exclusivas.

Aquí hay otro punto interesante. Si tiene dos polígonos juntos que comparten dos vértices (como las dos partes de una pieza de cartón doblada), y quiere aplicar sobre ellos una textura contínua, los valores u y v de los vértices que están en la junta serán iguales para ambos polígonos. Por ejemplo, si ambos son rectangulares, un polígono usará (0,0), (0,32), (32,32) y (32,0), y el otro usará (32,0), (32,32), (64,32) y (64,0). Esto aplicará la textura perfectamente.

Por supuesto puede usar números con decimales para u y v indicando puntos que están parcialmente en un texel. Además, dado que el plano de la textura es infinito, puede especificar valores mayores que el tamaño de la textura. Esto puede ser usado para repetir la textura varias veces sobre el polígono.
El valor c especifica el color del vértice, y es interpretado de forma diferente por los modos de render.

El parámetro type especifica el modo de render, y puede ser cualquiera de los siguientes:

POLYTYPE_FLAT:
Un simple polígono con sombreado plano, que toma el color del valor c del primer vértice. Este tipo de polígono es afectado por la función drawing_mode(), por lo que puede ser usado para renderizar polígonos transparentes o XOR.

POLYTYPE_GCOL:
Un polígono con un color de sombreado goraud. Los colores de cada vértice son tomados del valor c, e interpolados a través del polígono. Esto es muy rápido, pero sólo funcionará en modos de 256 colores si su paleta tiene un suave gradiente de colores. En modos truecolor interpreta el color como valor empaquetado en formato directo de hardware producido por la función makecol().

POLYTYPE_GRGB:
Un polígono con sombreado goraud que interpola tripletes RGB en vez de un solo color. En modos de 256 colores usa la tabla global rgb_map para convertir el resultado a color de 8 bits, por lo que sólo puede ser usado después de que haya creado una tabla de mapa de colores. Los colores para cada vértice son tomados del valor c, que es interpretado como un triplete RGB de 24 bits (0xFF0000 es rojo, 0x00FF00 es verde y 0x0000FF es azul).

POLYTYPE_ATEX:
Un polígono con textura afín. Esto dibuja la textura a través del polígono con una simple interpolación 2d lineal, que es rápida pero matemáticamente incorrecta. Puede estar bien si el polígono es pequeño o plano hacia la cámara, pero como no cuenta con la acortación de perspectiva, puede producir extraños artefactos movidos en la textura. Para ver lo que quiero decir, ejecuta test.exe y mire lo que pasa con el test polygon3d() cuando hace un zoom muy cerca del cubo.

POLYTYPE_PTEX:
Un polígono texturizado con corrección de perspectiva. Esto usa el valor z de la estructura del vértice así como las coordenadas u/v, por lo que las texturas se ven correctamente independientemente del ángulo de visualización. Ya que esto envuelve cálculos de división en al bucle interior de la texturización, este modo es mucho más lento que POLYTYPE_ATEX, y usa coma flotante, por lo que será muy lento en cualquier cosa peor que un Pentium (incluso con una FPU, un 486 no es capaz de mezclar división de coma flotante con otras operaciones de enteros tal y como puede hacer un Pentium).

POLYTYPE_ATEX_MASK:
POLYTYPE_PTEX_MASK:
Como POLYTYPE_ATEX and POLYTYPE_PTEX, pero los pixels a cero de la textura son ignorados, permitiendo que la textura sea transparente.

POLYTYPE_ATEX_LIT:
POLYTYPE_PTEX_LIT:
Como POLYTYPE_ATEX y POLYTYPE_PTEX, pero la tabla global color_map (para modos de 256 colores) o la función de fundido (para modos truecolor

no-MMX) es usada para fundir la textura con el nivel de luz tomado del valor c en la estructura del vértice. ¡Esto sólo puede ser usado después de que haya creado una tabla de mapa de color o funciones de fundido!

POLYTYPE_ATEX_MASK_LIT:
POLYTYPE_PTEX_MASK_LIT:
Como POLYTYPE_ATEX_LIT y POLYTYPE_PTEX_LIT, pero los pixels a cero de la textura son ignorados, permitiendo que la textura sea transparente.

POLYTYPE_ATEX_TRANS:
POLYTYPE_PTEX_TRANS:
Renderiza texturas translúcidas. Son aplicables todas las reglas generales de dibujado translúcido. No obstante, estos modos tienen una limitación: sólo funcionan con bitmaps en memoria o con memoria de vídeo lineal (no con video por bancos). Ni si quiera lo intente en estos casos, ya que las funciones no realizan chequeos y su programa morirá horriblemente (o como mínimo dibujará mal las cosas).

POLYTYPE_ATEX_MASK_TRANS:
POLYTYPE_PTEX_MASK_TRANS:
Como POLYTYPE_ATEX_TRANS y POLYTYPE_PTEX_TRANS, pero los pixels a cero de la textura son ignorados.

Si el bit CPU_MMX de la variable global cpu_capabilities está activado, las rutinas GRGB y *LIT truecolor serán optimizadas usando instrucciones MMX. Si el bit CPU_3DNOW está activado, las rutinas truecolor PTEX*LIT tomarán ventaja de la extensión de CPU 3DNow!.

Usar rutinas MMX para *LIT tiene un efecto secundario: normalmente (sin MMX), estas rutinas usan las funciones de fundido y otras funciones de luz, creadas con set_trans_blender() o set_blender_mode(). Las versiones MMX sólo usan el valor RGB que se pasa a set_trans_blender() y hacen la interpolación lineal internamente. Por esta razón, un nuevo conjundo de funciones de fundido que se pasa a set_blender_mode() es ignorado.

- **triangle3d**

```
void triangle3d(BITMAP *bmp, int type, BITMAP *tex, V3D *v1, *v2, *v3);
void triangle3d_f(BITMAP *bmp, int type, BITMAP *tex, V3D_f *v1, *v2, *v3);
```

Dibuja triángulos en 3d, usando las estructuras de vértices de punto fijo o coma flotante. A diferencia de quad3d[_f], las funciones triangle3d[_f] no envuelven polygon3d[_f]. Las funciones triangle3d[_f] usan sus propias rutinas para determinar los grados del gradiente. Por esto, triangle3d[_f](bmp,

type, tex, v1, v2, v3) es más rápido que polygon3d[_f](bmp, type, tex, 3, v[]).

- **quad3d**

void quad3d(BITMAP *bmp, int type, BITMAP *tex, V3D *v1, *v2, *v3, *v4);
void quad3d_f(BITMAP *bmp, int type, BITMAP *tex, V3D_f *v1, *v2, *v3, *v4);

Dibuja cuadriláteros en 3d, usando las estructuras de vértices de punto fijo o coma flotante. Esto es equivalente a llamar polygon3d(bmp, type, tex, 4, v[]); o polygon3d_f(bmp, type, tex, 4, v[]);

- **clip3d_f**

int clip3d_f(int type, float min_z, float max_z, int vc, V3D_f *vtx[], V3D_f *vout[], V3D_f *vtmp[], int out[]);

Recorta el polígono dado en vtx. vc es el número de vértices, el resultado va en vout, y vtmp y out son necesarios para uso interno. Los punteros en vtx, vout y vtmp deben apuntar a estructuras V3D_f válidas. Como en el proceso de recorte pueden aparecer nuevos vértices, el tamaño de vout, vtmp y out debería ser al menos vc * (1.5 ^ n), donde n es el número de planos de corte (5 o 6), y '^' denota "elevado a la". El frustum (volúmen visualizado) está definido por -z<x<z, -z<y<z, 0<min_z<z<max_z. Si max_z<=min_z, el recorte z<max_z no se hace. Como puede ver, el recorte se realiza en el espacio de la cámara, con la perspectiva en mente, por lo que esta rutina debería ser llamada despues de aplicar la matriz de cámara, pero antes de la proyección de perspectiva. La rutina interpolará correctamente u, v, y c en la estructura de vértices. Sin embargo, esto no está previsto para GCOL en profundidades de color high/truecolor.

- **clip3d**

int clip3d(int type, fixed min_z, fixed max_z, int vc, V3D *vtx[], V3D *vout[], V3D *vtmp[], int out[]);

Versión de punto fijo de clip3d_f(). Esta función se debería usar con cuidado, dad la precisión limitada de la aritmética de punto fijo, y las altas posibilidades de errores por redondeo: el código de punto flotante es mejor en la mayoría de situaciones.

- **zbuffered rendering**

Un Z-buffer almacena la profundidad de cada pixel dibujado en una pantalla. Cuando un objeto 3d es renderizado, la profundidad de cada pixel es

comparada con el valor ya almacenado en el Z-buffer: si el pixel es más cercano se dibuja, en caso contrario se ignora.

No hace falta ordenar los polígonos. No obstante, sigue siendo útil ignorar los polígonos que no están de cara a la cámara, ya que así se previene la comparación de muchos polígonos ocultos contra el Z-buffer. El render mediante Zbuffer es el único algoritmo soportado por Allegro que resuelve directamente la intersección entre figuras (mire por ejemplo exzbuf.c). El precio que hay que pagar son unas rutinas más complejas (y más lentas).

Los polígonos con Z-buffer son por diseño una extensión de los estilos de render normales POLYTYPE_*. Sólo hay que hacer una OR entre POLYTYPE y el valor POLYTYPE_ZBUF, y las rutinas normales como polygon3d(), polygon3d_f(), quad3d(), etc, renderizarán polígonos con Z-buffer. Ejemplo:

```
polygon3d(bmp, POLYTYPE_ATEX | POLYTYPE_ZBUF, tex, vc, vtx);
```

Por supuesto, las coordenadas z deben ser válidas sin importar el estilo de render.

El procedimiento de render con Z-buffer parece un render con doble buffer. Debería seguir los siguientes pasos: crear el Z-buffer al comienzo de su programa y hacer que la librería lo use mediante set_zbuffer(). Entonces, por cada frame, borre el Z-buffer y dibuje polígonos con POLYTYPE_* | POLYTYPE_ZBUF para finalmente destruir el Z-buffer al finalizar su programa.
Notas sobre los renders con Z-buffer:

· A diferencia de las renderizaciones normales con POLYTYPE_FLAT, el render con Z-buffer no usa la rutina hline(). Por lo tanto, DRAW_MODE no tiene efecto sobre el resultado.

· Las rutinas *LIT* funcionan del modo tradicional - a través del conjunto de rutinas de blending.

· Todas las rutinas con Z-buffer son mucho más lentas que sin Z-buffer (todas usan el coprocesador para interpolar y comprobar valores 1/z).

- **create_zbuffer**

```
ZBUFFER *create_zbuffer(BITMAP *bmp);
```

Crea el Z-buffer usando el tamaño del BITMAP que esté planeando usar para dibujar sus polígonos. Se pueden definir varios Z-buffers, pero sólo se puede usar uno a la vez, por lo que debe usar set_zbuffer() para elegir cuál será activo.

- **create_sub_zbuffer**

ZBUFFER *create_sub_zbuffer(ZBUFFER *parent, int x, int y, int width, int height);

Crea un sub-z-buffer, es decir, un z-buffer que comparte memoria de dibujado con un z-buffer ya existente, pero posiblemente con un tamaño diferente. Son aplicables las mismas reglas que con los sub-bitmaps: la anchura y altura pueden extenderse fuera de los bordes del z-buffer padre (serán recortados), pero el punto de origen debe estar en una región del padre.

Cuando dibuje con z-buffer en un bitmap, la esquina superior izquierda del bitmap siempre está alineada con la esquina superior izquierda del z-buffer actual. Por lo que esta función es útil principalmente si quiere dibujar en un subbitmap y usar el área correspondiente del z-buffer. En otros casos, ej, si quiere dibujar en un sub-bitmap de la pantalla (y no en otras partes de la pantalla), normalmente querrá crear un z-buffer normal (no un sub-z-buffer) del tamaño de la pantalla. No necesita crear un z-buffer del tamaño de la pantalla virtual y entonces un sub-z-buffer de éste.

- **set_zbuffer**

void set_zbuffer(ZBUFFER *zbuf);

Hace que un Z-buffer sea el activo. Este deberá haber sido creado previamente mediante create_zbuffer().

- **clear_zbuffer**

void clear_zbuffer(ZBUFFER *zbuf, float z);

Escribe z en el Z-buffer (0 significa muy lejos). Esta función debe ser usada para iniciar el Z-buffer antes de cada frame. Realmente, las rutinas de bajo nivel comparan la profundidad del pixel actual con 1/z: por ejemplo, si quiere recortar polígonos más lejanos de 10, debe usar: clear_zbuffer(zbuf, 0.1);

- **destroy_zbuffer**

void destroy_zbuffer(ZBUFFER *zbuf);

Destruye el Z-buffer cuando haya finalizado.

- **render de escenas**

Allegro provee dos métodos simples para quitar caras ocultas:

· Z-buffering - (vea más arriba)

· Algoritmos scan-line - a lo largo de cada línea de su pantalla, mantiene información sobre el polígono "en el que está" y cuál es el más cercano. Este estado cambia sólamente cuando la línea cruza el borde de un polígono. Por lo que debe mantener una lista de bordes y polígonos. Y debe ordenar los bordes para cada scanline (esto es contrarrestado manteniendo el órden del scanline previo - no cambiará mucho). La GRAN ventaja es que sólo dibuja cada pixel una vez. Si tiene muchos polígonos que se solapan, puede obtener increíbles velocidades comparadas con cualquiera de los algoritmos previos. Este algoritmo es cubierto por las rutinas *_scene.

El render de escenas sigue los siguientes pasos aproximádamente:

· Iniciar la escena (ajustar el área de recorte, limpiar el bitmap, dibujar un fondo, etc).

· Llame clear_scene().

· Transformar todos sus puntos al espacio de la cámara.

· Recortar polígonos.

· Proyectar con persp_project() o persp_project_f().

· "Dibujar" polígonos con scene_polygon3d() y/o scene_polygon3d_f(). Esto realmente no dibuja nada, sólo inicializa las táblas.

· Renderizar todos los polígonos definidos previamente en el bitmap con render_scene().

· Dibujar gráficos no tridimensionales.

· Mostrar el bitmap (blit a la pantalla, cambio de página, etc).

Por cada linea horizontal del área de visualización se usa una lista ordenada de bordes para saber qué polígonos "están dentro" y cuáles están cerca. Se usa coherencia vertical - la lista de bordes es ordenada a partir de la anterior - no cambiará mucho. Las rutinas de render de escenas usan las mismas rutinas de bajo nivel que polygon3d().

Notas del render de escena:

· A diferencia de polygon3d(), scene_polygon3d() requiere coordenadas z válidas para todos los vértices, indiferentemente del estilo de render (en contraste con polygon3d(), que sólo usa coordenadas z para *PTEX*).

· Todos los polígonos pasados a scene_polygon3d() deben ser proyectados con perspectiva.

· Tras usar render_scene(), el modo es ajustado a SOLID.

Usar muchos polígonos con máscara reduce el rendimiento, porque cuando se encuentra un polígono con máscara en primera linea de visión, los polígonos que están detrás deben ser dibujados también. Lo mismo es aplicable a polígonos FLAT dibujados con DRAW_MODE_TRANS.

El render con Z-buffer también funciona con el render de escenas. Puede ser útil si tiene algunos polígonos que se interseccionan, pero la mayoría de los polígonos pueden ser renderizados sin problemas usando el algoritmo normal de ordenado de scanlines. Igual que antes: simplemente haga una OR del POLYTIPE con POLYTYPE_ZBUF. Además, tiene que limpiar el z-buffer al comienzo del frame. Ejemplo: clear_scene(buffer);if (some_polys_are_zbuf) clear_zbuffer(0.);while (polygons) { ... if (this_poly_is_zbuf) type |= POLYTYPE_ZBUF; scene_polygon3d(type, tex, vc, vtx);}render_scene();

- **create_scene**

int create_scene(int nedge, int npoly);

Reserva memoria para una escena, nedge y npoly son sus estimaciones de cuántas aristas y polígonos renderizará (no puede salirse del límite que especifica aquí). Si usa los mismos valores en llamadas sucesivas, el espacio será reusado (no nuevos malloc()).

La memoria reservada es algo menor que 150 * (nedge + npoly) bytes. Devuelve cero con éxito, o negativo si no se pudo reservar la memoria.

- **clear_scene**

void clear_scene(BITMAP *bmp);

Inicializa la escena. El bitmap es donde renderizará sus gráficos.

- **destroy_scene**

void destroy_scene();

Libera la memoria previamente reservada por create_scene.

- **scene_polygon3d**

int scene_polygon3d(int type, BITMAP *texture, int vc, V3D *vtx[]); int scene_polygon3d_f(int type, BITMAP *texture, int vc, V3D_f *vtx[]);

Pone un polígono en la lista de render. Realmente no renderiza nada en este momento. Debe llamar esta función entre clear_scene() y render_scene().

Los argumentos son iguales que para polygon3d(), excepto por el parámetro de bitmap que falta. Se usará el que indicó mediante clear_scene().

A diferencia de polygon3d(), los polígonos pueden ser cóncavos o estar interseccionados. Las figuras que penetran en otras pueden salir bien, pero no son manejadas realmente por este código.

Note que sólo se almacena un puntero a la textura, por lo que debería mantenerla en memoria hasta render_scene(), donde será usada.

Ya que el estilo FLAT es implementado con la función de bajo nivel hline(), el estilo FLAT está sujeto a DRAW_MODE. Todos los modos son válidos. Junto con el polígono, el modo será almacenado para el momento del render, y también otras variables relacionadas (puntero al mapa de color, puntero al patron, ancla, valores de blending).

El valor de los bits CPU_MMX y CPU_3DNOW de la variable global cpu_capabilities afectará la elección de la rutina de bajo nivel en ensamblador que será usada por render_scene() con este polígono.

Devuelve cero con éxito o negativo si no será renderizado debido a que falta la rutina de render apropiada.

- **render_scene**

void render_scene();

Renderiza toda la escena creada con scene_polygon3d() en el bitmap que pasó a clear_scene(). El render se realiza una linea a cada vez, sin procesar dos veces el mismo pixel.

Note que entre clear_scene() y render_scene() no debería modificar el rectángulo de recorte del bitmap destino. Por razones de velocidad, debería ajustar el rectángulo de recorte al mínimo.

Tenga en cuenta también que las texturas pasadas a scene_polygon3d() son almacenadas como punteros y serán usadas en render_scene().

- **scene_gap**

extern float scene_gap;

Este número (valor por defecto = 100.0) controla el comportamiento del algoritmo de ordenado en z. Cuando un borde está muy cerca del plano de otro polígono, hay un intervalo de incertidumbre en el cual no se puede determinar qué objeto es visible (qué z es más pequeña). Esto es debido a errores numéricos acumulativos para los bordes que han sufrido bastantes transformaciones e interpolaciones.

El valor por defecto significa que si los valores 1/z (en espacio proyectado) difieren sólo en 1/100 (uno por ciento), serán considerados iguales y el eje x de los planos será usado para saber qué plano está acercándose mientras nos movemos hacia la derecha.

Valores mayores significan márgenes menores, e incrementan la posibilidad de confundir planos/bordes realmente adyacentes. Valores menores significan márgenes mayores, e incrementan la posibilidad de confundir un polígono cercano con uno adyacejte. El valor de 100 está cercano a lo más óptimo. No obstante, el valor optimo oscila con diferentes resoluciones, y puede ser dependiente de la aplicación. Está aquí para que lo pueda ajustar al máximo.

TRANSPARENCIAS Y DIBUJO CON PATRÓN

- **drawing_mode**

void drawing_mode(int mode, BITMAP *pattern, int x_anchor, int y_anchor);

Ajusta el modo de dibujo gráfico. Esto sólo afecta a las rutinas geométricas como putpixel, lines, rectangles, circles, polygons, floodfill, etc, y no a la salida de texto, blits o dibujado de sprites. El modo debería ser uno de los siguientes valores.

DRAW_MODE_SOLID	-por defecto, dibujado sólido
DRAW_MODE_XOR	-dibujado orexclusivo
DRAW_MODE_COPY_PATTERN	-rellenado con patrón multicolor
DRAW_MODE_SOLID_PATTERN	-rellenado con patrón de un solo color
DRAW_MODE_MASKED_PATTERN	-rellenado con patrón enmascarado
DRAW_MODE_TRANS	-fundido de color translúcido

En DRAW_MODE_XOR, los pixels son escritos en el bitmap con una operación or-exclusiva en vez de con la copia simple, por lo que dibujar la misma figura dos veces la borrará. Como esto requiere tanto leer como escribir en el bitmap de memoria, el dibujado xor es mucho más lento que el modo normal.

Con los modos con patrón, usted indica un bitmap de patrón que será dibujado sobre la superficie de la figura. Allegro almacena un puntero a este bitmap en vez de una copia, por lo que no debe destruir el bitmap mientras sea usado como patrón. El ancho y alto del patrón debe ser un múltiplo de dos, pero pueden ser diferentes, es decir, un patrón de 64x16 está bien, pero uno de 17x3 no. El patrón será repetido en una rejilla comenzando en el punto (x_anchor, y_anchor). Normalmente debería pasar cero para estos valores, lo que le dejará dibujar varias figuras y que sus patrones se junten en los bordes. Un alineamiento de cero puede sin embargo ser peculiar cuando mueva una figura con patrón por la pantalla, porque la figura se moverá, pero el patrón no, por lo que en algunas situaciones quizás le interese alterar las posiciones del ancla (anchor).

Cuando selecciona DRAW_MODE_COPY_PATTERN, los pixels simplemente son copiados del bitmap de patrón al bitmap destino. Esto le permite usar patrones multicolor, y significa que el color que pase a la rutina de dibujado es ignorado. Este es el más rápido de los modos con patrón.

En DRAW_MODE_SOLID_PATTERN, cada pixel del patrón es comparado con el color de máscara (cero en modos de 256 colores, rosa fucsia en modos truecolor). Si el pixel del patrón es sólido, un pixel del color que pasó a la rutina de dibujado es escrito en el bitmap destino, de otro modo escribe un cero. El patrón es por esto tratado como una máscara monocroma, que le permite usar el mismo patrón para dibujar diferentes figuras con colores diferentes, pero previene el uso de patrones multicolores.

DRAW_MODE_MASKED_PATTERN es casi lo mismo que DRAW_MODE_SOLID_PATTERN, pero los pixels enmascarados son ignorados en vez de copiados como cero, por lo que el fondo se verá a través de los agujeros.

En DRAW_MODE_TRANS, la tabla global color_map o las funciones de fundido se usan para sobreimprimir pixels encima de la imágen existente. Esta sólo debe usarse después de haber constuído la tabla de mapeo de color (para modos de 256 colores) o las funciones de fundido (para modos truecolor). Dado que debe leer y escribir en la memoria del bitmap, el dibujado transparente es muy lento si dibuja directamente en la memora de vídeo, así que siempre que sea posible debería dibujar en bitmaps de memoría.

- **xor_mode**

```
void xor_mode(int on);
```

Esto es un atajo para activar o desactivar el modo de dibujado xor. Llamar xor_mode(TRUE) es equivalente a drawing_mode (DRAW_MODE_XOR, NULL, 0, 0); Llamar xor_mode(FALSE) es equivalente a drawing_mode (DRAW_MODE_SOLID, NULL, 0, 0);

- **solid_mode**

```
void solid_mode();
```

Esto es un atajo para seleccionar el dibujado sólido. Es equivalente a llamar drawing_mode(DRAW_MODE_SOLID, NULL, 0, 0);

- **transparencia en modos de 256 colores**

En modos de vídeo con paleta, la translucidez y la iluminación son implementadas con una tabla precalculada de 64k, que contiene el resultado de la combinación de cualquier color c1 con c2. Tiene que crear esta tabla antes de usar cualquiera de las rutinas de iluminación o translucidez. Dependiendo de cómo se crea tabla, será posible hacer un rango diferente de efectos. Por ejemplo, la translucidez se puede implementar usando un color

intermedio entre c1 y c2 como resultado de su combinación. La iluminación se consigue tratando uno de los colores como nivel de luz (0-255) en vez de como color, y creando la tabla apropiadamente. Un rango de efectos especializados es posible, si por ejemplo sustituye cualquier color con otro color haciendo los colores individuales de origen o destino totalmente sólidos o invisibles.

Las tablas de color pueden ser precalculadas con la utilidad colormap, o generadas en tiempo real. La estructura COLOR_MAP se define así:

```
typedef struct
{
      unsigned char data[PAL_SIZE][PAL_SIZE];
} COLOR_MAP;
```

- **color_map**

```
extern COLOR_MAP *color_map;
```

Puntero global a una tabla de color. ¡Esto debe ser ajustado antes de usar cualquiera de las funciones de dibujado translúcido o iluminado en modos de 256 colores!

- **create_light_table**

```
void create_light_table(COLOR_MAP *table, const PALETTE pal, int r, g, b, void (*callback)(int pos));
```

Llena la tabla de mapeo de color especificada con los datos precalculados necesarios para hacer efectos de translucidez con la paleta especificada. Cuando se combinan los colores c1 y c2 con esta tabla, c1 se trata como un nivel de luz desde 0 a 255. Con un nivel de luz de 255 la tabla devolverá el color c2 sin cambios, con un nivel de luz 0 devolverá el valor r,g,b que especifque a la función, y con niveles de luz intermedios devolverá un color intermedio. Los valores r,g y b están entre 0-63. Si la función callback no es NULL, se la llamará 256 veces durante el cálculo, permitiéndole mostrar un indicador de progreso.

- **create_trans_table**

```
void create_trans_table(COLOR_MAP *table, const PALETTE pal, int r, g, b, void (*callback)(int pos));
```

Rellena la tabla de color especificada con los datos precalculados necesarios para hacer efectos de translucidez con la paleta especificada. Cuando se combinan los colores c1 y c2 en esta tabla, el resultado será un color intermedio entre los dos. Los valores r, g, b que especifique son la solidez de

cada componente de color, desde 0 (totalmente transparente) hasta 255 (totalmente sólido). Para una solidez del 50%, pasa 128. Esta función trata el color origen #0 como un caso especial, dejando el destino sin cambiar siempre que se encuentre un pixel del color cero, para que los sprites con máscara puedan ser dibujados correctamente. Si la función callback no es NULL, será llamada 256 veces durante el cálculo, permitiéndole enseñar un indicador de progreso.

- **create_color_table**

void create_color_table(COLOR_MAP *table, const PALETTE pal, void (*blend)(PALETTE pal, int x, int y, RGB *rgb), void (*callback)(int pos));

Llena la tabla de mapeo de color con datos precalculados necesarios para poder hacer efectos propios con la paelta especificada, llamando a la función de fundido para determinar los resultados de cada combinación de color. A su rutina de fundido se le pasará un puntero a la paleta y los dos colores que van a ser combinados, y debería devolver el resultado deseado en una estructura RGB con formato 0-63. Entonces Allegro buscará en la paleta aquél color que mejor encaje con el que pidió, por lo que no importa si la paleta no tiene un color que encaje exactamente. Si la función callback no es NULL, se le llamará 256 veces durante el cálculo, permitiéndole que muestre un indicador de progreso.

- **create_blender_table**

void create_blender_table(COLOR_MAP *table, const PALETTE pal, void (*callback)(int pos));

Llena la tabla de mapeo de color especificada con datos precalculados para hacer un equivalente "paletizado" de cualquiera de los modos de fundido truecolor que esté actualmente seleccionado. Después de llamar a set_trans_blender(), set_blender_mode() o cualquiera de las otras rutinas de modo de fundido, puede usar esta función para crear un tabla de mapeo de 8 bits que tendrá los mismos resultados que el modo de fundido de 24 bits que tenga seleccionado.

- **transparencia en truecolor**

En los modo de video truecolor, la translucidez y la iluminación están implementadas por una función de fundido de la forma:

unsigned long (*BLENDER_FUNC)(unsigned long x, y, n);

Esta rutina toma dos colores como parámetros, los descompone en sus componenetes rojo, verde y azul, los combina acorde con el valor de

interpolación n, y entonces fusiona de nuevo el resultado en un solo valor de color, que devuelve.

Como estas rutinas se pueden usar desde diferentes profundidades de color, hay tres callbacks, una para usar con píxels de 15 bits (5.5.5), una para píxels de 16 bits (5.6.5), y otra para píxels de 24 bits (8.8.8), que puede compartirse entre el código de 24 y 32 bits dado que el empaquetamiento de bits es el mismo.

- **set_trans_blender**

void set_trans_blender(int r, int g, int b, int a);

Selecciona el conjunto de rutinas de fundido por defecto, que hacen una interpolación lineal simple entre los colores fuente y destino. Cuando se llama a una función de dibujo translúcido, el parámetro alfa ajustado por esta rutina se usa como factor de interpolación, que controla la solidez del dibujado (de 0 a 255). Cuando una función de dibujo iluminado de es llamada, el valor alfa se ignora, y se usa el color pasado a la función de sprite para seleccionar un nivel alfa. La rutina de fundido se usará para interpolar entre el color del sprite y los valores RGB que se le pasaron a esta función (en un rango de 0 a 255).

- **set_alpha_blender**

void set_alpha_blender();

Activa el modo de fundido especial de canal-alfa, que se usa para dibujar sprites RGBA de 32 bits. Después de llamar a esta función, puede usar draw_trans_sprite() o draw_trans_rle_sprite() para dibujar una imagen de 32 bits en un otra hicolor o truecolor. Los valore alfa se tomarán directamente del gráfico origen, asíq ue puede variar la solidez de cada parte de la imagen. Sin embargo, no puede usar ninguna de las funciones normales de translucidez mientras este modo esté activo, así que debería volver a uno de los modos normales de fundido (p.ej. set_trans_blender()) antes de dibujar otra cosa que no sean sprites en RGBA de 32 bits.

- **set_write_alpha_blender**

void set_write_alpha_blender();

Activa el modo especial de edición de canal-alfa, que se usa para dibujar canales alfa encima de un sprite RGB existente, para transformarlo en una imagen en formato RGBA. Después de llamar a esta función, puede ajustar el modo de dibujo a DRAW_MODE_TRANS y entonces escribir valores de color (de 0 a 255) en una imágen de 32 bits. Esto dejará los valores de color igual, pero alterará el alfa con los valores que esté escribiendo. Después de activar

este modo también puede usar draw_trans_sprite() para superponer una mascara alfa de 8 bits encima de un sprite existente de 32 bits.

- **set_add_blender**

void set_add_blender(int r, int g, int b, int a);

Activa un modo de fundido de color para combinar píxels truecolor iluminados o translúcidos.

- **set_burn_blender**

void set_burn_blender(int r, int g, int b, int a);

Activa un modo de fundido "chamuscado" para combinar píxels truecolor iluminados o translúcidos. Aquí el brillo de los colores de la imagen origen reduce el brillo de la imagen destino, oscureciendola.

- **set_color_blender**

void set_color_blender(int r, int g, int b, int a);

Activa un modo de fundido de color para combinar píxels truecolor iluminados o translúcidos. Aplica sólo el tono y saturación de la imagen origen a la imagen destino. La luminosidad de la imagen destino no queda afectada.

- **set_difference_blender**

void set_difference_blender(int r, int g, int b, int a);

Activa el modo de fundido por diferencia para combinar píxels truecolor translúcidos o iluminados. Esto crea una imágen que tiene colores calculados por la diferencia entre los colores fuente y destino.

- **set_dissolve_blender**

void set_dissolve_blender(int r, int g, int b, int a);

Activa un modo de fundido por disolución para combinar píxels truecolor translúcidos o iluminados. Aleatoriamente, reemplaza los colores de algunos píxels de la imagen destino con los correspondientes de la imágen origen. El número de pixels reeemplazados depende del valor alpha (cuanto más alto sea el valor, más pixels se reemplazarán; va pillando la idea :)

- **set_dodge_blender**

void set_dodge_blender(int r, int g, int b, int a);

Activa un mode de fundido "trucado" para combinar píxels truecolor iluminados o translúcidos. La claridad de colores en la imágen origen ilumina

los colores de la imagen destino. El mayor efecto se consigue con el blanco; el negro no tiene ningún efecto.

- **set_hue_blender**

void set_hue_blender(int r, int g, int b, int a);

Activa un modo de fundido por tono para combinar píxels truecolor iluminados o translúcidos. Éste aplica el tono del origen al destino.

- **set_invert_blender**

void set_invert_blender(int r, int g, int b, int a);

Activa el modo de fundido inverso para combinar píxels truecolor iluminados o translúcidos. Funde el color inverso (o negativo) del origen con el de destino.

- **set_luminance_blender**

void set_luminance_blender(int r, int g, int b, int a);

Activa el modo de fundido de iluminación para combinar píxels truecolor iluminados o translúcidos. Aplica la ilumincación del origen al destino. El color de la destinación no queda afectado.

- **set_multiply_blender**

void set_multiply_blender(int r, int g, int b, int a);

Activa un modo de fundido por multiplicación para combinar píxels truecolor iluminados o translúcidoz. Combina las imágenes origen y destino, multiplicando los coloroes para producir un color más oscuro. Si el color se multiplica por blanco no cambia; cuando se multiplica por negro se vuelve negro.

- **set_saturation_blender**

void set_saturation_blender(int r, int g, int b, int a);

Activa un modo de fundido de saturación para combinar píxels truecolor iluminados o translúcidos. Aplica la saturación del origen a la imagen destino.

- **set_screen_blender**

void set_screen_blender(int r, int g, int b, int a);

Activa un modo de fundido de pantalla para combinar píxels truecolor iluminados o translúcidos. Este modo de fundido ilumina el color de la imagen destino multiplicando el color inverso del origen con el color destino. Es más o menos el opuesto al modo de fundido por multiplicación.

- **set_blender_mode**

void set_blender_mode(BLENDER_FUNC b15, b16, b24, int r, g, b, a);

Especifíca un conjunto de rutinas propias de fundido en truecolor, que pueden ser usadas para implementar cualquier modo de interpolación que necesite. Esta función comparte un único fundido entre los modos de 24 y 32 bits.

- **set_blender_mode_ex**

void set_blender_mode_ex(BLENDER_FUNC b15, b16, b24, b32, b15x, b16x, b24x, int r, g, b, a);

Como set_blender_mode(), pero le permite especificar un conjunto más completo de rutinas de fundido. Las rutinas b15, b16, b24 y b32 se usan cuando se vayan a dibujar píxels en destinos con el mismo formato, meintras que b15x, b16x y b24x las usan draw_trans_sprite() y draw_trans_rle_sprite cuando van adibujar imágenes RGBA en un destino que tiene otro formato. Estos fundidos se les pasará un parámetro x de 32 bits, junto con un valor y de una profundidad diferente de color, y debe hacer algo sensible como respuesta.

CONVERSIONES DE FORMATOS DE COLOR

En general, Allegro está diseñada para usarse sólo con una profundidad de color a la vez, así que sólo llamará una vez a set_color_depth() y entonces almacenará todos sus bitmaps en el mismo formato. Si quiere mezclar diferentes formatos de píxel, puede usar create_bitmap_ex() en vez de create_bitmap(), y llamar a bitmap_color_depth() para pedir el formato de la imagen en cuestión. La mayoría de las rútinas gráficas requieren que todos sus parámetros de entrada estén en el mismo formato (p.ej. no puede expandir un bitmap origen de 15 bits en un destino de 24 bits), pero hay cuatro excepciones: blit() y las rutinas de rotación puede copiar entren bitmaps de cualquier formato, convirtiendo los datos cuando se requiera, draw_sprite() puede dibujar imágenes origen de 256 colores en destion de cualquier formato, draw_character() _siempre_ usa un bitmap origen de 256 colores, sea cual sea el formato de destino, las funciones draw_trans_sprite() y draw_trans_rle_sprite() son capaces de dibujar imágenes de 32 bits RGBA en un destino hicolor o truecolor, siempre y cuando llame antes a set_alpha_blender(), y la función draw_trans_sprite() es capaz de dibujar una imagen con 8 bits de canal alfa encima de una imagen ya existente de 32 bits, siempre y cuando llame antes a set_write_alpha_blender().

Expandir un bitmap origen de 256 colores en un bitmap destino truecolor es muy rápido (iobviamente deberá ajustar la paleta de colores correcta antes de la conversión!). Convertir entre diferentes formatos truecolor es algo más lento, y reducir imágenes truecolor a 256 puede ser muy lento (puede acelerarse si crea una tabla global rgb_map antes de realizar la conversión).

- **bestfit_color**

```
int bestfit_color(const PALLETE pal, int r, int g, int b);
```

Busca en la paleta el valor más parecido al color requerido, que es especificado en el formato hardware de la VGA 0-63. Normalmente debería llamar makecol8() en vez de esto, pero esta función de bajo nivel puede ser útil si necesita usar una paleta de colores diferente a la seleccionada, o si no quiere usar la tabla precalculada rgb_map a propósito.

- **rgb_map**

```
extern RGB_MAP *rgb_map;
```

Para acelerar la reducción de valores RGB a 8 bits, Allegro usa una tabla precalculada de 32k (5 bits por cada componente de color). Debe crear esta tabla antes de usar las rutinas de sombreado gouraud, y si está presente, la tabla acelerará vastamente la función makecol8(). Las tablas RGB pueden ser precalculadas con la utilidad rgbmap, o generadas en tiempo real. La estructura RGB_MAP está definida como:

```
typedef struct
{
      unsigned char data[32][32][32];
} RGB_MAP;
```

- **create_rgb_table**

void create_rgb_table(RGB_MAP *table, const PALETTE pal, void (*callback)(int pos));

Llena la tabla RGB especificada con datos precalculados de la paleta elegida. Si la función callback no es NULL, será llamada 256 veces durante el cálculo, permitiéndole enseñar un indicador de progreso.

- **hsv_to_rgb**

void hsv_to_rgb(float h, float s, float v, int *r, int *g, int *b); void rgb_to_hsv(int r, int g, int b, float *h, float *s, float *v);

Convierte valores de color entre los modos HSV y RGB. Los valores RGB van de 0 a 255, hue va de 0 a 360, y la saturación y el valor van de 0 a 1.

ACCESO DIRECTO A LA MEMORIA DE VÍDEO

La estructura bitmap es así:

```
typedef struct BITMAP
{
      int w, h;                 -tamaño del bitmap en pixels
      int clip;                 -no-cero si recortar está activado
      int cl, cr, ct, cb;       -límites de recorte izquierdo, derecho, superior e
                                inferior
      int seg;                  -segmento para uso con los punteros a línea
      unsigned char *line[];    -punteros al comienzo de cada línea
} BITMAP;
```

También hay otras cosas en la estructura, pero podrían cambiar, y no debería usar nada excepto lo de arriba. El rectángulo de recorte es inclusivo arriba a la izquierda (0 permite dibujar en la posición 0) pero exclusivo abajo a la derecha (10 permite dibujar en la posición 9, pero no en la 10). Fíjese que éste no es el mismo formato que el que se usa con set_clip(), el cual toma coordenadas inclusivas para las cuatro esquinas.

Hay varios modos de conseguir acceso directo a la memoria de imagen de un bitmap, variando en complejidad dependiendo del tipo de bitmap que use.

El modo más simple sólo servirá si trabaja con bitmaps de memoria (obtenidos con create_bitmap(), ficheros de datos, y ficheros de imágenes) y sub-bitmaps de bitmaps de memoria. Esto usa una tabla de punteros char, llamados 'line', la cual es parte de la estructura bitmap y contiene punteros al comienzo de cada línea de la imagen. Por ejemplo, una función putpixel simple para un bitmap de memoria es:

```
void memory_putpixel(BITMAP *bmp, int x, int y, int color)
{
      bmp->line[y][x] = color;
}
```

Para modos truecolor, es necesario especificar el tipo del puntero de línea, por ejemplo:

```
void memory_putpixel_15_or_16_bpp(BITMAP *bmp, int x, int y, int color)
```

```
{
	((short *)bmp->line[y])[x] = color;
}

void memory_putpixel_32(BITMAP *bmp, int x, int y, int color)
{
	((long *)bmp->line[y])[x] = color;
}
```

Si quiere escribir en la pantalla y también en bitmaps de memoria, necesita usar algunas macros auxiliares, porque la memoria de vídeo puede no ser parte de su espacio de direcciones normal. Esta simple rutina funcionará para cualquier pantalla lineal, p.ej unos framebuffers lineales de VESA.

```
void linear_screen_putpixel(BITMAP *bmp, int x, int y, int color)
{
	bmp_select(bmp);
	bmp_write8((unsigned long)bmp>line[y]+x, color);
}
```

Para los modos truecolor debería reemplazar bmp_write8() por bmp_write16(), bmp_write24(), o bmp_write32(), y multiplicar el desplazamiento x por el número de bytes por píxel. Por supuesto hay funciones similares para leer el valor de un pixel de un bitmap, y son bmp_read8(), bmp_read16(), bmp_read24() y bmp_read32().

Esto, sin embargo, seguirá sin funcionar en los modos de SVGA por bancos, o en plataformas como Windows, que hacen un procesado especial dentro de las funciones de cambio de banco. Para un acceso más flexible a los bitmaps de memoria, necesita llamar a las rutinas:

```
unsigned long bmp_write_line(BITMAP *bmp, int line);
```

Selecciona la línea de un bitmap en la que va a dibujar.

```
unsigned long bmp_read_line(BITMAP *bmp, int line);
```

Selecciona la línea de un bitmap de la que va a leer.

```
unsigned long bmp_unwrite_line(BITMAP *bmp);
```

Libera el bitmap de memoria despued de que haya acabado de trabajar con él. Sólo necesita llamar a esta función una vez al final de una operación de

dibujado, incluso si ha llamado a bmp_write_line() o bmp_read_line() diversas beces antes.

Estas están implementadas como rutinas de ensamblador en línea, por lo que no son tan ineficientes como podrían parecer. Si el bitmap no necesita cambio de banco (ejemplo: es un bitmap de memoria, pantalla en modo 13h, etc), estas funciones simplemente retornan bmp->line[line].

A pesar de que los bitmaps de SVGA usan bancos, Allegro permite acceso lineal a la memoria dentro de cada scanline, por lo que sólo necesita pasar una coordenada y a estas funciones. Varias posiciones x pueden ser obtenidas simplemente sumando la coordenada x a la dirección devuelta. El valor devuelto es un entero largo sin signo en vez de un puntero a caracter porque la memoria bitmap puede no estar en su segmento de datos, y necesita acceder a él con punteros far. Por ejemplo, una función putpixel usando funciones de cambio de banco es:

```
void banked_putpixel(BITMAP *bmp, int x, int y, int color)
{
      unsigned long address = bmp_write_line(bmp, y);
      bmp_select(bmp);
      bmp_write8(address+x, color); bmp_unwrite_line(bmp);
}
```

Se dará cuenta de que Allegro tiene funciones separadas para seleccionar los bancos de lectura y escritura. Es importante que distinga entre estos, porque en algunas tarjetas de vídeo los bancos pueden ser seleccionados individualmente, y en otros la memoria de vídeo es leída y escrita en direcciones diferentes. No obstante, la vida nunca es tan simple como desearíamos que fuese (esto es verdad incluso cuando _no_ estamos hablando de programación de gráficos :-) y por supuesto algunas tarjetas de vídeo sólo pueden proveer un banco. En éstas, las funciones de lectura/escritura se comportarán idénticamente, por lo que no debería asumir que puede leer de una parte de la memoria de vídeo y escribir en otra al mismo tiempo. Puede llamar bmp_read_line(), y leer lo que quiera de la línea, entonces llamar bmp_write_line() con el mismo o diferente valor de línea, y escribir lo que quiera en ella, pero no debería llamar bmp_read_line() y bmp_write_line() a la vez y esperar poder leer una línea y escribir en otra simultáneamente. Sería bueno si esto fuese posible, pero si lo hace, su código no funcionará en tarjetas SVGA de un banco.

Y también está el modo-X. Si nunca antes había programado gráficos en este modo, probablemente no entienda esto, pero para aquellos que quieren saber cómo Allegro trabaja con los bitmaps de pantalla del modo-X, aquí va...

Los punteros a línea todavía están presentes, y contienen direcciones lineales, esto es, la dirección con la cual accedes al primer pixel de la línea. Está garantizada la alineación cada cuatro pixels de las direcciones, por lo que puede fijar el plano de escritura, dividir su coordenada entre cuatro, y añadirla al puntero de línea. Por ejemplo, un putpixel en modo-X es:

```
void modex_putpixel(BITMAP *b, int x, int y, int color)
{
	outportw(0x3C4, (0x100<<(x&3))|2);
	bmp_select(bmp);
	bmp_write8((unsigned long)bmp->line[y]+(x>>2), color);
}
```

Ah sí: el truco de djgpp del nearptr. Personalmente, no me gusta demasiado porque desactiva la protección de la memoria y no es portable a otras plataformas, pero hay mucha gente que suspira por él porque puede dar acceso directo a la memoria de pantalla via un puntero normal de C. ¡Aviso: Este método sólo funcionará con la librería djgpp, cuando esté usando el modo VGA 13h o un framebuffer lineal!

En su código de inicialización:

```
#include <sys/nearptr.h>
unsigned char *screenmemory;
unsigned long screen_base_addr;
__djgpp_nearptr_enable();
__dpmi_get_segment_ base_address(screen->seg, &screen_base_addr);
screenmemory = (unsigned char *)(screen_base_addr + screen->line[0] -
__djgpp_base_address);
```

Luego:

```
void nearptr_putpixel(int x, int y, int color)
{
	screenmemory[x + y*SCREEN_W] = color;
}
```

RUTINAS FLIC

Hay dos funciones de alto nivel para reproducir animaciones FLI/FLC: play_fli(), la cual lee datos directamente del

disco, y play_memory_fli(), la cual usa datos que ya están cargados en la RAM. Aparte de las diferentes fuentes de las que se sacan los datos, estas dos funciones se comportan de forma idéntica. Ambas dibujan la animación en el

bitmap especificado, que debería ser normalmente screen. Los cuadros se alinearán con la esquina superior izquierda del bitmap: si quiere posicionarlos en otra parte de la pantalla tendrá que crear un sub-bitmap y decirle al reproductor FLI que dibuje allí la animación. Si loop está activado, el reproductor volverá al principio de la animación cuando ésta acabe, de otro modo, reproducirá la animación una vez. Si la función callback no es NULL, será llamada por cada frame, permitiéndole realizar tareas en segundo plano. La rutina de callback debe devolver cero: si retorna un valor que no es cero, el reproductor se parará (este es el único modo de parar una animación que esté siendo repetida). El reproductor FLI devuelve FLI_OK si ha llegado al final del fichero, FLI_ERROR si hubo problemas, y el valor de la función callback, si fue ésta la que paró la reproducción. Si necesita distinguir entre diferentes valores de retorno, su rutina de callback debería devolver enteros positivos, ya que FLI_OK es cero y FLI_ERROR negativo. Tome nota de que el reproductor FLI sólo funcionará si se instaló el módulo de temporización, y que alterará la paleta de colores según la del fichero de animación.

Ocasionalmente podría necesitar un control más detallado sobre la reproducción FLI, por ejemplo si quiere sobreimprimir algún texto sobre la animación, o reproducirla a una velocidad diferente. Puede hacer ambas cosas usando las funciones de bajo nivel descritas abajo.

- **play_fli**

int play_fli(const char *filename, BITMAP *bmp, int loop, int (*callback)());

Reproduce un fichero FLI o FLC del Autodesk Animator, leyendo los datos del disco según se necesiten.

- **play_memory_fli**

int play_memory_fli(const void *fli_data, BITMAP *bmp, int loop, int (*callback)());

Reproduce una animación FLI o FLC del AutoDesk Animator, leyendo los datos de una copia de un fichero que se almacena en memoria. Puede obtener el puntero fli_data reservando un bloque de memoria con malloc y copiando el fichero FLI allí, o importando un FLI en un fichero de datos con el grabber. Reproducir animaciones desde la memoria es obviamente más rápido que desde disco, y es particularmente útil con animaciones FLI pequeñas que se repiten. Sin embargo, las animaciones pueden fácilmente llegar a ser grandes, por lo que en la mayoría de los casos sería mejor que use play_fli().

- **open_fli**

int open_fli(const char *filename); int open_memory_fli(const void *fli_data);

Abre un fichero FLI para reproducirlo, leyendo los datos del disco o de la memoria respectivamente. Devuelve FLI_OK con éxito. La información del FLI actual está almacenada en variables globales, por lo que sólo puede tener una animación abierta a la vez.

- **close_fli**

```
void close_fli();
```

Cierra un FLI cuando haya acabado con él.

- **next_fli_frame**

```
int next_fli_frame(int loop);
```

Lee el siguiente cuadro de la animación actual. Si loop está activado, el reproductor volverá al principio cuando llegue al final del fichero, o devolverá FLI_EOF si loop está desactivado. Devuelve FLI_OK con éxito, FLI_ERROR o FLI_NOT_OPEN si hubo error, y FLI_EOF al alcanzar el final del fichero. El cuadro es leído y almacenado en las variables globales fli_bitmap y fli_palette.

- **fli_bitmap**

```
extern BITMAP *fli_bitmap;
```

Contiene el cuadro actual de la animación FLI/FLC.

- **fli_palette**

```
extern PALETTE fli_palette;
```

Contiene la paleta del FLI actual.

- **fli_bmp_dirty_from**

```
extern int fli_bmp_dirty_from; extern int fli_bmp_dirty_to;
```

Estas variables son fijadas por next_fli_frame() para indicar qué parte de fli_bitmap ha cambiado desde la última llamada a reset_fli_variables(). Si fli_bmp_dirty_from es mayor que fli_bmp_dirty_to, el bitmap no cambió. De otro modo, las líneas de fli_bmp_dirty_from a fli_bmp_dirty_to (inclusive) fueron alteradas. Puede usar estos valores cuando copie fli_bitmap en pantalla, para evitar mover datos innecesariamente.

- **fli_pal_dirty_from**

```
extern int fli_pal_dirty_from; extern int fli_pal_dirty_to;
```

Estas variables son fijadas por next_fli_frame() para indicar qué parte de fli_palette fue cambiada desde la última llamada a reset_fli_variables(). Si

fli_pal_dirty_from es mayor que fli_pal_dirty_to, la paleta no fue modificada. De otro modo, los colores de fli_pal_dirty_from a fli_pal_dirty_to (inclusive) fueron alterados. Puede usar estas variables cuando actualice la paleta hardware, para evitar llamadas innecesarias a set_palette().

- **reset_fli_variables**

```
void reset_fli_variables();
```

Una vez haya hecho lo que fuese a hacer con fli_bitmap y fli_palette, llame esta función para resetear las variables fli_bmp_dirty_* y fli_pal_dirty_*.

- **fli_frame**

```
extern int fli_frame;
```

Variable global que contiene el número de cuadro actual del fichero FLI. Esto es útil para sincronizar otros eventos con la animación, por ejemplo podría comprobarla en la función callback de play_fli() y usarla para reproducir un sample en un punto particular.

- **fli_timer**

```
extern volatile int fli_timer;
```

Variable global para cronometrar la reproducción FLI. Cuando abre un fichero FLI, una interrupción de temporizador es instalada, la cual incrementa esta variable cada vez que se va a visualizar un nuevo cuadro. Llamando a next_fli_frame() la decrementa, por lo que puede comprobarla y saber el momento de enseñar un nuevo cuadro cuando sea mayor que cero.

RUTINAS DE INICIALIZAIÓN DE SONIDO

- **detect_digi_driver**

```
int detect_digi_driver(int driver_id);
```

Detecta si el dispositivo especificado de sonido digital está disponible. Devuelve el máximo número de voces que el driver puede proporcionar, o cero si el hardware no está presente. Esta función debe llamarse _antes_ de install_sound().

- **detect_midi_driver**

```
int detect_midi_driver(int driver_id);
```

Detecta si el dispositivo especificado de sondo MIDI está disponible. Devuelve el número máximo de voces que el dirver puede proporcionar, o cero si el hardware no está presente. Hay dos valores especiales que pueden ser devueltos y que debería controlar: si la función devuelve -1 se trata de un controlador ladrón de notas (ej. DIGMID) que comparte las voces con el driver de sonido digital actual, y si devuelve 0xFFFF es un dispositivo externo como un MPU-401, en los cuales no hay manera de determinar cuantas voces hay disponibles. Esta función debe llamarse _antes_ de install_sound().

- **reserve_voices**

```
void reserve_voices(int digi_voices, int midi_voices);
```

Llame esta función para especificar el número de voces que van a ser usadas por los controladores de sonido digital y MIDI respectivamente. Esto debe ser llamado _antes_ de llamar install_sound(). Si reserva demasiadas voces, las llamadas siguientes a install_sound() fallarán. Cuántas voces haya disponibles, depende del controlador, y en algunos casos llegará a reservar más de las deseadas (ejemplo: los controladores de música sintética FM siempre dan 9 voces en una OPL2 y 18 en una OPL3, y el controlador de sonido digital de la SB redondeará el número de voces al siguiente valor potencia de dos). Pase un valor negativo para recuperar los valores por defecto. Debería tener en cuenta, que la calidad del sonido es inversamente proporcional al número de voces que reserve, por lo que no reserve más de las que necesite.

- **set_volume_per_voice**

void set_volume_per_voice(int scale);

Por defecto, cuando reserva más voces para su driver de sonido digital, Allegro intentará reducir el volúmen de cada una de ellas para compensarlo. Esto se hace para evitar la distorsión. Por defecto el volúmen por voz es tal, que si reserva n voces, puede reproducir hasta n/2 sonidos normalizados centrados sin riesgo de distorsión. La excepción es cuando tiene menos de 8 voces, ya que el volúmen queda igual que para 8 voces.

Si el sonido resultante es demasiado alto o demasiado bajo, esta función puede ser usada para ajustar el volúmen de cada voz. Primero debería verificar que sus altavoces están ajustados a un volúmen razonable, el volúmen global de Allegro ajustado al máximo (mire set_volume() más abajo), y cualquier otro tipo de mezcladores como el control de volúmen de Windows están ajustados razonablemente.

Una vez haya determinado que el volúmen de Allegro no es ideal para su aplicación, use esta función para ajustarlo. Esto debe ser realizado _antes_ de llamara install_sound(). Note que esta función por ahora sólo es relevante para los drivers que usan el mezclador de Allegro (la mayoría de ellos).

Si pasa un 0 a esta función, cada sonido centrado será reproducido al máximo volúmen posible sin distorsión, al igual que los sonidos reproducidos con un driver mono. Los sonidos en los extremos izquierdo y derecho serán distorsionados si se reproducen al máximo volúmen. Si quiere reproducir sonidos panoramizados sin distorsión al máximo volúmen, debe pasar 1 a esta función. Nota: esto es diferente del comportamiento que había en las WIPs 3.9.34, 3.9.35 y 3.9.36. Si usa esta función con cualquiera de esas versiones, deberá incrementar su parámetro en uno para obtener el mismo volúmen.

Cada vez que incremente el parámetro en uno, el volúmen de cada voz será reducido a la mitad. Por ejemplo, si pasa 4, podrá reproducir hasta 16 sonidos centrados con volúmen máximo sin ninguna distorsión.

Aquí están los valores por defecto, dependiendo del número de voces:

1-8 voces	-set_volume_per_voice(2);
16 voces	-set_volume_per_voice(3);
32 voces	-set_volume_per_voice(4);
64 voces	-set_volume_per_voice(5);

Por supuesto esta función no modificará el volúmen que usted especifique mediante play_sample() o voice_set_volume(). Simplemente alterará la salida

global del programa. Si reproduce sonidos a un volúmen bajo, o si no están nromalizados, podrá reproducir más sonidos simultáneamente sin distorsión.

Aviso: Allegro usa una tabla de recorte para recortar la onda de sonido. Esta tabla es lo suficientemente grande para acomodar un sonido total de hasta 4 veces el máximo posible sin distorsión. Si su volúmen se sale de éste límite, la onda 'dará la vuelta' (los picos serán valles y viceversa), distorsionando aun más el sonido. Debe tener cuidado para que esto no ocurra.

Es recomendado que introduzca el valor de este parámetro de forma fija en su programa en vez de ofrecerlo al usuario. El usuario puede alterar el volúmen con el fichero de configuración, o puede modificarlo con set_volume().

Para restaurar el volúmen por voz a su comportamiento habitual, pase -1.

- **install_sound**

```
int install_sound(int digi, int midi, const char *cfg_path);
```

Inicializa el módulo de sonido. Normalmente debería pasarle DIGI_AUTODETECT y MIDI_AUTODETECT como los parámetros de controlador, en cuyo caso Allegro leerá la configuración del hardware del fichero de configuración actual. Esto permite al usuario seleccionar diferentes valores mediante una utilidad de setup: vea la sección de configuración para ver los detalles. Alternativamente, veal la documentación específica de la plataforma para tener una lista de los drivers disponibles. El parámetro cfg_path está presente sólo por compatibilidad con las versiones anteriores de Allegro, y no tiene ningún efecto. Devuelve cero si el sonido se instaló correctamente, y -1 si falló. Si falla guardará una descripción del problema en allegro_error.

- **remove_sound**

```
void remove_sound();
```

Limpia el entorno cuando haya acabado con las rutinas de sonido. Normalmente no hace falta que llame esto, ya que allegro_exit() lo hará por usted.

- **set_volume**

```
void set_volume(int digi_volume, int midi_volume);
```

Altera el volumen de salida del sonido. Especifique el volumen para la reproducción de MIDIs y samples digitales, con enteros de 0 a 255, o pase un valor negativo para no cambiar alguno de los ajustes. Es posible que esta rutina use el mezclador de hardware para controlar el volumen, de otro modo,

le dirá a los reproductores de samples y MIDI que simulen el mezclador por software.

RUTINAS DE SAMPLES DIGITALES

- **load_sample**

```
SAMPLE *load_sample(const char *filename);
```

Carga un sample de un fichero, devolviendo un puntero a él, o NULL si hay error. Actualmente, esta función soporta ficheros WAV mono o estéreo y ficheros VOC mono, tanto de 8 como 16 bits.

- **load_wav**

```
SAMPLE *load_wav(const char *filename);
```

Carga un sample de un fichero RIFF WAV.

- **load_voc**

```
SAMPLE *load_voc(const char *filename);
```

Carga un sample de un fichero VOC de Creative Labs.

- **create_sample**

```
SAMPLE *create_sample(int bits, int stereo, int freq, int len);
```

Construye una nueva estructura de samples del tipo especificado. El campo data apunta a un bloque de datos de onda: lee la definición de la estructura en allegro.h para más detalles.

- **destroy_sample**

```
void destroy_sample(SAMPLE *spl);
```

Destruye una estructura de sample cuando no la necesita. Es seguro llamar esto incluso cuando el sample está siendo reproducido, porque lo comprueba y apaga si está activo.

- **lock_sample**

```
void lock_sample(SAMPLE *spl);
```

Bajo DOS, bloquea toda la memoria usada por el sample. Normalmente no necesita llamar a esta función porque load_sample() y create_sample() lo harán por usted.

- **play_sample**

int play_sample(const SAMPLE *spl, int vol, int pan, int freq, int loop);

Reproduce un sample con el volumen, panoramización y frecuencia especificados. El volumen y panoramización van de 0 (min/izquierda) a 255 (máx/derecha). La frecuencia no es absoluta, es relativa: 1000 representa la frecuencia a la que el sample fue grabado, 2000 es el doble, etc. Si la variable de repetición está activada, el sample será repetido hasta que llame stop_sample(), y puede ser manipulado mientras es reproducido llamando adjust_sample().

- **adjust_sample**

void adjust_sample(const SAMPLE *spl, int vol, int pan, int freq, int loop);

Altera los parámetros de un sample mientras es reproducido (útil para manipular sonidos con repetición). Puede alterar el volumen, la panoramización y frecuencia, y también puede anular la variable de repetición, lo que parará el sample cuando llegue al final del loop. Si hay varias copias del mismo sample activas, esto ajustará el primero que vaya a ser reproducido. Si el sample no está siendo reproducido, esto no tiene efecto.

- **stop_sample**

void stop_sample(const SAMPLE *spl);

Mata un sample, algo necesario si tiene uno en modo repetición. Si hay varias copias del sample, todas serán paradas.

- **voice control**

Si necesita controlar los samples con más detalle, puede usar las funciones de voz de bajo nivel en vez de llamar play_sample(). Esto conlleva más trabajo, porque debe reservar y liberar las voces explícitamente, ya que éstas no se liberan solas al acabar la reproducción, pero esto le permite especificar precisamente lo que quiere hacer con el sonido. Incluso puede modificar algunos campos de la estructura sample:

int priority; De 0 a 255 (por defecto 255), esto indica cómo las voces se reservan si intenta reproducir más de las que controla el controlador. Esto puede ser usado para asegurarse de que los sonidos secundarios son recortados mientras que los importantes son reproducidos.unsigned long loop_start;unsigned long loop_end; Posiciones de repetición en unidades de sample, por defecto ajustadas al comienzo y final del sample.

- **allocate_voice**

int allocate_voice(const SAMPLE *spl);

Reserva una voz de la tarjeta y la prepara para reproducir el sample especificado, ajustando parámetros por defecto (volumen máximo, panoramización central, sin cambio de frecuencia, sin loop). Cuando acabe de usar la voz, debe liberarla llamando deallocate_voice() o release_voice(). Devuelve el número de voz, o -1 si no hay voces libres.

- **deallocate_voice**

void deallocate_voice(int voice);

Libera una voz de la tarjeta, parando su sonido y liberando los recursos que estuviese utilizando.

- **reallocate_voice**

void reallocate_voice(int voice, const SAMPLE *spl);

Ajusta una voz previamente reservada para usar un sample diferente. Llamar reallocate_voice(voice, sample) es equivalente a:

deallocate_voice(voice);
voice = allocate_voice(sample);

- **release_voice**

void release_voice(int voice);

Libera una voz, indicando que ya no está interesado en su manipulación. El sonido seguirá siendo reproducido, y los recursos que usa serán liberados automáticamente cuando acabe la reproducción. Esto es esencialmente lo mismo que deallocate_voice(), pero espera a que los sonidos acaben para hacer efecto.

- **voice_start**

void voice_start(int voice);

Activa una voz, usando los parámetros que le haya ajustado.

- **voice_stop**

void voice_stop(int voice);

Para una voz, almacenando la posición actual y estado para que luego pueda continuar la reproducción llamando voice_start().

- **voice_set_priority**

```
void voice_set_priority(int voice, int priority);
```

Ajusta la prioridad de una voz (rango 0-255). Esto es usado para decidir qué voces serán recortadas si intenta usar más de las que puede soportar el controlador de la tarjeta de sonido.

- **voice_check**

```
SAMPLE *voice_check(int voice);
```

Comprueba si una voz está activa, devolviendo el sample que está siendo reproducido, o NULL si la voz está inactiva (ej: la voz ha sido liberada, o ha llamado a release_voice() y el sample a terminado de ser reproducido).

- **voice_get_position**

```
int voice_get_position(int voice);
```

Devuelve la posición actual de la voz, en unidades de sample, o -1 si ha finalizado la reproducción.

- **voice_set_position**

```
void voice_set_position(int voice, int position);
```

Ajusta la posición de una voz, en unidades de sample.

- **voice_set_playmode**

```
void voice_set_playmode(int voice, int playmode);
```

Ajusta el estado de repetición de una voz. Esto puede hacerse mientras se reproduce la voz, por lo que puede reproducir un sample con repetición (teniendo el comienzo y final del loop ajustados correctamente), y entonces borrar la variable de repetición cuando quiera finalizar el sonido, lo que hará que éste llegue al final del loop, continúe con la siguiente parte del sample, y acabe de modo normal. El parámetro mode es un campo de bits que contiene los siguientes valores.

PLAYMODE_PLAY
Reproduce el sample una sola vez. Esto está ajustado por defecto si no fija la variable de repetición.

PLAYMODE_LOOP
Repite el sample, saltando al principio del bucle cuando se llegue al final de éste.

PLAYMODE_FORWARD

Reproduce el sample desde el comienzo hasta el final. Esto está ajustado por defecto si no activa el bit PLAYMODE_BACKWARD.

PLAYMODE_BACKWARD
Invierte la dirección del sample. Si combina esto con el bit de repetición, el sample saltará al final del loop cuando llegue al comienzo (esto es: no necesita invertir los valores de comienzo y final del loop cuando reproduzca un sample en modo invertido).

PLAYMODE_BIDIR
Si se usa en combinación con el bit loop, hace que el sample cambie de dirección cada vez que llega al extremo del bucle, por lo que alterna la dirección de reproducción.

- **voice_get_volume**

```
int voice_get_volume(int voice);
```

Devuelve el volumen de la voz, en el rango 0-255.

- **voice_set_volume**

```
void voice_set_volume(int voice, int volume);
```

Ajusta el volumen de la voz, en el rango 0-255.

- **voice_ramp_volume**

```
void voice_ramp_volume(int voice, int time, int endvol);
```

Comienza un cambio de volumen (crescendo o diminuendo) desde el volumen actual al volumen final, especificando time en milisegundos.

- **voice_stop_volumeramp**

```
void voice_stop_volumeramp(int voice);
```

Interrumpe una operación de cambio de volumen.

- **voice_get_frequency**

```
int voice_get_frequency(int voice);
```

Devuelve la frecuencia actual en Hz.

- **voice_set_frequency**

```
void voice_set_frequency(int voice, int frequency);
```

Ajusta la frecuencia de la voz en Hz.

- **voice_sweep_frequency**

```
void voice_sweep_frequency(int voice, int time, int endfreq);
```

Comienza un cambio de frecuencia (glissando) desde la frecuencia actual hasta la frecuencia final, especificando time en milisegundos.

- **voice_stop_frequency_sweep**

```
void voice_stop_frequency_sweep(int voice);
```

Interrumpe una operación de cambio de frecuencia.

- **voice_get_pan**

```
int voice_get_pan(int voice);
```

Devuelve la panoramización actual, desde 0 (izquierda) hasta 255 (derecha).

- **voice_set_pan**

```
void voice_set_pan(int voice, int pan);
```

Ajusta la panoramización, desde 0 (izquierda) hasta 255 (derecho).

- **voice_sweep_pan**

```
void voice_sweep_pan(int voice, int time, int endpan);
```

Comienza una panoramización (movimiento izquierda derecha) desde la posición actual hasta la posición endpan, especificando time en milisegundos.

- **voice_stop_pan_sweep**

```
void voice_stop_pan_sweep(int voice);
```

Interrumpe una panoramización.

- **voice_set_echo**

```
void voice_set_echo(int voice, int strength, int delay);
```

Ajusta el parámetro de eco para una voz (no implementado actualmente).

- **voice_set_tremolo**

```
void voice_set_tremolo(int voice, int rate, int depth);
```

Ajusta el parámetro de trémolo para una voz (no implementado actualmente).

- **voice_set_vibrato**

```
void voice_set_vibrato(int voice, int rate, int depth);
```

Ajusta el parámetro de vibrado para una voz (no implementado actualmente).

load_midi
MIDI *load_midi(const char *filename);
Carga un fichero MIDI (maneja ambos formatos 0 y 1), devolviendo un puntero a la estructura MIDI, o NULL si hubo

problemas.
destroy_midi
void destroy_midi(MIDI *midi);
Destruye una estructura MIDI cuando ya no la necesite. Es seguro llamar esto incluso cuando el fichero MIDI está siendo reproducido, porque lo comprueba y detiene en caso de que esté activo.
lock_midi
void lock_midi(MIDI *midi);
Bajo DOS, bloquea toda la memoria usada por un fichero MIDI. Normalmente no necesita llamar a esta función porque load_midi() lo hace por usted.
play_midi
int play_midi(MIDI *midi, int loop);
Reproduce el fichero MIDI especificado, deteniendo cualquier música anterior. Si la variable loop está activada, los datos serán repetidos hasta que los sustituya con otra cosa, de otro modo se parará la música al final del fichero. Pasando un puntero NULL parará cualquier música que esté siendo reproducida. Devuelve distinto de cero si hubo problemas (esto puede ocurrir si un controlador wavetable cacheable no consigue cargar los samples requeridos, o al menos ocurrirá en el futuro cuando alguien escriba algunos controladores wavetable cacheables :-)
play_looped_midi
int play_looped_midi(MIDI *midi, int loop_start, int loop_end);
Reproduce un fichero MIDI con una posición de bucle definida por el usuario. Cuando el reproductor llega al final del bucle o al final del fichero (loop_end puede ser -1 para repetir en EOF), volverá al principio del comienzo del bucle. Ambas posiciones son especificadas en el mismo formato de golpes de ritmo que la variable midi_pos.
stop_midi
void stop_midi();
Para la música que esté siendo reproducida. Esto es lo mismo que llamar play_midi(NULL, FALSE).
midi_pause
void midi_pause();

Pone el reproductor MIDI en pausa.
midi_resume
void midi_resume();
Continua la reproducción de un MIDI pausado.
midi_seek

int midi_seek(int target);
Avanza hasta la posición especificada (midi_pos) en el fichero MIDI usado. Si el objetivo está antes en el fichero que el midi_pos actual, avanza desde el principio; de otro modo busca desde la posición actual. Devuelve cero si no lo ha conseguido, no-cero si llega al final del fichero (1 significa que paró la reproducción, 2 significa que volvió a reproducir desde el principio). Si la función se para porque llegó a EOF, midi_pos contendrá el valor negativo de la longitud del fichero MIDI.
midi_out

void midi_out(unsigned char *data, int length);
Introduce un bloque de comandos MIDI en el reproductor en tiempo real, permitiéndole activar notas, tocar campanas, etc, sobre el fichero MIDI que esté siendo reproducido.
load_midi_patches
int load_midi_patches();
Fuerza al controlador MIDI a cargar un conjunto de patches completo para ser usados. Normalmente no deberá llamar esto, porque Allegro automáticamente carga todos los datos requeridos por el fichero MIDI seleccionado, pero debe llamar esto antes de mandar mensajes de cambio de programa vía comando midi_out(). Devuelve distinto de cero si ocurrió un fallo.
midi_pos
extern volatile long midi_pos;
Contiene la posición actual (número de beat) del fichero MIDI, o un número negativo si no se está reproduciendo ninguna música. Util para sincronizar animaciones con la música, y para comprobar si un fichero MIDI se ha acabado de reproducir.
midi_loop_start
extern long midi_loop_start;
extern long midi_loop_end;
Los puntos de comienzo y final del bucle, ajustados por la función play_looped_midi(). Estos pueden ser alterados mientras suena la música, pero debería estar seguro de ponerlos a valores sensatos (comienzo < final). Si está cambiando ambos al mismo tiempo, asegúrese de alterarlos en el mismo orden en caso de que una interrupción midi ocurra entre sus dos cambios. Si los valores están a -1, representan el comienzo y final del fichero respectivamente.
midi_msg_callback

extern void (*midi_msg_callback)(int msg, int byte1, int byte2);
extern void (*midi_meta_callback)(int type, const unsigned char *data, int length);
midi_sysex_callback
extern void (*midi_sysex_callback)(const unsigned char *data, int length);
Funciones de enganche que permiten interceptar eventos MIDI del reproductor. Si se activan a cualquier cosa menos NULL, estas rutinas serán llamadas por cada mensaje MIDI, metaevento, y bloque de datos exclusivo del sistema respectivamente. Estas funciones serán ejecutadas en un contexto de control de interrupción, por lo que todo el código y datos que usen debería estar bloqueado (locked), y no deben llamar funciones del sistema operativo. En general, simplemente use estas rutinas para activar algunas variables y responder a ellas más tarde en su código principal.
load_ibk
int load_ibk(char *filename, int drums);
Lee una definición de un fichero patch .IBK usado por el controlador Adlib. Si los tambores están activados, lo cargará como un patch de percusión, de otro modo reemplazará el conjunto de instrumentos MIDI General. Puede llamar esto antes o después de iniciar el código de sonido, o simplemente puede activar las variables ibk_file e ibk_drum_file en el fichero de configuración para cargar los datos automáticamente. ¡Fíjese que esta función no tiene ningún efecto en otros controladores que no sean Adlib!
RUTINAS DE FLUJO DE SONIDO
Las rutinas de flujo de sonido son para reproducir sonidos digitales que son demasiado grandes para caber en la estructura SAMPLE, bien porque son ficheros enormes que quiere cargar en trozos según necesita los datos, o porque está haciendo algo inteligente como generar la onde del sonido en tiempo real.
play_audio_stream
AUDIOSTREAM *play_audio_stream(int len, bits, stereo, freq, vol, pan);
Esta función crea un nuevo flujo de audio y empieza a reproducirlo. El parámetro len es el tamaño de cada búffer de transferencia (en samples), que normalmente debería ser una potencia de 2 y cercana a 1k: búffers más grandes son más eficientes y requieren menos actualizaciones, pero hay un desfase mayor entre los datos que usted porporciona y los que se están reproduciendo actualmente. El parametro bits debe ser 8 o 16, freq es la frecuencia de muestreo de los datos, y los valores vol y pan usan el el mismo rango 0-255, como las funciones normales de reproducción de samples. Si quiere ajustar la frecuencia, el volumen o la panoramización del flujo una vez se esté reproduciendo, puede usar las funciones normales voice_*() con stream->voice como un parámetro. Los datos del sample estan siempre en formato sin signo, con formas de onda en estéreo que consisten en samples alternativos izquierda/derecha.
stop_audio_stream

void stop_audio_stream(AUDIOSTREAM *stream);
Destruye un flujo de audio cuando no lo necesite más.
get_audio_stream_buffer
void *get_audio_stream_buffer(AUDIOSTREAM *stream);
Debe llamar este función a intervalos regulares mientras el flujo de audio está siendo reproducido, para proveer el siguiente buffer de datos del sample (cuanto más pequeño sea el tamaño del buffer del flujo, más frecuentemente debe llamar esta función). Si devuelve NULL, el flujo todavía está reproduciendo los datos y no debe hacer nada. Si devuelve un valor, esa es la localización del próximo buffer a tocar, y debería cargar el número apropiado de samples (tantos como especificó al crear el flujo) a esa dirección, por ejemplo usando un fread() de un fichero. Después de llenar el buffer con datos, llame free_audio_stream_buffer() para indicar que los datos nuevos ahora son válidos. Fíjese que esta función no debería ser llamada desde una función de temporizador.
free_audio_stream_buffer
void free_audio_stream_buffer(AUDIOSTREAM *stream);
Llame esta función después de que get_audio_stream_buffer() devuelva una dirección que no sea NULL, para indicar que ya ha cargado un nuevo bloque de samples en esa dirección y que los datos están listos para ser reproducidos.
RUTINAS DE GRABACIÓN DE SONIDO
install_sound_input
int install_sound_input(int digi_card, int midi_card); Inicializa el módulo de grabación de sonido, devolviendo cero si no hubo problemas. Debe instalar el sistema normal de reproducción de sonido antes de llamar esta rutina. Los dos parámetros de tarjetas deben ser los mismos que en install_sound(), incluyendo DIGI_NONE y MIDI_NONE para desactivar partes del módulo, o DIGI_AUTODETECT y MIDI_AUTODETECT para adivinar el hardware.
remove_sound_input
void remove_sound_input();
Desactiva el módulo cuando haya acabado de usarlo. Normalmente no debe llamar esta función, porque remove_sound() y/o allegro_exit() lo harán por usted.
get_sound_input_cap_bits
int get_sound_input_cap_bits();
Comprueba qué formatos de sonido son soportados por el controlador de entrada de audio, devolviendo uno de los valores del campo de bits:
0 = entrada de audio no soportada
8 = entrada de audio de ocho bits soportada
16 = entrada de audio de dieciséis bits soportada
24 = entrada de audio de ocho y dieciséis bits soportada
get_sound_input_cap_stereo
int get_sound_input_cap_stereo();

Comprueba si el controlador de entrada de audio actual es capaz de grabar en estéreo.
get_sound_input_cap_rate
int get_sound_input_cap_rate(int bits, int stereo);
Devuelve la frecuencia de grabación de samples máxima posible en el formato especificado, o cero si los ajustes no son soportados.
get_sound_input_cap_parm
int get_sound_input_cap_parm(int rate, int bits, int stereo);
Comprueba si la frecuencia de grabación especificada, número de bits y mono/estéreo es soportado por el controlador de audio actual, devolviendo uno de los siguientes valores:
0 = es imposible grabar en este formato
1 = grabar es posible, pero la salida de audio será suspendida
2 = es posible grabar y reproducir sonidos a la vez
-n = razón de muestreo no soportada, pero la razón 'n' puede funcionar
set_sound_input_source
int set_sound_input_source(int source);
Selecciona la fuente de la entrada de audio, devolviendo cero si no hubo problemas o -1 si el hardware no proporciona un registro de selección de entrada. El parámetro debe ser uno de los valores:
SOUND_INPUT_MIC
SOUND_INPUT_LINE
SOUND_INPUT_CD
start_sound_input
int start_sound_input(int rate, int bits, int stereo);
Comienza a grabar en el formato especificado, suspendiendo la reproducción de sonidos si es necesario (esto sucederá siempre con los controladores actuales). Devuelve el tamaño del buffer en bytes si hubo éxito, o cero si hubo algún error.
stop_sound_input
void stop_sound_input();
Para la grabación, ajustando la tarjeta de vuelta al modo normal de reproducción.
read_sound_input
int read_sound_input(void *buffer);
Recupera el buffer de audio grabado más reciente en el lugar especificado, devolviendo no-cero si el buffer ha sido copiado, o cero si todavía no hay nuevos datos disponibles. El tamaño del buffer puede ser obtenido comprobando el valor de retorno de start_sound_input(). Debe llamar esta función a
intervalos regulares durante la grabación (típicamente unas 100 veces por segundo), o podría perder datos. Si no puede hacer esto lo suficientemente rápido, use la función callback digi_recorder() para almacenar la onda de sonido en un buffer más grande que haya creado antes. Nota: muchas tarjetas

de sonido reproducen un click o sonido raro cuando alternan entre los modos de grabación y reproducción, por lo que es buena idea descartar el primer buffer después de comenzar la grabación. La onda siempre se almacena en formato sin signo, con los datos estéreo siendo samples alternados izquierda/derecha.

digi_recorder

extern void (*digi_recorder)();

Si está activada, esta función es llamada por el controlador de entrada de sonido siempre que un nuevo buffer de sonido está disponible, momento en el que puede usar read_sound_input() para copiar los datos a un lugar permanente. Esta rutina se ejecuta en contexto de interrupción, por lo que debe ejecutarse muy rápidamente, el código y la memoria que modifica debe estar bloqueada (locked), y no puede llamar desde ella rutinas de sistema o acceder a ficheros del disco.

midi_recorder

extern void (*midi_recorder)(unsigned char data);

Si está activada, esta función es llamada por el controlador de entrada MIDI siempre que un nuevo byte de datos MIDI esté disponible. Se ejecuta en contexto de interrupción, por lo que debe ser muy rápida y su código/datos deben estar bloqueados (locked).

RUTINAS DE FICHEROS Y COMPRESIÓN

Las siguientes rutinas implementan un sistema de ficheros I/O con buffer rápido, que soporta la lectura y escritura de ficheros comprimidos usando un algoritmo de buffer de anillo basado en el compresor LZSS de Haruhiko Okumura. Esto no consigue tan buenas compresiones como zip o lha, pero la descompresión es muy rápida y no requiere mucha memoria. Los ficheros comprimidos siempre comienzan con el valor de 32 bits F_PACK_MAGIC, y autodetecta ficheros con el valor F_NOPACK_MAGIC.

Los siguients bit FA_* están garantizados en todas las plataformas: FA_RDONLY, FA_HIDDEN, FA_SYSTEM, FA_LABEL, FA_DIREC y FA_ARCH. No use otros bits de DOS/Windows, o su código no compilará en otras plataformas. Los bits FA_SYSTEM, FA_LABEL y FA_ARCH sólo son útiles bajo DOS/Windows (entradas con el bit de sistema, archivo y etiquetas de volúmen). FA_RDONLY es para directorios con el bit de sólo lectura en sistemas tipo DOS, o directorios sin permiso de escritura por el usuario actual en sistemas tipo Unix. FA_HIDDEN es para ficheros ocultos en DOS, o aquellos que compeinzan con '.' en sistemas Unix (excepto los ficheros '.' y '..'). FA_DIREC representa directorios. Los bits se pueden combinar usando '|' (operador OR binario).

Cuando estos bits son pasados a las funciones como el parámetro 'attrib', representan un superconjunto de los bits que debe tener un fichero para ser incluído en la búsqueda.

Esto es, para que un fichero encaje con el patrón, sus atributos pueden contener cualquiera de los bits especificados, pero no debe contener ningúno

de los bits no especificados. Por lo tanto, si usa 'FA_DIREC | FA_RDONLY', los ficheros y directorios normales serán incluídos junto con los ficheros y directorios de sólo lectura, pero no los ficheros y directorios ocultos. Similarmente, si usa 'FA_ARCH' entonces tanto los ficheros archivados como no archivados serán incluídos.

get_executable_name

void get_executable_name(char *buf, int size);

Llena buf con la ruta completa del ejecutable actual, escribiendo como mucho size bytes. Esto normalmente viene de argv[0], pero en los sistemas Unix donde argv[0] no especifica la ruta, se buscará el fichero en $PATH.

fix_filename_case

char *fix_filename_case(char *path);

Convierte un nombre de fichero a un estado estandarizado. En platadormas DOS, los nombres serán todo mayúsculas. Devuelve una copia del parámetro de camino.

fix_filename_slashes

char *fix_filename_slashes(char *path);

Convierte los separadores de directorios de un nombre de fichero a un carácter estándar. En plataformas DOS, esto es la antibarra. Devuelve una copia del parámetro de camino.

fix_filename_path

char *fix_filename_path(char *dest, const char *path, int size);

Convierte un nombre de fichero parcial en una ruta completa, escribiendo en dest como máximo el número de bytes especificados. Devuelve una copia del parámetro dest.

replace_filename

char *replace_filename(char *dest, const char *path, const char *filename, int size);

Sustituye el camino+nombre de fichero especificados con un nuevo nombre de fichero, escribiendo en dest como máximo el número de bytes especificados. Devuelve una copia del parámetro dest.

get_filename

char *get_filename(const char *path);

Cuando se le pasa el path específico de un fichero, devuelve un puntero a la porción del nombre del fichero. Tanto '\' como '/' son reconocidos como separadores de directorios.

get_extension

char *get_extension(const char *filename);

Cuando se le pasa un nombre de fichero completo (con o sin información de path) devuelve un puntero a la extensión del fichero.

append_filename

char *append_filename(char *dest, const char *path, const char *filename, int size);

Concatena el nombre de fichero especificado al final del camino especificado, escribiendo en dest como máximo el número de bytes especificados. Devuelve una copia del parámetro dest.
get_filename
char *get_filename(const char *path);
Cuando se le pasa el path específico de un fichero, devuelve un puntero a la porción del nombre del fichero. Tanto '\' como '/' son reconocidos como separadores de directorios.
get_extension
char *get_extension(const char *filename);
Cuando se le pasa un nombre de fichero completo (con o sin información de path) devuelve un puntero a la extensión del fichero.
put_backslash
void put_backslash(char *filename);
Si el último caracter de un nombre no es '\' o '/', esta rutina le añadirá '\'.
file_exists
int file_exists(const char *filename, int attrib, int *aret);
Chequea la existencia de un fichero de nombre y atributos dados (lea más arriba), devolviendo distinto de cero si el fichero existe. Si aret no es NULL, contendrá los atributos del fichero existente al acabar la llamada. Si ocurre un error, el código de error de sistema será almacenado en errno.
exists
int exists(const char *filename);
Versión reducida de file_exists(), que comprueba la existencia de ficheros normales, los cuales pueden tener los bits de archivo o sólo lectura activados, pero no son ocultos, directorios, ficheros de sistema, etc.
file_size
long file_size(const char *filename);
Devuelve el tamaño del fichero en bytes. Si el fichero no existe u ocurre un error, devolverá cero y almacenará el código de error de sistema en errno.
file_time

time_t file_time(const char *filename);
Devuelve el tiempo de modificación de un fichero (número de segundos desde las 00:00:00 GMT del 1 de Enero de 1970). delete_file
int delete_file(const char *filename);
Borra un fichero.

for_each_file
int for_each_file(const char *name, int attrib, void (*callback)(const char *filename, int attrib, int param), int param);
Encuentra todos los ficheros que se ajusten a la máscara (ej: *.exe) y atributos especificados (lea más arriba), y ejecuta callback() por cada uno de

ellos. A callback() se le pasan tres parámetros, el primero es la cadena que contiene el nombre completo del fichero, el segundo los atributos del fichero, y el tercer parámetro es un entero que es copia de param (puede usar esto para lo que quiera). Si ocurre un error, el código de error será almacenado en errno, y callback() puede abortar for_each_file al activar errno. Devuelve el número de llamadas con éxito hechas a callback().

al_findfirst

int al_findfirst(const char *pattern, struct al_ffblk *info, int attrib);

Función de bajo nivel para buscar ficheros. Esta función busca el primer fichero que concuerde con el patrón y los atributos de fichero especificados (lea más arriba). La información sobre el fichero (si existe) será puesta en la estructura al_ffblk que debe proveer usted. La función devuelve cero si se encontró un fichero, distinto de cero si no se encontró ninguno, y en este caso ajusta errno apropiadamente. La estructura al_ffblk tiene la siguiente forma:

struct al_ffblk{ int attrib; - atributos del fichero encontrado time_t time; - tiempo de modificación del fichero long size; - tamaño del fichero char name[512]; - nombre del fichero};

Hay más cosas en esta estructura, pero son para uso interno. al_findnext

int al_findnext(struct al_ffblk *info);

Esto encuentra el siguiente fichero en una búsqueda comenzada por al_findfirst. Devuelve cero si se encontró un fichero, distinto de cero si no se encontró ninguno, y en éste caso ajusta errno apropiadamente.

al_findclose

void al_findclose(struct al_ffblk *info);

Esto cierra una búsqueda previamente abierta mediante al_findfirst().

find_allegro_resource

int find_allegro_resource(char *dest, const char *resource, const char *ext, const char *datafile, const char *objectname, const char *envvar, const char *subdir, int size);

Busca un archivo de recursos, ej allegro.cfg o language.dat. Pasándole una cadena resource describiendo qué se está buscando, junto con una información extra opcional como la extensión por defecto, en qué datafile mirar, qué nombre de objeto debería tener en el datafile, cualquier variable de entorno que se tenga que chequear, y cualquier subdirectorio que le gustaría comprobar, así como la localización por defecto, esta función mira en un infierno de sitios distintos :-) Devuelve cero si ha tenido éxito, y guarda el path absoluto del fichero (como mucho size bytes) en el parámetro dest.

packfile_password

void packfile_password(const char *password);

Activa el password de encriptación que será usado para todas las operaciones de lectura/escritura con ficheros abiertos en el futuro con las funciones packfile de Allegro (estén comprimidos o nó). Los ficheros escritos con un password no pueden ser leídos a no ser que se seleccione el password correcto, por lo que cuidado: si olvida la clave, inadie podrá recuperar su

datos! Pase NULL o una cadena vacía para volver al modo normal, no encriptado. Si está usando esta función para evitar que otros accedan a sus ficheros de datos, tenga cuidado de no salvar una copia obvia de su clave en el ejecutable: si hay cadenas como "Soy la clave del fichero de datos", sería muy fácil acceder a sus datos :-)
Importante: tan pronto como haya abierto un fichero usando un password de encriptación, llame a packfile_password(NULL). Mejor aún, no use esta función. Nunca.
pack_fopen
PACKFILE *pack_fopen(const char *filename, const char *mode);
Abre un fichero según el modo, que puede contener cualquiera de los siguientes letras.
· 'r' - abrir fichero para leer.
· 'w' - abrir fichero para escribir, sobreescribiendo
datos existentes.
· 'p' - abrir fichero en modo comprimido. Los datos
serán comprimidos a medida que se escriben en el fichero, y automáticamente descomprimidos durante las operaciones de lectura. Los ficheros creados de este modo producirán basura si se intentan leer sin activar antes este modo.
· '!' - abrir fichero para escribir en modo normal,
sin compresión, pero añade el valor F_NOPACK_MAGIC al comienzo del fichero, para que luego pueda ser abierto en modo comprimido y Allegro autodetectará que los datos no necesitan ser descomprimidos.
En vez de estos modos, una de las constantes F_READ, FWRITE, F_READ_PACKED, F_WRITE_PACKED o F_WRITE_NOPACK puede ser usada como el parámetro de modo. Si todo funciona, pack_fopen() devuelve un puntero a una estructura de fichero, y con error, devuelve NULL y almacena el código de error en errno. Un intento de leer un fichero normal en modo comprimido activará errno a EDOM.
Las funciones de ficheros también entienden varios nombres "mágicos" que pueden ser usados por varios motivos. Estos nombres son:
· "#" - lee datos que han sido añadidos al fichero
ejecutable con la utilidad exedat, como si fuesen de un fichero independiente.
· 'nombre.dat#nombre_obj' - abre un objeto específico
de un fichero de datos, y lo lee como si fuese de un fichero normal. Puede crear ficheros de datos anidados exáctamente como una estructura normal de directorios, por ejemplo podría abrir el fichero 'nombre.dat#graficos/nivel1/datomapa'.
· '#nombre_obj' - combinación de lo de arriba, leer
un objeto de un fichero de datos que ha sido añadido al ejecutable.
Con estos nombres especiales, los contenidos de un objeto de un fichero de datos o de un fichero añadido pueden ser leídos de modo idéntico que un fichero normal, por lo que cualquiera de las funciones de acceso a ficheros de Allegro (ejemplo: load_pcx() y set_config_file()) pueden ser usadas para

leerlos. Sin embargo, no podrá escribir en estos ficheros: sólo pueden ser leídos. Además, debe tener su fichero de datos descomprimido o con compresión por objetos si planea leer objetos individuales (de otra manera, habrá una sobrecarga de búsqueda al ser leído). Finalmente, tenga en cuenta que los tipos de objetos especiales de Allegro no son los mismos que los de los ficheros de los que importe los datos. Cuando importe datos como bitmaps o samples en el grabber, éstos son convertidos a un formato específico de Allegro, pero el marcador de sintaxis de ficheros '#' lee los objetos como trozos binarios raw. Esto significa, que si por ejemplo, quiere usar load_pcx para leer una imagen de un fichero de datos, debería importarla como un bloque binario en vez de un objeto BITMAP.
pack_fdopen
PACKFILE *pack_fdopen(int fd, const char *mode);
Función de bajo nivel que convierte el descriptor de fichero POSIX en un PACKFILE. El modo puede tener los mismos valores que para pack_fopen() y debe ser compatible con el modo del descriptor del fichero. A diferencia de fdopen() de la libc, pack_fdopen() no es capaz de convertir un fichero parcialmente leído o escrito (es decir, el desplazamiento del fichero debe ser 0).
Devuelve un puntero a una estructura packfile con éxito, o NULL si hubo problemas almacenando el código de error en errno. Un intento de leer un fichero normal en modo comprimido hará que errno sea ajustado a EDOM.
packfile functions

int pack_fclose(PACKFILE *f);
int pack_fseek(PACKFILE *f, int offset); int pack_feof(PACKFILE *f);

int pack_ferror(PACKFILE *f); int pack_getc(PACKFILE *f);
int pack_putc(int c, PACKFILE *f); int pack_igetw(PACKFILE *f);

long pack_igetl(PACKFILE *f);
int pack_iputw(int w, PACKFILE *f); long pack_iputl(long l, PACKFILE *f); int pack_mgetw(PACKFILE *f);
long pack_mgetl(PACKFILE *f);
int pack_mputw(int w, PACKFILE *f); long pack_mputl(long l, PACKFILE *f);

long pack_fread(void *p, long n, PACKFILE *f);
long pack_fwrite(const void *p, long n, PACKFILE *f); char *pack_fgets(char *p, int max, PACKFILE *f);

int pack_fputs(const char *p, PACKFILE *f);
Todas estas funcionan como las funciones equivalentes stdio, excepto que pack_fread() y pack_fwrite() toman un sólo parámetro de tamaño en vez de

ese estúpido sistema de tamaño y num_elements, sólo puede avanzar en un fichero hacia delante desde la posición relativa actual, y pack_fgets() no incluye el retorno de carro en las cadenas que devuelve. Las rutinas pack_i* y pack_m leen y escriben valores de 16 y 32 bits usando los sistemas de orden de Intel y Motorola respectivamente. Tome nota que la búsqueda es muy lenta cuando lea ficheros comprimidos, y que debería ser evitada a no ser que sepa que el fichero no está comprimido.

pack_fopen_chunk

PACKFILE *pack_fopen_chunk(PACKFILE *f, int pack);

Abre sub-chunks en un fichero. Los chunks son primariamente usados por el código de ficheros de datos, pero pueden serle útiles para sus propias rutinas de ficheros. Un chunk provee una vista lógica de parte de un fichero, que puede ser comprimido como un ente individual y será automáticamente insertado y comprobará los contadores de tamaño para prevenir la lectura después del final del chunk. Para escribir un chunk en un fichero f, use este código:

/* Asumo que f es un PACKFILE * que ha sido abierto en modo escritura*/f = pack_fopen_chunk(f, pack);escribe datos en ff = pack_fclose_chunk(f);

Los datos escritos en el chunk serán precedidos con dos contadores de tamaño (32 bits, big-endian). Para los chunks sin compresión, éstos serán ajustados al tamaño de los datos del chunk. Para chunks comprimidos (creados al activar la variable pack), el primer tamaño es el tamaño real del chunk, y el segundo será el tamaño negativo de los datos descomprimidos.

Para leer el chunk, use este código:

/* Asumo que f es un PACKFILE * que ha sido abierto en modo escritura*/f = pack_fopen_chunk(f, FALSE);lee datos de ff = pack_fclose_chunk(f);

Esta secuencia leerá los contadores de tamaño creados cuando el chunk fue escrito, y automáticamente descomprimirá el contenido del chunk si fue comprimido. El tamaño también evitará leer después del final del chunk (Allegro devolverá

EOF si intenta esto), y automáticamente ignora los datos no leídos del chunk cuando llamae pack_fclose_chunk().

Los chunks pueden ser anidados unos dentro de otros al hacer llamadas repetidas a pack_fopen_chunk(). Al escribir un fichero, el estado de compresión es heredado del fichero padre, por lo que sólo tiene que activar la variable pack si el fichero padre no fue comprimido pero quiere comprimir los datos del chunk. Si el fichero padre ya está abierto en modo comprimido, activar la variable pack hará que los datos sean comprimidos dos veces: una cuando los datos son escritos en el chunk, y otra cuando el chunk es escrito en el fichero padre.

pack_fclose_chunk

PACKFILE *pack_fclose_chunk(PACKFILE *f);

Cierra un sub-chunk de un fichero, que previamente ha sido obtenido al llamar pack_fopen_chunk().

RUTINAS DE FICHEROS DE DATOS
Los ficheros de datos son creados por la utilidad grabber, y tienen la extensión .dat. Pueden contener bitmaps, paletas de color, fuentes, sonidos, música MIDI, animaciones FLI/FLC y cualquier otro tipo binario de datos que importe.
Atención: cuando use imágenes truecolor, debería activar el modo gráfico antes de cargar ningún bitmap! Si no, el formato (RGB o BGR) será desconocido, y el fichero probablemente será convertido erróneamente.
Mire la documentación en pack_fopen() para obtener información sobre como leer directamente de un fichero de datos.
load_datafile
DATAFILE *load_datafile(const char *filename);
Carga un fichero de datos en memora, devolviendo un puntero hacia él, o NULL si ha habido un error. Si el fichero de datos ha sido encriptado, primero tiene que usar la función packfile_password() para introducir la clave correcta. Mire grabber.txt para mas información. Si el fichero de datos contiene gráficos truecolor, debe entrar en modo gráfico o llamar set_color_conversion() antes de cargarlo.
load_datafile_callback
DATAFILE *load_datafile_callback(const char *filename, void (*callback)(DATAFILE *d));
Carga el datafile en memoria, llamando a la función de enganche (hook) especificada una vez por cada objeto en el fichero, pasándole un puntero al objeto leído recientemente. unload_datafile
void unload_datafile(DATAFILE *dat);
Libera todos los objetos de un fichero de datos. load_datafile_object

DATAFILE *load_datafile_object(const char *filename, const char *objectname);
Carga un objeto específico de un fichero dat. Esto no funcionará si elimina los nombres de los objetos del fichero, y será muy lento si salva el fichero de datos con compresión general. Mire grabber.txt para más información.
unload_datafile_object
void unload_datafile_object(DATAFILE *dat);
Libera un objeto previamente cargado con load_datafile_object().
find_datafile_object
DATAFILE *find_datafile_object(DATAFILE *dat, const char *objectname);
Busca en un fichero de datos que esté cargado un objeto con el nombre especificado, devolviendo un puntero a él, o NULL si el objeto no fue encontrado. Entiende '/' y '#' como separadores para paths de ficheros de datos anidados.
get_datafile_property
char *get_datafile_property(DATAFILE *dat, int type);
Retorna la propiedad especifica de un objeto, o una cadena vacía si la propiedad no esta presente. Mire grabber.txt para mas información.

register_datafile_object
void register_datafile_object(int id, void *(*load)(PACKFILE *f, long size), void (*destroy)(void *data));
Usado para añadir tipos de objetos propios, especificando las funciones de carga y destrucción de este tipo. Mire grabber.txt para mas información.
fixup_datafile
void fixup_datafile(DATAFILE *data);
Si está usando ficheros de datos compilados (producidos por la utilidad dat2s) que contienen imágenes truecolor, tiene que llamar esta función una vez haya puesto el modo de vídeo que vaya a usar, para convertir los valores de los colores al formato apropiado. Puede intercambiar los formatos RGB y BGR, y convertirlos a diferentes profundidades de color siempre que sea posible sin cambiar el tamaño de la imagen (por ejemplo: cambiando entre 15 y 16 bits de color para bitmaps y sprites RLE, y entre 24 y 32 bits de color para sprites RLE).
using datafiles
Cuando cargue un fichero de datos, obtendrá un puntero a un array de estructuras DATAFILE:
typedef struct DATAFILE{ void *dat; - puntero a los datos int type; - tipo del dato long size; tamaño de los datos en bytes void *prop; - propiedades de los objetos} DATAFILE;
El campo type puede tener uno de los siguientes valores:
DAT_FILE - dat apunta a un fichero de datos
anidadoDAT_DATA - dat apunta a un bloque ordinario de datosDAT_FONT - dat apunta a una fuenteDAT_SAMPLE dat apunta a un fichero de sonidoDAT_MIDI - dat apunta a un fichero MIDIDAT_PATCH - dat apunta a un 'patch' para la GUSDAT_FLI - dat apunta a una animación FLI/FLCDAT_BITMAP - dat apunta a una estructura BITMAPDAT_RLE_SPRITE - dat apunta a una estructura RLE_SPRITEDAT_C_SPRITE - dat apunta a un sprite compilado linearmenteDAT_XC_SPRITE - dat apunta a un sprite de modoXDAT_PALETTE - dat apunta a un array de 256 estructuras RGBDAT_END - bit especial que marca el final de una
lista de datos
El programa grabber también puede producir un fichero de cabecera que define el índice de los objetos dentro de un fichero de datos como una serie de constantes definidas, usando los nombres que les dio en el grabber. Por ejemplo, si creó un fichero de datos llamado foo.dat que contiene el bitmap llamado LA_IMAGEN, puede enseñarlo con el siguiente fragmento de código:
#include "foo.h"DATAFILE *data = load_datafile("foo.dat");draw_sprite(screen, data[LA_IMAGEN].dat, x, y);
Si está programando en C++. obtendrá un error porque el campo dat es un puntero void y draw_sprite espera un puntero BITMAP. Puede solucionarlo con una conversión de puntero. Ejemplo:

draw_sprite(screen, (BITMAP *)data[LA_IMAGEN].dat, x, y);
Cuando carge un sólo objeto de un fichero de datos, obtendrá un puntero a una estructura DATAFILE única. Esto significa que no puede acceder a él como un array, y no contiene el objeto DAT_END. Ejemplo:
objeto_musica = load_datafile_object("datos.dat", "MUSICA");play_midi(objeto_musica->dat);
RUTINAS MATEMÁTICAS DE PUNTO FIJO
Allegro trae algunas rutinas para trabajar con números de punto fijo, y define el tipo 'fixed' como un entero de 32 bits con signo. La parte alta es usada por el valor del entero y la parte baja es usada por el valor de la fracción, dando un rango de valores de -32768 a 32767 y un detalle de unos 4 o 5 decimales. Los números de punto fijo pueden ser asignados, comparados, añadidos, substraídos, negados y desplazados (para multiplicar o dividir por potencias de 2) usando los operadores de enteros normales, pero tendría que tener cuidado de usar las rutinas de conversión apropiadas cuando combine números de punto fijo con enteros o números de coma flotante. Escribir 'punto_fijo_1 + punto_fijo_2' esta bien, pero 'punto_fijo + entero' no esta bien.
itofix
fixed itofix(int x);
Convierte un valor de entero a punto fijo. Esto es lo mismo que x<<16.
fixtoi
int fixtoi(fixed x);
Convierte un valor de punto fijo a entero, redondeando. Si quieres evitar el redondeo, usa x>>16.
ftofix
fixed ftofix(float x);
Convierte un valor de coma flotante a punto fijo.
fixtof
float fixtof(fixed x);
Convierte un valor de punto fijo a coma flotante.
fmul
fixed fmul(fixed x, fixed y);
Un valor de punto fijo puede ser multiplicado o dividido por un entero con los operadores normales '*' y '/'. Sin embargo, para multiplicar dos valores de punto fijo necesita usar esta función.
Si hay desbordamiento o división por cero, errno será activado y el valor máximo posible será devuelto, pero errno no es limpiado si la operación es realizada con éxito. Esto significa que si va a comprobar un desbordamiento de división, debería poner errno=0 antes de llamar a fmul(). fdiv
fixed fdiv(fixed x, fixed y);
División de valores de punto fijo: mire fmul().
fadd
fixed fadd(fixed x, fixed y);

A pesar de que los números de punto fijo pueden ser añadidos con el operador normal de enteros '+', eso no le da protección contra desbordamientos. Si el desbordamiento es un problema, debería usar esta función. Es mas lenta que los operadores de enteros, pero si hay un desbordamiento de división, ajustará el tamaño del resultado en vez de dejarlo al azar, y activara errno.
fsub
fixed fsub(fixed x, fixed y);
Resta de números en punto fijo: mire fadd().
fceil
int fceil(fixed x);
Devuelve el menor entero que no sea menor que x. Es decir, redondea hacia el infinito positivo.
fixed point trig
Las funciones de raíz cuadrada, seno, coseno, tangente,
cosecante y secante están implementadas usando tablas precalculadas, que son muy rápidas pero no muy exactas. Por ahora, la cotangente realiza una búsqueda iterativa en la tabla de la tangente, por lo que es mas lenta que las otras.
Los ángulos están representados en formato binario con 256 siendo igual al círculo completo, 64 es un ángulo recto y así sucesivamente. Esto tiene la ventaja de que un 'and' a nivel de bits puede ser usado para que el ángulo quede entre cero y el círculo completo, eliminando esos tests cansinos 'if (angle >= 360)'.
fsin
fixed fsin(fixed x);
Mira la tabla precalculada del seno.
fcos
fixed fcos(fixed x);
Mira la tabla precalculada del coseno.
ftan
fixed ftan(fixed x);
Mira la tabla precalculada de la tangente.
fasin
fixed fasin(fixed x);
Mira la tabla de la cosecante.
facos
fixed facos(fixed x);
Mira la tabla de la secante.
fatan
fixed fatan(fixed x);

Cotangente de punto fijo.
fatan2
fixed fatan2(fixed y, fixed x);

Versión de punto fijo de la rutina atan2() de libc.
fsqrt
fixed fsqrt(fixed x);
Raíz cuadrada de punto fijo.
fhypot
fixed fhypot(fixed x, fixed y);
Hypotenusa en punto fijo (devuelve la raíz cuadrada de x*x + y*y).

fix class
Si está programando en C++ puede ignorar todo lo de arriba y usar la clase "fija", que sobrecarga muchos operadores para proveer conversión automática desde y hacia valores enteros y de coma flotante, y llama las rutinas de arriba cuando se necesitan. Sin embargo no debería mezclar la clase "fija" con los typedefs de punto fijo, ya que el compilador tratará los valores de punto fijo como enteros regulares e insertará conversiones innecesarias. Por ejemplo, si x es un objeto de clase fija, llamar fsqrt(x) devolverá un resultado erróneo. Debería usar sqrt(x) o x.swrt() en vez de eso.
RUTINAS MATEMÁTICAS 3D
Allegro también contiene algunas funciones de ayuda de 3d para manipular vectores, construir o usar matrices de transformación, y hacer proyecciones de perspectiva de un espacio 3d en la pantalla. Estas funciones no son, y nunca serán, una librería 3d total (mi objetivo es dar rutinas de soporte genéricas, y no código gráfico muy especializado :-) pero estas funciones pueden serle útiles para desarrollar su propio código 3d.
Hay dos versiones de todas las funciones matemáticas de 3d: una usando aritmética de punto fijo, y la otra usando coma flotante. La sintaxis para ambas es idéntica, pero las funciones y estructuras de coma flotante tienen el sufijo '_f'. Ejemplo: la función cross_product() de punto fijo tiene el equivalente de coma flotante en cross_product_f(). Si está programando en C++, Allegro también sobrecarga estas funciones para que las use con la clase "fija".
La transformación 3d se realiza modelando una matriz. Esta es un array de 4x4 números que pueden ser multiplicados con un punto 3d para producir otro punto 3d. Si ponemos los valores correctos en la matriz, podemos usarla para varias operaciones como translación, rotación y escalado. El truco consiste en que puede multiplicar dos matrices para producir una tercera, y esta tendrá el mismo efecto en los puntos 3d que aplicando las dos matrices originales una después de la otra. Por ejemplo, si tiene una matriz que rota un punto, y otra que lo mueve en una dirección, puede combinarlas para producir una matriz que realizara la rotación y translación en un paso. Puede crear transformaciones extremadamente complejas de este modo, teniendo que multiplicar cada punto 3d por una sola matriz.

Allegro hace trampa al implementar la estructura de la matriz. La rotación y el escalado de un punto 3d puede ser realizado con una matriz simple de 3x3, pero para trasladar el punto y proyectarlo en la pantalla, la matriz tiene que ser extendida a 4x4, y el punto extendido a una cuarta dimensión, al añadir una coordenada extra: w=1. Esto es algo malo en términos de eficiencia, pero afortunadamente, es posible realizar una optimización. Dada la siguiente matriz 4x4:
(a, b, c, d)(e, f, g, h)(i, j, k, l)(m, n, o, p)
se puede observar un patrón de qué partes hacen qué. La rejilla 3x3 de arriba a la izquierda implementa la rotación y el escalado. Los tres valores de arriba de la cuarta columna (d, h y l) implementan la translación, y siempre y cuando la
matriz sea usada sólo para transformaciones afines, m, n y o serán siempre cero y p siempre será 1. Si no sabe que significa 'afín', lea a Foley & Van Damme: básicamente cubre el escalado, la translación y rotación del objeto pero no la proyección. Ya que Allegro usa una función aparte para la proyección, las funciones de matriz sólo tienen que servir para la transformación afín, lo que significa que no hay que guardar la fila inferior de la matriz. Allegro asume que esta contiene (0,0,0,1), y por eso optimiza las funciones de manipulación de matrices.
Las matrices se almacenan en estructuras:
typedef struct MATRIX - matriz de punto fijo{ fixed v[3][3]; - componente 3x3 de escalado y rotación fixed t[3]; - componente x/y/z de translación} MATRIX;typedef struct MATRIX_f matriz de coma flotante{ float v[3][3]; componente 3x3 de escalado y rotación float t[3];

- componente x/y/z de translación} MATRIX_f
identity_matrix
extern MATRIX identity_matrix;

extern MATRIX_f identity_matrix_f;
Variable global que contiene la matriz con identidad 'vacía'. Multiplicar por la matriz de identidad no tiene ningún efecto.
get_translation_matrix
void get_translation_matrix(MATRIX *m, fixed x, fixed y, fixed z);
void get_translation_matrix_f(MATRIX_f *m, float x, float y, float z);
Construye una matriz de translación, guardándola en m. Si se aplica a un punto (px, py, pz), esta matriz producirá el punto (px+x, py+y, pz+z). En otras palabras: mueve las cosas.
get_scaling_matrix
void get_scaling_matrix(MATRIX *m, fixed x, fixed y, fixed z);
void get_scaling_matrix_f(MATRIX_f *m, float x, float y, float z);

Construye una matriz de escalado, almacenándola en m. Cuando se aplica a un punto (px, py, pz), esta matriz produce un punto (px*x, py*y, pz*z). En otras palabras, agranda o empequeñece las cosas.
get_x_rotate_matrix
void get_x_rotate_matrix(MATRIX *m, fixed r);
void get_x_rotate_matrix_f(MATRIX_f *m, float r);
Construye las matrices de rotación del eje X, almacenándolas en m. Cuando se aplican a un punto, estas matrices lo rotarán sobre el eje X el ángulo especificado (en binario, 256 grados hacen un círculo).
get_y_rotate_matrix
void get_y_rotate_matrix(MATRIX *m, fixed r);

void get_y_rotate_matrix_f(MATRIX_f *m, float r);

Construye las matrices de rotación del eje Y, almacenándolas en m. Cuando se aplican a un punto, estas matrices lo rotarán sobre el eje Y el ángulo especificado (en binario, 256 grados hacen un círculo).
get_z_rotate_matrix
void get_z_rotate_matrix(MATRIX *m, fixed r);
void get_z_rotate_matrix_f(MATRIX_f *m, float r);
Construye las matrices de rotación del eje Z, almacenándolas en m. Cuando se aplican a un punto, estas matrices lo rotarán sobre el eje Z el ángulo especificado (en binario, 256 grados hacen un círculo).
get_rotation_matrix
void get_rotation_matrix(MATRIX *m, fixed x, fixed y, fixed z);
void get_rotation_matrix_f(MATRIX_f *m, float x, float y, float z);
Construye una matriz de transformación que rotará puntos en todos los ejes los grados especificados. (en binario, 256 grados hacen un círculo).
get_align_matrix
void get_align_matrix(MATRIX *m, fixed xfront, yfront, zfront, fixed xup, fixed yup, fixed zup);
Rota la matriz de tal forma que la alinea sobre las coordenadas de los vectores especificados (estos no tienen que ser normalizados o perpendiculares, pero up y front no pueden ser iguales). Un vector front de 1,0,0 y un vector up de 0,1,0 devolverán la matriz de identidad.
get_align_matrix_f
void get_align_matrix_f(MATRIX *m, float xfront, yfront, zfront, float xup, yup, zup);
Versión en coma flotante de get_align_matrix().
get_vector_rotation_matrix
void get_vector_rotation_matrix(MATRIX *m, fixed x, y, z, fixed a);
void get_vector_rotation_matrix_f(MATRIX_f *m, float x, y, z, float a);

Construye una matriz de transformación que rotará puntos sobre todos los vectores x,y,z un ángulo especificado (en binario, 256 grados hacen un círculo).
get_transformation_matrix
void get_transformation_matrix(MATRIX *m, fixed scale, fixed xrot, yrot, zrot, x, y, z);
Construye una matriz de transformación que rotará puntos en todos los ejes los ángulos especificados (en binario, 256 grados hacen un círculo), escalará el resultado (pasa el valor 1 si no quiere cambiar la escala), y entonces los trasladará a la posición x, y, z requerida.

get_transformation_matrix_f
void get_transformation_matrix_f(MATRIX_f *m, float scale, float xrot, yrot, zrot, x, y, z);
Versión en coma flotante de get_transformation_matrix().
get_camera_matrix
void get_camera_matrix(MATRIX *m, fixed x, y, z, xfront, yfront, zfront, fixed xup, yup, zup, fov, aspect);
Construye la matriz de cámara para trasladar objetos del espacio a una vista normalizada del espacio, preparada para la proyección de perspectiva. Los parámetros x, y, z especifican la posición de la cámara, xfront, yfront y zfront son los vectores 'de frente' que especifican hacia adonde apunta la cámara (estos pueden ser de cualquier tamaño, no es necesaria la normalización), y xup, yup y zup son los vectores de la dirección 'arriba'. El parámetro fov especifica el campo de visión (el ancho del foco de la cámara) en binario, haciendo 256 grados un círculo. Para proyecciones típicas, un campo de visión de entre 32 a 48 trabajara bien. Finalmente, la razón de aspecto es usada para el escalado en la dimensión Y relativamente al eje X, para que pueda ajustar las proporciones de la imagen final (ponga a uno para no escalar).
get_camera_matrix_f
void get_camera_matrix_f(MATRIX_f *m, float x, y, z, xfront, yfront,zfront, float xup, yup, zup, fov, aspect);
Versión en coma flotante de get_camera_matrix().
qtranslate_matrix
void qtranslate_matrix(MATRIX *m, fixed x, fixed y, fixed z); void qtranslate_matrix_f(MATRIX_f *m, float x, float y, float z);
Rutina optimizada para trasladar una matriz ya generada: esto simplemente añade el 'offset' de translación, por lo que no hay que crear dos matrices temporales y multiplicarlas.
qscale_matrix
void qscale_matrix(MATRIX *m, fixed scale);
void qscale_matrix_f(MATRIX_f *m, float scale);

Rutina optimizada para escalar una matriz ya generada: esto simplemente añade el factor de escalación, por lo que no hay que crear dos matrices temporales y multiplicarlas.
matrix_mul
void matrix_mul(const MATRIX *m1, MATRIX *m2, MATRIX *out); void matrix_mul_f(const MATRIX_f *m1, MATRIX_f *m2, MATRIX_f *out);
Multiplica dos matrices, guardando el resultado en out (que puede ser un duplicado de una de las dos mátrices de entrada, pero es más rápido cuando las entradas y la salida son todas distintas). La matriz resultante tendrá el mismo efecto que la combinación de m1 y m2, p.ej cuando son aplicadas en un

punto, (p * out) = ((p * m1) * m2). Cualquier número de transformaciones se puede concatenar de esta forma. Fíjese que la multiplicación de matrices no es communtaiva, así matrix_mul(m1,m2) != matrix_mul(m2,m1).
vector_length
fixed vector_length(fixed x, fixed y, fixed z);

float vector_length_f(float x, float y, float z);
Calcula la longitud del vector (x, y, z), usando ese buen teorema de Pitágoras.
normalize_vector
void normalize_vector(fixed *x, fixed *y, fixed *z);
void normalize_vector_f(float *x, float *y, float *z);
Convierte un vector (*x, *y, *z) a un vector normalizado. Este apunta en la misma dirección que el vector original, pero tiene una longitud de uno.
dot_product
fixed dot_product(fixed x1, y1, z1, x2, y2, z2);

float dot_product_f(float x1, y1, z1, x2, y2, z2);
Calcula el producto escalar (x1, y1, z1) . (x2, y2, z2), devolviendo el resultado.
cross_product
void cross_product(fixed x1, y1, z1, x2, y2, z2, *xout, *yout, *zout);
void cross_product_f(float x1, y1, z1, x2, y2, z2, *xout, *yout, *zout);
Calcula el producto vectorial (x1, y1, z1) x (x2, y2, z2), almacenando el resultado en (*xout, *yout, *zout). El resultado es perpendicular a los dos vectores de entrada, para que pueda ser usado para generar las normales de los polígonos.
polygon_z_normal
fixed polygon_z_normal(const V3D *v1, V3D *v2, V3D *v3); float polygon_z_normal_f(const V3D_f *v1, V3D_f *v2, V3D_f *v3);
Encuentra la componente Z de la normal de un vector de tres vértices especificados (que deben ser parte de un polígono convexo). Esto es usado principalmente en la ocultación de caras. Las caras traseras de un poliedro

cerrado nunca son visibles al espectador, y por tanto no necesitan ser dibujadas. Esto puede ocultar aproximadamente la mitad de los polígonos de una escena. Si la normal es negativa, el polígono se puede eliminar, si es cero, el polígono está perpendicular a la pantalla.

apply_matrix

void apply_matrix(const MATRIX *m, fixed x, y, z, *xout, *yout, *zout);

void apply_matrix_f(const MATRIX_f *m, float x, y, z, *xout, *yout, *zout);

Multiplica el punto (x, y, z) por la transformación de la matriz m, almacenando el resultado en el punto (*xout, *yout, *zout).

set_projection_viewport

void set_projection_viewport(int x, int y, int w, int h);

Ajusta el punto de visión usado para escalar la salida de la función persp_project(). Pase las dimensiones de la pantalla y el área donde la quiere dibujar, que típicamente será 0, 0, SCREEN_W, SCREEN_H.

persp_project

void persp_project(fixed x, y, z, *xout, *yout);

void persp_project_f(float x, y, z, *xout, *yout);

Proyecta el punto 3d (x, y, z) del espacio sobre una pantalla 2d, almacenando el resultado en (*xout, *yout) usando los parámetros anteriormente ajustados por set_projection_viewport(). Esta función proyecta desde la pirámide de vista normalizada, que tiene una cámara en el origen apuntando al eje z positivo. El eje x va de izquierda a derecha, y va de arriba a abajo, y z se incrementa con la profundidad de la pantalla. La cámara tiene un ángulo de visión de 90 grados, es decir, los planos x=z y -x=z serán los bordes izquierdo y derecho de la pantalla, y los planos y=z y -y=z serán la parte superior e inferior de la pantalla. Si quiere un campo de visión diferente a la posición de la cámara, debería transformar todos sus objetos con la matriz de visión apropiada. Ejemplo, para obtener el efecto de haber girado la cámara 10 grados a la izquierda, rote todos sus objetos 10 grados a la derecha.

RUTINAS MATEMÁTICAS PARA USAR CUATERNIONES

Los cuaterniones son una forma alternativa de representar la parte de rotación de una transformación, y pueden ser más fáciles de manipular que las matrices. Como con una matriz, usted puede codificar transformaciones geométricas en una, concatenar varias de ellas para mezclar múltiples transformaciones, y aplicarlas a un vector, pero sólo pueden almacenar rotaciones puras. La gran ventaja es que puede interpolar precisamente entre dos cuaterniones para obtener una rotación parcial, evitando los enormes problemas de la interpolación más convencional con ángulos eulerianos.

Los cuaterniones sólo poseen versiones de punto flotante, sin ningún sufijo "_f". Por otro lado, la mayoría de las funciones de cueterniones se corresponden con una función matricial que realiza una operación similar.

Cuaternión significa 'de cuatro partes', y es exactamente eso. Aquí está la estructura:

typedef struct QUAT{ float w, x, y, z;}

Usted se divertirá mucho buscando el significado real de estos números, pero eso está más allá del alcance de esta documentación. Los cuaterniones funcionan - créame.

identity_quat

extern QUAT identity_quat;

Variable global que contiene el cuaternión identidad 'que no hace nada'. Multiplicar por el cuaternión identidad no tiene efecto alguno.

get_x_rotate_quat

void get_x_rotate_quat(QUAT *q, float r);
void get_y_rotate_quat(QUAT *q, float r);
void get_z_rotate_quat(QUAT *q, float r);

Construye cuaterniones de ejes de rotación, almacenándolos en q. Cuando sean aplicados a un punto, éstos cuaterniones lo rotarán sobre el eje relevante el ángulo especificado (dado en binario, 256 grados forman un círculo).

get_rotation_quat

void get_rotation_quat(QUAT *q, float x, float y, float z);

Construye un cuaternión que rotará puntos alrededor de los tres ejes las cantidades especificadas (dadas en binario, 256 grados forman un círculo).

get_vector_rotation_quat

void get_vector_rotation_quat(QUAT *q, float x, y, z, float a);

Construye un cuaternión que rotará puntos alrededor del vector x,y,z el ángulo especificado (dado en binario, 256 grados forman un círculo).

quat_to_matrix

void quat_to_matrix(const QUAT *q, MATRIX_f *m);

Construye una matriz de rotación a partir de un cuaternión. matrix_to_quat

void matrix_to_quat(const MATRIX_f *m, QUAT *q);

Construye un cuaternión a partir de una matriz de rotación. La translación es descartada durante la conversión. Use get_align_matrix_f() si la matriz no está ortonormalizada, porque de otra forma podrían pasar cosas extrañas.

quat_mul

void quat_mul(const QUAT *p, const QUAT *q, QUAT *out);

Multiplica dos cuaterniones, almacenando el resultado en out. El cuaternión resultante tendrá el mismo efecto que la combinación de p y q, es decir, cuando es aplicado a un punto, (punto * out) = ((punto * p) * q). Cualquier cantidad de rotaciones pueden ser concatenadas de ésta manera. Note que la multiplicación del cuaternión no es conmutativa, es decir que quat_mul(p, q) != quat_mul(q, p).

apply_quat

void apply_quat(const QUAT *q, float x, y, z, *xout, *yout, *zout);

Multiplica el punto (x, y, z) por el cuaternión q, almacenando el resultado en (*xout, *yout, *zout). Esto es un
poquito más lento que apply_matrix_f(), así que úselo para trasladar unos pocos puntos. Si usted tiene muchos puntos, es mucho más eficiente llamar a quat_to_matrix() y entonces usar apply_matrix_f().

quat_interpolate
void quat_interpolate(const QUAT *from, QUAT *to, float t, QUAT *out);
Construye un cuaternión que representa una rotación entre from y to. El argumento t puede ser cualquiera entre 0 y 1, y representa dónde estará el resultado entre from y to. 0 devuelve from, 1 devuelve to, y 0.5 devolverá una rotación exactamente en la mitad. El resultado es copiado a out. Esta función creará una rotación corta (menos de 180 grados) entre from y to.
quat_slerp
void quat_slerp(const QUAT *from, QUAT *to, float t, QUAT *out, int how);
Igual que quat_interpolate(), pero permite más control sobre cómo es hecha la rotación. El parámetro how puede ser alguno de estos valores:
QUAT_SHORT - como quat_interpolate(), usa el camino más corto
QUAT_LONG - la rotación será mayor que 180 grados
QUAT_CW - rotación horaria vista desde arriba
QUAT_CCW - rotación antihoraria vista desde arriba QUAT_USER - los cuaterniones son interpolados exactamente como son dados.
RUTINAS GUI
Allegro posee un gestor de diálogos orientados a objetos que originalmente se basa en el sistema GEM del Atari (form_do(), objc_draw(), etc: programadores veteranos del ST saben de lo que estoy hablando :-) Puede usar el GUI tal y como esta para crear interfaces simples para cosas como el programa test y la utilidad grabber, o puede usarlo como base para sistemas más complicados que cree. Allegro le deja definir sus propios tipos de objetos y escribir nuevos procesos de diálogo, por lo que tendrá control total sobre los aspectos visuales de la interfaz mientras todavía usa Allegro para controlar el ratón, teclado, joystick, etc.
Un diálogo GUI se almacena como un array de objetos DIALOG, de los cuales cada uno cuenta con los siguientes parámetros:
typedef struct DIALOG{ int (*proc)(int, DIALOG *, int); -
proceso del diálogo (controlador de
mensajes) int x, y, w, h; - posición y tamaño del objeto int fg, bg; colores de letra y fondo int key; atajo ASCII del teclado int flags; variable con el estado del objeto int d1, d2;
- úselos para lo que quiera void *dp, *dp2, *dp3;

- punteros a datos específicos del objeto} DIALOG;
El array debe acabar con un objeto que tiene el proceso de diálogo puesto a NULL.

El campo de bits puede contener cualquier combinación de los siguientes bits:
D_EXIT - este objeto debe cerrar el diálogo al activarseD_SELECTED - este objeto está seleccionadoD_GOTFOCUS - este objeto tiene el foco de entradaD_GOTMOUSE - el ratón esta actualmente encima del

objetoD_HIDDEN - este objeto está oculto e inactivoD_DISABLED - este objeto está de color gris e inactivoD_DIRTY - este objeto necesita ser redibujadoD_INTERNAL - ino use esto! Es para uso interno de la biblioteca...D_USER - cualquier valor múltiplo de dos mayor que éste esta libre para que lo use

Cada objeto es controlado por un proceso de diálogo, que se almacena en el puntero proc. Este será llamado por el gestor de diálogos cuando se produzca alguna acción que involucre al objeto, o puede llamarlo directamente con la función object_message(). El proceso de diálogo debe seguir la siguiente forma:

int foo(int msg, DIALOG *d, int c);

Se le pasará una variable (msg) indicando qué debe hacer, un puntero al objeto concerniente (d), y si msg es MSG_CHAR o MSG_XCHAR, la tecla que ha sido pulsada (c). Fíjese que d es un puntero a un objeto especifico y no al diálogo entero.

El proceso del diálogo debería devolver uno de los siguientes valores:

D_O_K - estado normal de retornoD_CLOSE - le dice al gestor de diálogos que cierre el diálogoD_REDRAW

- le dice al gestor de diálogos que redibuje el diálogoD_REDRAWME - dice al gestor de diálogos que redibuje el objeto actualD_WANTFOCUS - requiere que se le de el foco de entrada al objetoD_USED_CHAR - MSG_CHAR y MSG_XCHAR devuelven esto si se uso una tecla

Los procesos de diálogo se pueden llamar con cualquiera de estos mensajes:

MSG_START:

Le dice al objeto que se inicialice. El gestor de diálogo manda esto a todos los objetos del diálogo justo antes de mostrarlo.

MSG_END:

Se manda a todos los objetos cuando se cierra un diálogo, permitiéndoles que hagan los procesos de limpieza que necesiten.

MSG_DRAW:

Le dice al objeto que se dibuje en pantalla. El puntero del ratón se desactivara cuando este mensaje sea mandado, para que el código de dibujado no se preocupe por él.

MSG_CLICK:

Informa al objeto que un botón del ratón a sido pulsado mientras el ratón estaba encima del objeto. Normalmente el objeto realizará su propio seguimiento del ratón mientras el botón esté pulsado, y sólo volverá de este controlador de mensaje cuando se suelte el botón.

MSG_DCLICK:

Se manda cuando el usuario hace un doble click en un objeto.

Primero se manda el mensaje MSG_CLICK cuando se presiona el botón por primera vez, y entonces MSG_DCLICK si se suelta y vuelve a presionar otra vez en un corto periodo de tiempo.
MSG_KEY:
Mandado cuando el atajo de teclado del objeto se presiona, o si se pulsa enter, espacio, o algún botón del joystick cuando el objeto tiene el foco de entrada.
MSG_CHAR:
Cuando se pulsa una tecla, este mensaje es mandado al objeto que tiene el foco de entrada, con un formato de código de carácter igual al de la función readkey() (valor ASCII en el byte bajo, scancode en el byte alto) como parámetro c. Si el objeto trata con la pulsación de teclas debería devolver D_USED_CHAR, en caso contrario debería devolver D_O_K para permitir operar al interfaz de teclado por defecto. Si necesita acceder a la entrada de un carácer Unicode, debería usar MSG_UCHAR en vez de MSG_CHAR.
MSG_UCHAR:
Si un objeto ignora la entrada MSG_CHAR, este mensaje será mandado inmediatemente después de él, pasando el valor completo de la tecla en Unicode como parámetro c. Esto le permite leer códigos de carácteres mayores que 255, pero no puede decirle nada sobre el scancode: si necesita saberlo, use MSG_CHAR en lugar de MSG_UCHAR. Este manejador debe devolver D_USED_CHAR si procesó la entrada, o D_O_K en otro caso.

MSG_XCHAR:
Cuando se pulsa una tecla, Allegro mandará MSG_CHAR y MSG_UCHAR al objeto que tenga el foco de entrada. Si este objeto no procesa la tecla (ej. devuelve D_O_K en vez de D_USED_CHAR), el gestor de diálogos buscará un objeto con un atajo de teclado asociado en el campo key, mandándole un MSG_KEY. Si esto falla, mandará un MSG_XCHAR al resto de los objetos del diálogo, permitiéndoles responder a pulsaciones de tecla especiales incluso cuando no tienen el foco de entrada. Normalmente debería ignorar este mensaje (devolver D_O_K en vez de D_USED_CHAR), en cuyo caso Allegro realizará las acciones por defecto tales como mover el foco de entrada en respuesta a los cursores y cerrar el diálogo si se pulsa ESC.
MSG_WANTFOCUS:
Determina si un objeto quiere recibir el foco de entrada. Deberá devolver D_WANTFOCUS si lo desea, o D_O_K si no esta interesado en recibir datos del usuario.
MSG_GOTFOCUS:
MSG_LOSTFOCUS:
Es mandado cuando un objeto gana o pierde el foco de entrada. Estos mensajes siempre son seguidos por un MSG_DRAW, para dejar que los objetos se redibujen de manera diferente. Si devuelve D_WANTFOCUS en respuesta a un evento MSG_LOSTFOCUS, esto le permitirá a su objeto quedarse con el foco de entrada incluso si el ratón deja de estar sobre él y se pone sobre el

fondo u otro objeto inerte, por lo que solamente perderá el foco de entrada cuando otro objeto esté preparado para cogerlo (este truco es usado por el objeto d_edit_proc()).
MSG_GOTMOUSE:
MSG_LOSTMOUSE:
Es mandado cuando el ratón se pone o quita de un objeto. A diferencia de los mensajes de foco, a estos no les sigue un
MSG_DRAW, por lo que si el objeto se dibuja de forma diferente cuando el ratón esta encima suyo, es responsable de redibujarse él solo como respuesta a estos mensajes.
MSG_IDLE:
Es mandado cuando el diálogo de gestor no tiene nada mejor que hacer.
MSG_RADIO:
Es mandado por botones de radio para deseleccionar otros botones en el mismo grupo cuando son pulsados. El número del grupo se pasa en el parámetro del mensaje c.
MSG_WHEEL:
Enviado al objeto que tenga el foco de entrada cada vez que se mueve la rueda del ratón. El parámetro c de mensaje contiene el número de clicks.
MSG_LPRESS, MSG_MPRESS, MSG_RPRESS:
Enviado cuando el botón correspondiente del ratón es presionado.
MSG_LRELEASE, MSG_MRELEASE, MSG_RRELEASE:
Enviado cuando el botón correspondiente del ratón es soltado.
MSG_USER:
El primer valor de mensaje libre. Cualquier número a partir de aquí (MSG_USER, MSG_USER+1, MSG_USER+2, ... MSG_USER+n) es libre para lo que quiera.
Allegro trae varios procesos de diálogo estándar. Puede usarlos tal como vienen para crear una interfaz de objetos simples, o puede usarlos desde sus propios proceses de diálogo, resultando en una especie de herencia de objetos. Por ejemplo, podría hacer un objeto que llama d_button_proc para dibujarse, pero controla el mensaje de pulsación del botón del ratón de forma diferente, o un objeto que llama d_button_proc para cualquier cosa excepto para dibujarse a si mismo, por lo que se comportaría como un botón normal pero podría ser completamente diferente.
Desde la versión 3.9.33 (CVS) de Allegro, algunos objetos y menús de la interfaz gráfica de usuario son dibujados de forma diferente en comparación con versiones anteriores de Allegro. Los cambios son:

- Las sombras bajo d_shadow_box_proc y d_button_proc son siempre negras.
- El cambio más importante (y visible inmediatamente), es que algunos objetos se dibujan más pequeños. La diferencia es exáctamente un pixel tanto vertical como horizontalmente, comparando con versiones anteriores. La razón es que en versiones

anteriores, estos objetos eran demasiado grandes para la pantalla - su tamaño era d->w+1 y d->h+1 en pixels (y no d->w y d->h, como debería ser). Este cambio afecta a los siguientes objetos: d_box_proc,d_shadow_box_proc,d_button_proc,d_check_proc,d_rad io_proc,d_list_proc,d_text_list_proc andd_textbox_proc.
Cuando quiera convertir diálogos antiguos para que visualmente sean iguales al compilar con la nueva versión de Allegro, simplemente incremente el tamaño un pixel en los campos de anchura y altura.

· Cuando un item del menú (no en un menú de barra) tiene un menú hijo, hay una pequeña flecha junto al nombre del menú hijo, apuntando a la derecha - para que los usuarios vean que ese menú tiene submenús - y no es necesario usar nombres de menú como por ejemplo "Nuevo...", para indicar que el elemento tiene un submenú. El submenú será dibujado a la derecha del padre, intentando no sobreescribirlo.

d_clear_proc
int d_clear_proc(int msg, DIALOG *d, int c);
Esto simplemente borra la pantalla al ser dibujado. Util como el primer objeto de un diálogo.

d_box_proc
int d_box_proc(int msg, DIALOG *d, int c);
int d_shadow_box_proc(int msg, DIALOG *d, int c);
Estos dibujan cajas en la pantalla, con o sin sombra. d_bitmap_proc
int d_bitmap_proc(int msg, DIALOG *d, int c);
Esto dibuja un bitmap en la pantalla, que deberá ser apuntado por el campo dp.

d_text_proc
int d_text_proc(int msg, DIALOG *d, int c);
int d_ctext_proc(int msg, DIALOG *d, int c);
int d_rtext_proc(int msg, DIALOG *d, int c);
Estos dibujan texto en la pantalla. El campo dp debe apuntar a la cadena de texto a visualizar. d_ctext_proc() centra la cadena alrededor de la coordenada x, y d_rtext_proc la alinea a la derecha. Todos los carácteres '&' de la cadena serán reemplazados por líneas debajo del siguiente carácter, para enseñar los atajos de teclado (tal y como en MS Windows). Para enseñar el caracter normal, ponga "&&". Para dibujar el texto con otra cosa que no sea la fuente por defecto, ajuste el campo dp2 para que apunte a una fuente propia.

d_button_proc
int d_button_proc(int msg, DIALOG *d, int c);
Un botón objeto (el campo dp apunta a una cadena de texto). Este objeto puede ser seleccionado pinchando sobre él con el ratón o presionando su atajo de teclado. Si se pone el bit D_EXIT, el seleccionarlo provocará el cierre del diálogo. Si no, encenderá y apagará el diálogo. Como en d_text_proc(), puede usar el caracter '&' para enseñar el atajo de teclado del botón.

d_check_proc
int d_check_proc(int msg, DIALOG *d, int c);
Este es un ejemplo de cómo puede derivar objetos desde otros objetos. La mayoría de la funcionalidad viene de d_button_proc(), pero se visualiza como un recuadro a marcar.
Si el campo d1 no es cero, el texto se imprimirá a la derecha de la marca, en caso contrario estará a la izquierda.
Nota: el ancho del objeto debe reservar espacio tanto para el texto como para el recuadro (que es cuadrado, con lados del mismo tamaño que la altura del objeto).
d_radio_proc
int d_radio_proc(int msg, DIALOG *d, int c);
Un objeto de botón de radio. Un diálogo puede contener cualquier número de grupos de botones de radio: el seleccionar un botón de radio provoca que los otros botones del mismo grupo se desactiven. El campo dp apunta a una cadena de texto, d1 requiere el número de grupo especifico, y d2 es el estilo del botón (0=círculo, 1=cuadrado).
d_icon_proc
int d_icon_proc(int msg, DIALOG *d, int c);
Un botón bitmap. El color fg es usado por la línea de puntos que enseña el foco, y el color bg hace la sombra que rellena la parte superior e izquierda del botón cuando este se presiona. d1 es la "profundidad", es decir, el número de pixels que el icono será movido abajo a la derecha cuando se seleccione (por defecto 2) si no hay imagen "seleccionada". d2 es la distancia entre los puntos de la línea del foco. dp apunta a un bitmap para el icono, mientras que dp2 y dp3 son las imágenes de botón seleccionado y no seleccionado (esto es opcional, puede ser NULL).
d_keyboard_proc
int d_keyboard_proc(int msg, DIALOG *d, int c);
Este es un objeto invisible para implementar atajos de teclado. Puede poner un código ASCII en el campo de la tecla del diálogo del objeto (un caracter como 'a' responde a la pulsación de tecla, un número del 1 al 26 responde a Ctrl+una tecla a-z), o puede poner un scancode del teclado en el campo d1 y/o d2. Cuando una de estas teclas es presionada, el objeto llamará a la función apuntada por dp. Esto debería devolver un entero, el cual se pasará de vuelta al gestor de diálogo, para que pueda devolver D_O_K, D_REDRAW, D_CLOSE, etc.
d_edit_proc
int d_edit_proc(int msg, DIALOG *d, int c);
Un objeto de texto editable (el campo dp apunta a la cadena de texto). Cuando tiene el foco de entrada (obtenido al pinchar sobre el objeto con el ratón), se puede teclear texto en el objeto. El campo d1 indica el número máximo de carácteres que se aceptaran, y d2 es la posición del cursor dentro de la cadena de texto.

d_list_proc
int d_list_proc(int msg, DIALOG *d, int c);
Un objeto caja de lista. Esto permite al usuario ir hacia arriba o abajo de la lista de ítems y seleccionar uno pinchando con el ratón o usando las teclas. Si el bit D_EXIT
está activado, haciendo doble click en la lista de ítems cerrará el diálogo. El índice del objeto seleccionado se encuentra en el campo d1, y d2 es usado para indicar hasta dónde se ha desplazado la lista de ítems. El campo dp apunta a una función que será llamada para obtener información sobre los contenidos de la lista. Esto debería seguir el modelo: char *foobar(int index, int *list_size);
Si index es cero o positivo, la función debe devolver un puntero a la cadena que debe enseñarse en la posición index en la lista. Si index es negativo, debe devolver NULL y en list_size debe poner el número de ítems de la lista.
Para crear una lista de selección múltiple, haga apuntar dp2 a un array de variables de un byte que indican el estado de selección de cada ítem de la lista (distinto de cero para entradas seleccionadas). ¡Esta tabla debe ser al menos tan grande como el número de objetos de la lista!
d_text_list_proc
int d_text_list_proc(int msg, DIALOG *d, int c);
Igual que d_list_proc, pero permite que el usuario escriba algunos de los primeros carácteres de una entrada de la caja de listas para así seleccionarlo. Se usa dp3 internamente, así que no deberá poner nada importante ahí por sí mismo.
d_textbox_proc
int d_textbox_proc(int msg, DIALOG *d, int c);
Un objeto caja de texto. El campo dp apunta al texto que debe enseñarse en la caja. Si el texto es largo, habrá una barra de desplazamiento vertical a la derecha del objeto que podrá ser usada para mover el texto dentro de la caja. El valor por defecto es imprimir el texto con ajuste de anchura a nivel de palabra, pero si se activa el bit D_SELECTED, el texto se imprimirá con ajuste de anchura a nivel de carácter. El campo d1 se usa internamente para guardar el número de líneas de texto, y d2 es usado para guardar hasta dónde se ha desplazado el texto.
d_slider_proc
int d_slider_proc(int msg, DIALOG *d, int c);
Una barra de desplazamiento. Este objeto tiene un valor en d2, con rango de 0 a d1. Enseñará una barra de desplazamiento vertical si h es más grande o igual a w, de otro modo enseñará una barra horizontal. El campo dp puede contener un bitmap opcional que usará la barra de desplazamiento, y dp2 puede contener una función opcional de callback, que será llamada cada vez que d2 cambie. La función callback debería seguir el prototipo:
int function(void *dp3, int d2);
El objeto d_slider_proc devolverá el valor de la función callback.

d_menu_proc
int d_menu_proc(int msg, DIALOG *d, int c);
Este objeto es una barra de menú que abre menús hijos cuando se pincha en él o cuando alguna combinación alt+tecla es
pulsada y se corresponde con algún atajo del menú. El objeto ignora muchos de los campos de la estructura de diálogo, particularmente el color se coge de las variables gui_*_color, y el ancho y alto se calculan automáticamente. El campo dp apunta a un array de estructuras de menú: mire do_menu() para más información. El nivel de arriba del menú será visualizado como una barra horizontal, pero cuando aparezcan los menús hijos, aparecerán con el formato vertical usual usado por do_menu(). Cuando un ítem del menú es seleccionado, el valor de retorno de la función del menú se pasa al gestor de diálogo, para que las funciones de sus menús puedan devolver D_O_K, D_REDRAW, o D_CLOSE.
d_yield_proc
int d_yield_proc(int msg, DIALOG *d, int c);
Un objeto ayudante invisible que rechaza rebanadas de tiempo de la CPU (si el sistema lo soporta) cuando la interfaz no tiene nada más que hacer que esperar las acciones del usuario.
gui_mouse_focus
extern int gui_mouse_focus;
Si esta activado, el foco de entrada sigue al puntero del ratón, de otro modo, un click es requerido para mover el foco de entrada.
gui_fg_color
extern int gui_fg_color, gui_bg_color;
Los colores de primer plano y fondo de los diálogos estándar (alertas, menús, y el selector de ficheros). Por defecto son 255 y 0.
gui_mg_color
extern int gui_mg_color;
El color usado para enseñar los diálogos en gris (los que tienen el bit D_DISABLED activado). Por defecto es 8.
gui_font_baseline
extern int gui_font_baseline;
Si se pone a un valor distinto de cero, ajusta los subrayados mostrados por los atajos de teclado para que igualen el tamaño de las letras de la fuente que sobresalgan por debajo.
gui_mouse_x
extern int (*gui_mouse_x)();

extern int (*gui_mouse_y)(); extern int (*gui_mouse_z)(); extern int (*gui_mouse_b)();
Funciones de enganche, usadas por las rutinas GUI siempre que necesiten acceder al estado del ratón. Por defecto éstas devuelven copias de las variables mouse_x, mouse_y, mouse_z y mouse_b, pero pueden ser usadas

para situar o escalar la posición del ratón, o para leer datos de una fuente totalmente diferente.

gui font
Puede cambiar el puntero global a 'font' para hacer que los objetos del GUI usen otra cosa que la fuente estándar 8x8. Los procesos estándar de diálogo, los menús, las cajas de alerta, trabajarán con fuentes de cualquier tamaño, pero el diálogo gfx_mode_select() aparecerán mal con cualquier cosa que no sean fuentes de 8x8.
gui_textout
int gui_textout(BITMAP *bmp, const char *s, int x, y, color, centre);
Función de ayuda usada por las rutinas GUI. Dibuja una cadena de texto en la pantalla, interpretando el caracter '&' como el subrayado para enseñar los atajos de teclado. Devuelve el ancho de la cadena en pixels.
gui_strlen
int gui_strlen(const char *s);
Función de ayuda usada por las rutinas GUI. Devuelve la longitud de una cadena de texto en pixels, ignorando los carácteres '&'.
position_dialog
void position_dialog(DIALOG *dialog, int x, int y);
Mueve un array de objetos de diálogo a la posición de pantalla especificada (indicando la esquina superior izquierda del diálogo).
centre_dialog
void centre_dialog(DIALOG *dialog);
Mueve un array de diálogos de objetos para que estén centrados en la pantalla.
set_dialog_color
void set_dialog_color(DIALOG *dialog, int fg, int bg);
Pone el color de primer plano y fondo de un array de diálogo de objetos.
find_dialog_focus
int find_dialog_focus(DIALOG *dialog);
Busca el diálogo para el objeto que tiene el foco de entrada, devolviendo un índice o -1 si no hay foco de entrada. Esto es útil cuando está llamando a do_dialog() varias veces seguidas y quiere dejar el foco de entrada en el mismo lugar que cuando se enseño el diálogo la ultima vez, por lo que pueda llamar a do_dialog(dlg, find_dialog_focus(dlg));
offer_focus
int offer_focus(DIALOG *d, int obj, int *focus_obj, int force);
Ofrece el foco de entrada a un objeto particular. Normalmente esta función envía el mensaje MSG_WANTFOCUS para preguntar si el objeto desea aceptar el foco. Pero si pasa un valor distinto de cero como argumento force, estrá indicando a la función que el objeto debe tener el foco de entrada.
object_message
int object_message(DIALOG *dialog, int msg, int c);

Envía un mensaje a un objeto y devuelve la respuesta generada. Recuerde que el primer parámetro es el objeto del diálogo (no el array completo) al que desea enviar el mensaje. Por ejemplo, para hacer que el segundo objeto del diálogo se dibuje, podría escribir: object_message(&dialog[1], MSG_DRAW, 0);
dialog_message
int dialog_message(DIALOG *dialog, int msg, int c, int *obj);
Manda un mensaje a todos los objetos de un array. Si alguno de los procesos de diálogo devuelve otro valor que no sea D_O_K, la función devuelve el valor y hace apuntar a obj al índice del objeto que produjo ese mensaje.
broadcast_dialog_message
int broadcast_dialog_message(int msg, int c);
Manda un mensaje a todos los objetos del diálogo activo. Si cualquiera de los procesos de diálogo devuelve otros valores que no sean D_O_K, devuelve ese valor.
do_dialog
int do_dialog(DIALOG *dialog, int focus_obj);
La función básica del gestor de diálogo. Esta enseña el diálogo (un array de objetos de diálogo, acabados por uno con el proceso de diálogo puesto a NULL), y pone el foco de entrada a focus_obj (-1 si no quiere que nada tenga el foco de entrada). La función interpreta la entrada del usuario y despacha mensajes a medida que se requiera, hasta que uno de los procesos de diálogo le dice que lo cierre. Entonces devuelve el índice del objeto que causó el cierre.
popup_dialog
int popup_dialog(DIALOG *dialog, int focus_obj);
Como do_dialog(), pero almacena los datos de la pantalla antes de dibujar el diálogo y los recupera cuando el diálogo es cerrado. El área de pantalla a guardar es calculada según las dimensiones del primero objeto en el diálogo, por lo que el resto de los objetos deben estar dentro de él.
init_dialog
DIALOG_PLAYER *init_dialog(DIALOG *dialog, int focus_obj);
Esta función da acceso de bajo nivel a la misma funcionalidad que do_dialog(), pero le permite combinar la caja de diálogo con sus propias estructuras de objeto. Inicializa un diálogo, devolviendo un puntero al objeto reproductor que puede ser usado con update_dialog() y shutdown_dialog(). Con estas funciones, puede implementar su propia versión de do_dialog() con las líneas:
DIALOG_PLAYER *player = init_dialog(dialog, focus_obj);while (update_dialog(player)) ;return shutdown_dialog(player);
update_dialog
int update_dialog(DIALOG_PLAYER *player);
Actualiza el estado de un diálogo de objeto devuelto por init_dialog(). Devuelve TRUE si el diálogo sigue activo, o FALSE si se ha cerrado. Si devuelve FALSE, depende de usted llamar a shutdown_dialog() o continuar

con la ejecución. El objeto que requirió la salida puede ser determinado desde el campo player->obj.
shutdown_dialog
int shutdown_dialog(DIALOG_PLAYER *player);
Destruye el player de diálogo de objeto devuelto por init_dialog(), devolviendo el objeto que causó la salida (esto es lo mismo que el valor de do_dialog()).
active_dialog
extern DIALOG *active_dialog;
Puntero global al diálogo activado más recientemente. Esto puede ser útil si un objeto necesita recorrer la lista de diálogos "hermanos".
gui menus
Los menús emergentes o desplegables son creados como un array de la estructura:
typedef struct MENU{ char *text; - texto a visualizar por el ítem del menú int (*proc)();
- llamado cuando el ítem del menú es
seleccionado struct MENU *child; - menú hijo anidado int flags;
- estado seleccionado o deseleccionado void *dp; - puntero a datos que necesite} MENU;
Cada ítem del menú contiene una cadena de texto. Puede usar el caracter '&' para indicar el atajo del teclado, o puede ser una cadena de texto de tamaño cero para visualizar el ítem como una barra divisoria no seleccionable. Si la cadena contiene un caracter de tabulación "\t", el texto que sigue será justificado a la derecha, por ejemplo para enseñar información sobre el atajo del teclado. El puntero proc es una función que será llamada cuando el ítem del menú sea seleccionado, y child apunta a otro menú, permitiéndole hacer menús anidados. proc y child pueden ser ambos NULL. La función proc devuelve un entero que es ignorado si el menú fue llamado por do_menu(), pero que es devuelto al gestor de diálogo si fue creado por un objeto d_menu_proc(). El array de ítems del menú se cierra con una entrada con el campo text puesto a NULL.
Los ítems del menú pueden ser deseleccionados (en gris) activando el bit D_DISABLED en el campo flags, y pueden enseñar un símbolo de marcado si se activa el bit D_SELECTED. Con la alineación y fuente por defecto, la marca sobreescribiría el texto del menú, por lo que si va a usar ítems de menú con símbolo de marca, seria una buena idea
prefijar todas sus opciones con un caracter de espacio o dos, para estar seguro de que hay suficiente sitio para el símbolo de marcado.
do_menu
int do_menu(MENU *menu, int x, int y)
Enseña y anima el menú emergente en la pantalla en las coordenadas especificadas (estas serán ajustadas si el menú no entra enteramente en la pantalla). Devuelve el índice del ítem de menú seleccionado, o -1 si el menú fue cancelado. Fíjese que el valor de retorno no puede indicar una selección de

menús hijo, por lo que tendrá que usar funciones "callback" si quiere menús multi-nivel.
active_menu
extern MENU *active_menu;
Cuando se activa una llamada de vuelta, esta variable será puesta al valor del ítem seleccionado, para que su rutina pueda determinar desde dónde fue llamada.
gui_menu_draw_menu
extern void (*gui_menu_draw_menu)(int x, int y, int w, int h);
extern void (*gui_menu_draw_menu_item)(MENU *m, int x, int y, int w, int h, int bar, int sel);
Si las ajusta, estas funciones serán llamadas cuando un menú necesite ser dibujado, por lo que puede cambiar el aspecto de los menús.
A gui_menu_draw_menu() se le pasará la posición y tamaño del menú. Debería dibujar el fondo del menú en la pantalla (screen).
gui_menu_draw_menu_item() será llamado una vez por cada elemento del menú que debe ser dibujado. bar será distinto de cero si el elemento es parte de una barra de menú horizontal del nivel superior, y sel será distinto de cero si el elemento del menú está seleccionado. También debería ser dibujado en la pantalla (screen).
alert
int alert(const char *s1, *s2, *s3, const char *b1, *b2, int c1, c2);
Enseña una caja de alerta emergente, conteniendo tres líneas de texto (s1-s3), y con uno o dos botones. El texto de los botones se pasa en b1 y b2 (b2 puede ser NULL), y los atajos de teclado se pasan en c1 y c2. Devuelve 1 o 2 dependiendo de que botón fue pulsado. Si la alerta se aborta pulsando ESC cuando ESC no es uno de los atajos del teclado, se trata como si se hubiese pulsado el segundo botón (esto es consistente con la típica alerta "Ok", "Cancelar").
alert3
int alert3(const char *s1, *s2, *s3, const char *b1, *b2, *b3, int c1, c2, c3);
Como alert(), pero con tres botones. Devuelve 1, 2, o 3.

file_select
int file_select(const char *message, char *path, const char *ext);
Deprecado. Use file_select_ex() en su lugar, pasando las dos constantes OLD_FILESEL_WIDTH y OLD_FILESEL_HEIGHT si desea que el selector de ficheros sea visualizado con las dimensiones del antiguo selector.
file_select_ex
int file_select_ex(const char *message, char *path, const char *ext, int size, int w, int h);
Visualiza el selector de ficheros de Allegro, con la cadena message como título. El parámetro path contiene el nombre del fichero inicial a visualizar (esto puede ser usado como el comienzo del directorio, o para dar un nombre por

defecto a un salvar-como). La selección del usuario se devuelve alterando el buffer path, cuya longitud en bytes está indicada por el parámetro size. Recuerde que debería tener como mínimo espacio para 80 carácteres (no bytes). La lista de ficheros es filtrada deacuerdo con las extensiones de ficheros en ext. Pasando NULL incluye todos los ficheros, "PCX;BMP" incluye solamente los ficheros con extensiones .PCX o .BMP. Para controlar ficheros por sus atributos, uno de los campos de la lista de extensiones puede empezar con una barra, seguida por un conjunto de carácteres de atributos. Cualquier atributo escrito solo, o con un + antes de él, indica que sólo se incluyan ficheros que tengan ese atributo activo. Cualquier atributo con un - antes de él indica que hay que dejar fuera los ficheros con ese atributo. Los atributos son 'r' para sólo lectura (read-only), 'h' para ocultos (hidden), 's' para archivos de sistema (system), 'd' para directorios (directory), y 'a' para tipo archivo (archive). Por ejemplo, una cadena de extension "PCX;BMP;/+rd" mostrará solamente ficheros PCX o BMP que sean de sólo lectura pero no los directorios. El diálogo del selector de ficheros será reescalado a la anchura y altura especificadas y al tamaño de la fuente. Si alguno de los parámetros está a cero, entonces la función escalará el diálogo a la dimensión apropiada de la pantalla. Devuelve cero si fue cerrada con el botón Cancelar, y distinto de cero si se cerró con OK.

gfx_mode_select

int gfx_mode_select(int *card, int *w, int *h);

Enseña el diálogo de selección de modo gráfico de Allegro, que permite al usuario seleccionar el modo y tarjeta de vídeo. Almacena la selección en las tres variables, y devuelve cero si se cerró con el botón Cancelar y distinto de cero si se cerró con OK.

gfx_mode_select_ex

int gfx_mode_select_ex(int *card, int *w, int *h, int *color_depth);

Versión extendida del diálogo de selección de modo gráfico, que permite al usuario seleccionar tanto el número de colores como la resolución y el controlador de vídeo. Esta versión de la función lee los valores iniciales de los parámetros cuando se activa, por lo que puede especificar los valores por defecto.

gui_shadow_box_proc

```
extern int (*gui_shadow_box_proc)(int msg, struct DIALOG *d, int c);
extern int (*gui_ctext_proc)(int msg, struct DIALOG *d, int c);
extern int (*gui_button_proc)(int msg, struct DIALOG *d, int c);
extern int (*gui_edit_proc)(int msg, struct DIALOG *d, int c);
extern int (*gui_list_proc)(int msg, struct DIALOG *d, int c);
extern int (*gui_text_list_proc)(int msg, struct DIALOG *d, int c);
```

Si ajusta alguno de estos punteros, las funciones a las que apunten serán usadas por los diálogos estándar de Allegro. Esto le permite personalizar el `look and feel', al estilo de gui_fg_color y gui_bg_color, pero con mucha mayor flexibilidad.

DETALLES ESPECÍFICOS DE UNIX

Para poder localizar cosas como los ficheros de configuración o traducción, Allegro necesita conocer el path de su ejecutable. Ya que no hay forma estándar para hacer eso, necesita capturar una copia de sus parámetros argv[], cosa que hace con trucos de preprocesador. Desafortunadamente no puede conseguirlo sin un poco de ayuda por su parte, por lo que tendrá que escribir END_OF_MAIN() justo tras su función main(). Muy fácil, realmente, y si se olvida de hacerlo, obtendrá un agradable error de enlazado sobre una función inexistente _mangled_main que se lo recordará :-)

GFX_*/Linux

Drivers: GFX_*/Linux

Cuando use Linux en modo consola, Allegro soporta los siguientes parámetros de tarjeta para la función set_gfx_mode():

· GFX_TEXT

Vuelve al modo texto.

· GFX_AUTODETECT

Permite que Allegro elija un controlador gráfico apropiado.

· GFX_AUTODETECT_FULLSCREEN

Autodetecta el driver gráfico, pero sólo usará drivers a pantalla completa, fallando si no están disponibles en la plataforma actual.

· GFX_AUTODETECT_WINDOWED

Igual que el anterior, pero sólo con drivers en ventana. Esto
siempre fallará bajo DOS.

· GFX_SAFE

Controlador especial para cuando quiere establecer un modo gráfico seguro y no le importa realmente en qué resolución o profundidad de color. Mire para más detalles la documentación de set_gfx_mode().

· GFX_FBCON

Usa el dispositivo framebuffer (ej: dev/fb0). Esto requiere que tenga soporte para framebuffer compilado en su núcleo, y que su hardware esté correctamente configurado. Actualmente es el único controlador en modo consola que funciona sin permisos de superusuario, a no ser que esté usando una versión de desarrollo de SVGAlib.

· GFX_VBEAF

Usa el controlador VBE/AF (vbead.drv), asumiendo que ha instalado uno que funcione bajo Linux (actualmente solo dos de los controladores del proyecto FreeBE/AF son capaces de hacerlo: ni idea sobre los de SciTech). VBE/AF requiere permisos de superusuario, pero es por ahora el único controlador para Linux que soporta aceleración de gráficos por hardware.

· GFX_SVGALIB

Usa la librería SVGAlib para mostrar gráficos. Esto requiere permisos de superusuario si su versión de SVGAlib los requiere.

· GFX_VGA

GFX_MODEX

Usa acceso directo al hardware para ajustar resoluciones VGA estándar o modo-X, soportando los mismos modos que las versiones DOS de estos controladores. Requiere permisos de superusuario.
GFX_*/X
Drivers: GFX_*/X
Cuando esté bajo X, Allegro soporta los siguientes parámetros de tarjeta para la función set_gfx_mode():
· GFX_TEXT
Vuelve al modo texto.
· GFX_AUTODETECT
Permite que Allegro elija un controlador gráfico apropiado.
· GFX_AUTODETECT_FULLSCREEN

Autodetecta el driver gráfico, pero sólo usará drivers a pantalla completa, fallando si no están disponibles en la plataforma actual.
· GFX_AUTODETECT_WINDOWED
Igual que el anterior, pero sólo con drivers en ventana. Esto siempre fallará bajo DOS.
· GFX_SAFE
Controlador especial para cuando quiere establecer un modo gráfico seguro y no le importa realmente en qué resolución o profundidad de color. Mire para más detalles la documentación de set_gfx_mode().
· GFX_XWINDOWS
El controlador gráfico X estándar. Esto debería funcionar en cualquier sistema Unix, y puede operar remotamente. No requiere permisos de superususario.
· GFX_XWINDOWS_FULLSCREEN
Igual que el anterior, pero mientras GFX_XWINDOWS se ejecuta en una ventana, este usará la extensión XF86VidMode para ejecutarse a pantalla completa incluso sin permisos de superusuario. Seguirá usando el protocolo X estándar, así que espere obtener el mismo bajo rendimiento que con el driver en ventana.
· GFX_XDGA
Usa la extensión DGA 1.0 de XFree86 para escribir directamente en la superficie de vídeo. DGA es normalmente más rápido que el modo X estándar, pero no produce programas en ventana que se comporten adecuadamente, y no funcionará remotamente. Este controlador requiere permisos de superusuario.
· GFX_XDGA_FULLSCREEN
Como GFX_XDGA, pero además cambia la resolución de la pantalla para que se ejecute en pantalla completa. Este controlador requiere permisos de superususario.
· GFX_XDGA2

Usa la nueva extensión DGA 2.0 de XFree86 4.0.x. Esta funcionará en pantalla completa, y soportará aceleración por hardware si está disponible. El controlador requiere premisos de superusuario.

· GFX_XDGA2_SOFT

Igual que GFX_XDGA2, pero desactiva la aceleración por hardware. Este controlador requiere permisos de
superusuario.

DIGI_*/Unix

Drivers: DIGI_*/Unix

Las funciones de sonido Unix soportan las siguientes tarjetas de sonido digital:

DIGI_AUTODETECT - permite que Allegro elija el controlador de sonidoDIGI_NONE
- sin sonido digitalDIGI_OSS - Open Sound SystemDIGI_ESD
- Enlightened Sound DaemonDIGI_ALSA - controlador de sonido ALSA

MIDI_*/Unix

Drivers: MIDI_*/Unix

Las funciones de sonido Unix soportan las siguientes tarjetas MIDI:

MIDI_AUTODETECT - permite que Allegro elija un controlador de sonido MIDIMIDI_NONE
- sin sonido MIDIMIDI_OSS - Open Sound SystemMIDI_DIGMID
- reproductor software basado en samplesMIDI_ALSA - controlador RawMIDI ALSA

DIFERENCIAS ENTRE PLATAFORMAS

Aquí tiene un rápido resumen de las cosas que pueden causar problemas cuando mueva su código de una plataforma a otra (puede encontrar una versión detallada de esto en la sección docs de la página web de Allegro).

Las versiones Windows y Unix requieren que escriba END_OF_MAIN() tras su función main(), lo que transforma mágicamente su main() de estilo ANSI C en un WinMain() de estilo Windows, y permite que el código Unix obtenga una copia de sus parámetros argv[].

En muchas plataformas Allegro funcionará lentamente si espera que bloquee automáticamente los bitmaps cuando dibuje sobre ellos. Para mejorar el rendimiento, necesita llamar usted mismo acquire_bitmap() y release_bitmap(), e intentar minimizar el número de bloqueos.

La versión Windows puede perder el contenido de la memoria de vídeo si el usuario cambia de tarea a otro programa, por lo que deberá tratar eso.

Ninguna de las plataformas actuales necesitan input polling, pero es posible que esto sea necesario en el futuro, por lo que si quiere asegurar al 100% la portabilidad de su programa, debería llamar poll_mouse() y poll_keyboard() en los lugares apropiados.

Allegro define un número estándar de macros que pueden ser usadas para verificar los atributos de la plataforma actual, o para aislarle de algunas diferencias entre sistemas:

ALLEGRO_PLATFORM_STR

Cadena de texto que contiene el nombre de la plataforma actual.
ALLEGRO_DOS
ALLEGRO_DJGPP
ALLEGRO_WATCOM
ALLEGRO_WINDOWS
ALLEGRO_MSVC
ALLEGRO_MINGW32
ALLEGRO_RSXNT
ALLEGRO_BCC32
ALLEGRO_UNIX
ALLEGRO_LINUX
ALLEGRO_BEOS
ALLEGRO_GCC
Definidos si está compilando en el sistema relevante. A menudo serán aplicables varias, ej: DOS+Watcom, o Windows+GCC+RSXNT.
ALLEGRO_I386
ALLEGRO_BIG_ENDIAN
ALLEGRO_LITTLE_ENDIAN
Definidos si está compilando para el procesador del tipo relevante.
ALLEGRO_VRAM_SINGLE_SURFACE
Definido si la pantalla es una sola superficie grande partida en múltiples sub-bitmaps de vídeo (ej: DOS), en vez de cada bitmap sea una entidad totalmente única (ej. Windows).
ALLEGRO_CONSOLE_OK
Definido cuando no está en modo gráfico, y hay una consola en modo texto a la cual puede mandar mensajes con printf(), salida estándar que puede ser capturada y redirigida por el usuario incluso cuando está en modo gráfico. Si este define está ausente, está ejecutando su programa en un entorno como Windows, el cual no tiene salida estándar.
ALLEGRO_LFN
Distinto de cero si hay soporte para ficheros de nombres largos, o cero si está limitado al formato 8.3 (en la versión DJGPP esto es una variable dependiente del entorno durante la ejecución).
INLINE
Use esto en lugar de la palabra reservada modificadora de función "inline" y su código funcionará correctamente en cualquiera de los compiladores soportados.
ZERO_SIZE
Use esto para declarar arrays de tamaño cero, ej: "char *line[ZERO_SIZE]" dentro de la estructura BITMAP. Algunos compiladores esperan un cero en ese lugar, mientras que otros no quieren tener nada dentro de [], por lo que esta macro permite que el mismo código funcione bien de cualquier modo.
LONG_LONG

Definido a lo que quiera que represente un entero "long long" de 64 bits para el compilador actual, o no definido si no está soportado.
OTHER_PATH_SEPARATOR
Definido a un carácter separador de ruta distinto de la barra para plataformas que lo usen (ej: antibarra bajo DOS y Windows), o definido a la barra normal si no hay otro separador de carácter.
DEVICE_SEPARATOR
Definido al carácter que separa el nombre de fichero del dispositivo (dos puntos para DOS y Windows), o cero si no hay
dispositivos explícitos en las rutas (ej: Unix).
USE_CONSOLE
Si define esto antes de incluir allegro.h, la versión Windows hará que su programa se compile como una aplicación de consola en vez del programa normal en modo gráfico.
END_OF_MAIN()
Al poner esto tras su función main() permitirá que los programas Windows funcionen con una rutina main() regular, para que no tenga que cambiar todo para usar WinMain().
REDUCIENDO EL TAMAÑO DE SU EJECUTABLE
Hay gente que se queja de que Allegro produce ejecutables muy grandes. Esto es cierto: con la versión DJGPP, un simple programa "hola mundo" ocupará unas 200k, aunque este tamaño por ejecutable es mucho menor en plataformas que soportan enlazado dinámico. Pero no se preocupe, Allegro ocupa un tamaño relativamente fijo, y no aumentará a medida que lo hace su programa. Tal y como George Foot dijo sin tapujos, quien esté preocupado por la relación entre código de biblioteca y programa debería ponerse manos a la obra y escribir más código de programa para equilibrar la cosa :-)
Dicho esto, hay varias cosas que puede hacer para reducir el tamaño de sus ejecutables:
· Para todas las plataformas, puede usar el compresor de
ejecutables UPX, el cual está disponible en http://upx.tsx.org/ . Este normalmente consigue una compresión de alrededor de un 40%. Cuando use DJGPP: para empezar, lea la sección 8.14 del FAQ de DJGPP, y tome nota del parámetro -s. ¡Y no se olvide de compilar su programa con las optimizaciones activadas!
· Si un programa DOS va a ser usado en un número limitado de resoluciones, puede especificar los controladores gráficos que desea incluir con su código: BEGIN_GFX_DRIVER_LIST driver1 driver2 etc...END_GFX_DRIVER_LIST
donde los nombres driverx son cualquiera de las definiciones: GFX_DRIVER_VBEAFGFX_DRIVER_VGAGFX_DRIVER_MODEXGFX_DRIVER_VES
A
3GFX_DRIVER_VESA2LGFX_DRIVER_VESA2BGFX_DRIVER_XTENDEDGFX_DRI
V ER_VESA1

Esta construcción debe ser incluida sólo en uno de sus ficheros C. El orden de los nombres es importante, porque la rutina de auto detección funciona de arriba a abajo hasta encontrar un controlador capaz de soportar el modo solicitado. Yo le sugiero que use la lista de arriba ordenada por defecto, y borre simplemente las líneas que no vaya a necesitar.

· Si su programa DOS no necesita usar todas las profundidades de color posibles, puede especificar cuales desea que sean soportadas por su programa: BEGIN_COLOR_DEPTH_LIST depth1 depth2 etc...END_COLOR_DEPTH_LIST
donde los nombres de profundidades de color son cualquiera de los defines:
COLOR_DEPTH_8COLOR_DEPTH_15COLOR_DEPTH_16COLOR_DEPTH_24COLOR_ DEPTH_32
Quitar cualquier profundidad de color le ahorrará un poco de espacio, con la excepción de los modos de 15 y 16 bits: éstos comparten una buena porción de código, así que si está incluyendo uno, no hay razón para quitar el otro. Esté avisado de que si intenta usar una profundidad de color no incluida en esta lista, isu programa se colgará de forma horrible!

· Del mismo modo que arriba, puede especificar qué controladores de sonido para DOS quiere incluir en su código: BEGIN_DIGI_DRIVER_LIST driver1 driver2 etc...END_DIGI_DRIVER_LIST
usando las definiciones de controladores digitales: DIGI_DRIVER_SOUNDSCAPEDIGI_DRIVER_AUDIODRIVEDIGI_DRIVER_WINSOUNDSYSDIGI_DRIVER_SB
y para la música MIDI:
BEGIN_MIDI_DRIVER_LIST driver1 driver2 etc...END_MIDI_DRIVER_LIST
usando las definiciones de controladores MIDI: MIDI_DRIVER_AWE32MIDI_DRIVER_DIGMIDMIDI_DRIVER_ADLIBMIDI_DRIVER_MPUMIDI_DRIVER_SB_OUT
Si va a usar alguno de éstas construcciones, debe usar las dos. Si sólo quiere incluir controladores de sonido digital, simplemente escriba DECLARE_MIDI_DRIVER_LIST() para no incluir controladores de música.

· De igual modo para los controladores de joystick de DOS puede declarar una lista:
BEGIN_JOYSTICK_DRIVER_LIST driver1 driver2 etc...END_JOYSTICK_DRIVER_LIST
usando las definiciones de controladores de joystick: JOYSTICK_DRIVER_WINGWARRIORJOYSTICK_DRIVER_SIDEWINDERJOYSTICK_DRIVER_GAMEPAD_PROJOYSTICK_DRIVER_GRIPJOYSTICK_DRIVER_STANDARDJOYSTICK_DRIVER_SNESPADJOYSTICK_DRIVER_PSXPADJOYSTICK_DRIVER_N64PADJOYSTICK_DRIVER_DB9JOYSTICK_DRIVER_TURBOGRAFXJOYSTICK_DRIVER_IFSEGA_ISAJOYSTICK_DRIVER_IFSEGA_PCIJOYSTICK_DRIVER_IFSEGA_PCI_FAST

El controlador estándar incluye soporte para joysticks duales, número superior de botones, Flightstick Pro, y Wingman Extreme, porque éstos son variaciones menores del código básico.

· Si _realmente_ está decidido a reducir los tamaños, mire el comienzo del fichero include/allegro/alconfig.h y verá las líneas:

#define ALLEGRO_COLOR8#define ALLEGRO_COLOR16#define ALLEGRO_COLOR24#define ALLEGRO_COLOR32

Si comenta cualquiera de estas definiciones y reconstruye la librería, tendrá una versión sin soporte para las profundidades de color ausentes, lo que incluso reducirá más el ejecutable que la macro DECLARE_COLOR_DEPTH_LIST(). El quitar el define ALLEGRO_COLOR16 eliminará el soporte para los modos de 15 y 16bits, ya que éstos comparten mucho

código.

Nota: los métodos mencionados para quitar drivers gráficos no usados sólo es aplicable a las versiones de la biblioteca enlazadas estáticamente, ejemplo: DOS. En plataformas Windows y Unix, puede crear Allegro como una DLL o biblioteca compartida, lo cual evita que se puedan usar estos métodos, pero ahorra tanto espacio que probablemente ni se llegará a preocupar. No obstante, quitar profundidades de color de alconfig.h funcionará en cualquier plataforma.

Si está distribuyendo una copia del programa setup junto con su juego, puede conseguir una reducción de tamaño dramática mezclando el código del programa setup con su programa principal, para que sólo sea necesario enlazar una copia de Allegro. Lea setup.txt para más detalles. En la versión DJGPP, tras comprimir su ejecutable, esto le ahorrará unos 200k en comparación con tener dos programas separados para setup y el propio juego.

DEPURANDO

Hay tres versiones de la biblioteca Allegro: el código normal optimizado, con soporte extra para depurar, y una versión para medir el rendimiento. Lea los ficheros readme específicos de cada plataforma para saber cómo instalar y enlazar con estas versiones alternativas. A pesar de que obviamente querrá usar la versión optimizada para la versión final de su programa, puede ser muy útil enlazar con la versión de depuración, porque hará la tarea de depuración más fácil, y porque incluye aserciones que le ayudarán a encontrar errores en su código con antelación. Allegro contiene varias funciones para ayudar en la depuración:

al_assert

void al_assert(char *file, int line);

Genera una aserción en el fichero en la línea especificada. El parámetro file debe estar codificado en ASCII. Si ha instalado un controlador propio de aserciones lo usa, o si la variable de entorno ALLEGRO_ASSERT escribe un mensaje en el fichero especificado por el entorno, y si no, aborta la ejecución

del programa con un mensaje de error. Normalmente usará la macro ASSERT() en vez de llamar directamente a esta función.
al_trace
void al_trace(char *msg, ...);
Muestra un mensaje de depuración usando una cadena con formato printf() codificada en ASCII. Si ha instalado una función trace propia, será usada, o si la variable de entorno ALLEGRO_TRACE existe se escribirá en el fichero especificado por ésta, en caso contrario se escribirá el mensaje en "allegro.log" en el directorio actual. Normalmente querrá usar la macro TRACE() en vez de llamar directamente a esta función.
ASSERT
void ASSERT(condition);
Macro ayudante de depuración. Normalmente se convierte en nada, pero si ha definido DEBUGMODE antes de incluir allegro.h, comprobará la condición indicada y llamará a al_assert() si ésta falla.
TRACE
void TRACE(char *msg, ...);
register_trace_handler Macro ayudante de depuración. Normalmente se convierte en nada, pero si ha definido DEBUGMODE antes de incluir allegro.h, pasará el mensaje (que debe estar codificado en ASCII) a al_trace().
register_assert_handler
void register_assert_handler(int
(*handler)(char *msg)); register_trace_handler Permite usar una función propia para tratar las aserciones fallidas. A su función se le pasará un mensaje de error formateado codificado en ASCII, y deberá devolver distinto de cero si ha procesado el error, o cero para continuar con las acciones por defecto. Puede usar esto para ignorar aserciones fallidas, o para mostrar mensajes de error en modo gráfico sin abortar el programa.
register_trace_handler
void register_trace_handler(int
(*handler)(char *msg)); register_assert_handler Permite usar una función propia para tratar los mensajes de trazado. A su función se le pasará un mensaje de error formateado codificado en ASCII, y deberá devolver distinto de cero si ha procesado el error, o cero para continuar con las acciones por defecto. Puede usar esto para ignorar los mensajes de trazado, para mostrarlos en un monitor monocromo secundario, etc.
COMANDOS MAKEFILE
Hay un número de opciones que puede usar para controlar exactamente cómo compilar Allegro. En plataformas Unix esto se hace pasando argumentos al script de configuración (ejecute "configure -help" para obtener una lista), en otras plataformas puede ajustar las siguientes variables de entorno:
· DEBUGMODE=1
Genera una versión para depurar, en vez de la versión normal optimizada.
· PROFILEMODE=1

Genera una versión para medir rendimientos, en vez de la versión normal optimizada.

· WARNMODE=1

Selecciona avisos de compilador más estrictos. Si está planeando trabajar en Allegro, en vez de simplemente usarlo, debería asegurarse de tener este modo activado.

· STATICLINK=1 (sólo MSVC y Mingw32)

Enlazar de forma estática, en vez de usar la DLL por defecto.

· TARGET_ARCH_COMPAT=[cpu] (implementado en la mayoría de plataformas GNU)

Esta opción optimizará el código para el procesador indicado manteniendo a la vez compatibilidad con procesadores anteriores. Ejemplo: set TARGET_ARCH_COMPAT=u586

· TARGET_ARCH_EXCL=[cpu] (implementado en la mayoría de plataformas GNU)

Esta opción optimizará el código para el procesador indicado. Tome nota de que el código generado *NO* funcionará en procesadores anteriores. Ejemplo: set TARGET_ARCH_EXCL=i586

· TARGET_OPTS=[opts] (implementado en la mayoría de plataformas GNU)

Esta opción le permite personalizar optimizaciones generales del compilador.

· CROSSCOMPILE=1 (sólo djgpp)

Le permite compilar la versión djgpp de la biblioteca bajo Linux, usando djgpp como compilador cruzado.

· ALLEGRO_USE_C=1 (sólo djgpp)

Permite generar la biblioteca con djgpp usando el código C de dibujado en vez de las rutinas en ensamblador. Esto sólo es útil para hacer pruebas, ya que la versión con ensamblador es más rápida.

Si usted sólo desea recompilar un programa test específico o una utilidad, puede especificarlo como parámetro del make, ej: "make demo" o "make grabber". El makefile tiene además varios comandos especiales:

· 'default'

El proceso normal. Compila la versión actual de la biblioteca (ya sea optimizada, para depurar o medir el rendimiento, seleccionada por las variables de entorno anteriores), genera el programa test y los ejemplos, y convierte los ficheros de documentación.

· 'all'

Compila las tres versiones de la biblioteca (optimizada, para depurar y para medir rendimiento), genera el programa test y los ejemplos, y convierte los ficheros de documentación.

· 'lib'

Compila la versión actual de la biblioteca (ya sea optimizada, para depurar o medir el rendimiento, seleccionada por las variables de entorno anteriores).

· 'install'

Copia la versión actual de la biblioteca (ya sea optimizada, para depurar o medir el rendimiento, seleccionada por las variables de entorno anteriores) en su directorio de bibliotecas, recompilando si es necesario, e instala los ficheros de cabecera de Allegro.

· 'installall'

Copia las tres versiones de la biblioteca (ya sea optimizada, para depurar o medir el rendimiento, seleccionada por las variables de entorno anteriores) en su directorio de bibliotecas, recompilando si es necesario, e instala los ficheros de cabecera de Allegro.

· 'uninstall'

Desinstala la biblioteca Allegro y borra los ficheros de cabecera de los directorios de su compilador. Esto requiere una utilidad 'rm' de estilo Unix, ej: del paquete GNU fileutils.

· 'docs'

Convierte los ficheros de documentación de las fuentes ._tx.

· 'docs-dvi' (sólo Unix)

Crea el fichero de independiente del dispositivo allegro.dvi. Este no es un comando por defecto, ya que necesita la herramienta texi2dvi para generarlo. El fichero generado está especialmente preparado para ser impreso en papel.

· 'docs-ps' o 'docs-gzipped-ps' (sólo Unix)

Crea un fichero Postscript a partir del fichero dvi generado previemante. Este no es un comando por defecto, ya que necesita las herramientas texi2dvi y dvips para generarlo. El segundo comando comprime el fichero Postscript generado. El fichero generado está especialmente preparado para ser impreso en papel.

· 'install-man' o 'install-gzipped-man' (sólo Unix)

Esto genera páginas de manual de Unix para cada función o variable de Allegro, y las instala. El segundo comando comprime las páginas antes de instalarlas.

· 'install-info' o 'install-gzipped-info' (sólo Unix)

Convierte la documentación en formato info y la instala. El segundo comando comprime el fichero info antes de instalarlo.

· 'clean'

Elimina todos los ficheros generados del directorio de Allegro, forzando una recompilación total la próxima vez que ejecute make. Este comando está diseñado de tal forma que si ejecuta "make install" y luego "make clean", todavía tendrá una versión funcional de Allegro . Esto requiere la utilidad rm de estilo Unix instalada, por ejemplo del paquete GNU fileutils.

· 'distclean'

Como 'make clean', pero más todavía. Elimina todos los archivos ejecutables y la documentación en formato HTML, dejándole con exactamente los mismos archivos que hay cuando descomprime una distribución nueva de Allegro.

· 'veryclean'

iUse esto con extrema precaución! Esta orden borra absolutamente todo archivo generado, incluyendo algunos que puede ser difícil recrear. Después de ejecutar este comando, una simple recompilación no funcionará: al menos tendrá que ejecutar "make depend", y tal vez también fixdll.bat si está usando la biblioteca de Windows. Este objetivo hace uso de herramientas no estándar como SED, así que a no ser que sepa usted lo que está haciendo y tenga estas herramientas instaladas, no debería usarlo.

· 'depend'

Regenera los archivos de dependencias (obj/*/makefile.dep). Es necesario ejecutar esto tras "make veryclean", o cuando se le añadan nuevas cabeceras a los fuentes de Allegro.

· 'compress' (sólo djgpp, Mingw32 y MSVC)

Usa el compresor de ejecutables DJP o UPX (el que tenga instalado) para comprimir los programas de ejemplo y las utilidades, lo cual puede recuperar una parte significativa de espacio libre en disco.