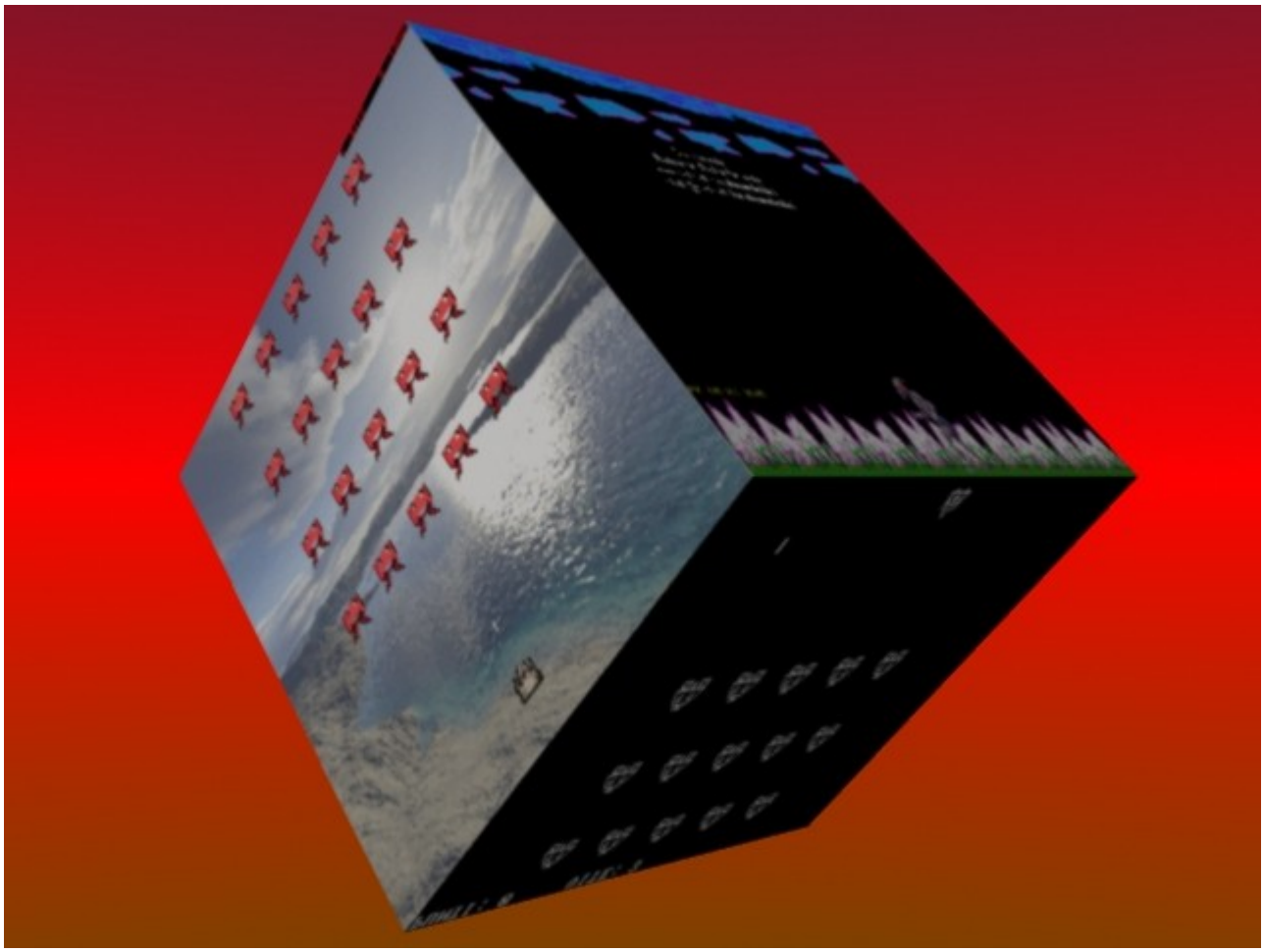
ALLEGRO.H dalla A alla Z...

(Programmare videogiochi in linguaggio C è facile)



Dott. Bartolomeo Davide Bertinetto

INDICE:

Bertinetto torna ai PDF gratuiti

Con queste righe vorrei spiegare perché sono ritornato al mondo dei PDF gratuiti.
AmazXX è un servizio stupefacente ma che non rispecchiava però più le mie intenzioni divulgative...
Tanti anni fa ho pubblicato il mio primo libro PDF e poi sono andato avanti con la scrittura di altri lavori. Tutti hanno avuto un buon successo nella loro forma gratuita. Successivamente ero passato alla grande 'A' e così è diventato il mio 'mondo' di scrittura per tantissimo tempo. Nessun problema quindi e il formato KindXX è stato un bel successo per me. Nonostante ciò il 'tarlo' del PDF gratuito nella mia mente è costantemente cresciuto fino ad arrivare ad oggi giorno, in cui realizzo questa idea. Voglio pertanto pubblicare per pura soddisfazione, senza troppi perfezionamenti 'maniacali' nella sintassi dei lavori, che fa perdere all'autore tempo in modo sproporzionato! Voglio dedicarmi unicamente all'idea e renderla accettabile nella forma ma senza esagerare troppo nei perfezionismi. Quando un lavoro è completo, viene pubblicato sul mio sito e quindi posso concentrarmi su altre idee senza dover riprendere continuamente vecchie opere per eliminare sviste insignificanti, solo perché si tratta di materiale a pagamento.
Purtroppo o per fortuna sono sommerso da nuovi progetti che affollano la mia mente, continuamente... Devo 'adattarmi' ancora una volta per riuscire a fare tutto da solo. Non ho voglia di affidarmi ad altre persone e così la pubblicazione gratuita PDF mi permetterà di caricare online lavori, esprimendo le mie idee anche se magari non perfetti dal punto di vista sintattico o di impaginazione.

Non pensate però a lavori di serie 'B', anzi confido che il risultato non sarà poi così diverso per il lettore rispetto a quando usavo Ebook Reader. Al contrario per me lo sarà molto in termini di mole di lavoro, dato che non avrò lo 'stress' dei 'sensi di colpa' se mi sarò scordato di rivedere una parte. Se qualcosa sarà da rianalizzare lo farò ma quando ne avrò il tempo, senza fretta.
Mi piace condividere i miei pensieri sui vari temi su cui sfociano le mie passioni. Certo qualche vecchia opera resterà su AmazXX e ne sono contento dato che si tratta di lavori realizzati con altri appassionati, con cui ho felicemente collaborato, con entusiasmo e soddisfazione!
Con questa premessa non intendo giustificarmi ma la scrivo per far conoscere il senso della mia decisione al lettore, per evitare che le mie creazioni sia considerate 'da poco'.
Per ultimo voglio aggiungere che il mio intento, fin da quando ho iniziato la passione di scrittore/divulgatore, è sempre stato quello di raggiungere la 'massima diffusione' e confido che in questo modo ci riuscirò all'ennesima potenza!
Concludo che così facendo sarà debellata completamente la pirateria dei mie lavori digitali, dato che ho lottato molto contro tale piaga del web ma ahimè senza successo!
Proprio perché i miei scritti sono in costante rilettura vi imploro di visitare il mio sito web www.bertinettobartolomeodavide.it di tanto in tanto, e controllare di essere in possesso dell'ultima versione del 'X' lavoro e non fidarvi troppo dei files provenienti da siti web che non hanno la mia approvazione.

Introduzione
Creare videogiochi è sempre stato il mio sogno fin da quando ero bambino con il piccolo Commodore 64. Ora che sono passati tanti anni ho deciso di realizzare un sito(http://www.bertinettobartolomeodavide.it/programmazione/ALLEGROH) per chi, come me, ha voglia di realizzare un videogame tutto da solo con poche righe di codice in linguaggio C / C++.
Sono sempre stato dell'idea che manipolare l'ambiente grafico di un computer sia uno degli aspetti più complessi della programmazione.
In questo sito sono spiegati, tramite esempi pratici, i sistemi base di realizzazione delle routine più diffuse nei videogiochi come: scrolling, visualizzazione di bitmap, sonoro, controlli di gioco, sprite, ecc...
Proprio per venire incontro a ragazzi che come me, hanno il sogno nel cassetto di scrivere un proprio video game, ho scritto un libro in cui spiego un passo alla volta, partendo da zero, come creare un videogioco completo e perfettamente funzionante sotto Windows e Linux.
Saranno sviluppati vari listati per questi due sistemi operativi un passo alla volta. Partendo dall'installazione della libreria allegro.h, fino a creare complessi videogiochi.
Il libro inizia spiegando i rudimenti della grafica 2D e poi di quella 3D con la gestione delle openGL in allegro.h grazie una una libreria supplementare chiamata alleggl.h. La panoramica dei videogiochi attuali e di ieri è sviluppata a 360 gradi con l'uso di software completamente gratuiti.

In questo libro ho raccolto tutti i listati più significativi che ho scritto nel corso degli anni. Molti erano già presenti su internet, nel mio sito personale(www.bertinettobartolomeodavide.it), dopo tutto questo tempo ho deciso di ricontrollarli... In questo modo li ho resi più chiari e sono stati eliminati molti bug. Tutto questo materiale è stato utilizzato da me per “imparare ad utilizzare la libreria allegro.h”.
Questo manuale deve essere considerato un libro a tutti gli effetti che raccoglie dei listati in linguaggio C/ C++ commentati. L'utente certamente imparerà le basi della programmazione di videogiochi con la libreria allegro.h e allegGL.h.

Una nota sulla licenza libera della libreria ALLEGRO.H (Inglese e Italiano)
La licenza giftware (versione originale in inglese)
Allegro is gift-ware. It was created by a number of people working in cooperation, and is given to you freely as a gift. You may use, modify, redistribute, and generally hack it about in any way you like, and you do not have to give us anything in return.
However, if you like this product you are encouraged to thank us by making a return gift to the Allegro community. This could be by writing an add-on package, providing a useful bug report, making an

improvement to the library, or perhaps just releasing the sources of your program so that other people can learn from them. If you redistribute parts of this code or make a game using it, it would be nice if you mentioned Allegro somewhere in the credits, but you are not required to do this. We trust you not to abuse our generosity.
By Shawn Hargreaves, 18 October 1998.
DISCLAIMER: THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

La licenza giftware (traduzione)
Allegro è giftware. E' stata creata da molte persone che hanno lavorato insieme, e vi è data gratuitamente come un regalo. Potete usarla, ridistribuirla e generalmente modificarla in tutti i modi che desideri, e non dovete darci niente in cambio.
Comunque, se vi piace questo prodotto siete incoraggiati a ringraziare la comunità Allegro con un regalo. Questo può essere fatto scrivendo un add-on, riportando un utile bug-report, facendo qualche miglioramento alla libreria, o anche solo rilasciando i sorgenti del vostro programma in modo che altre persone possano imparare da essi. Se ridistribuite parti del codice della libreria o create un gioco con essa, sarebbe carino se menzionaste Allegro da qualche parte nei crediti, ma non è necessario che lo facciate. Siamo certi che non abuserete della nostra generosità.
Scritto da Shawn Hargreaves, 18 Ottobre 1998.

Nota:
Come avrete potuto capire, la grandiosa libreria allegro.h, è liberamente utilizzabile in tutti i programmi che creerete. Siano essi di natura commerciale o gratuita. L'unica richiesta che viene fatta dagli sviluppatori (gente come voi e come me) è di menzionare l'uso di questa libreria da qualche parte nel vostro programma, solo se volete.

Diritti e licenze di questo libro

Stanno per seguire le indicazioni per la messa a punto della compilazione della libreria su compilatore C. Aggiungo però che questo passaggio per molti rappresenta un difficoltoso ostacolo da superare. Per questo motivo e anche perché alcuni file necessari potrebbero non essere più disponibili sul web, ho inserito sul mio sito web molte versioni del compilatore per Windows(testate fino alla versione XP) ma anche per Dos già pronte all'uso a questa pagina:

http://www.bertinettobartolomeodavide.it/programmazione/ALLEGROH/compilarealegroh/indexcompiler.htm

Come installare in ambiente Windows la libreria Allegro.h 4.20 in Dev-C++ 4.9.8.0 – Testata su windows XP -
FILE DA SCARICARE:

1.Scaricare la versione minima delle DirectX 8 SDK per MingW32(dx80_mgw.zip). Questo file lo potete trovare su questa home page: 'http://alleg.sourceforge.net/wip.html'.
2.Scaricare dal sito 'http://bloodshed.net/dev/devcpp.html' la versione completa di Dev-C++ 5 'devcpp4980.exe'.
3.Scaricate il file 'all420.zip' di allegro.h dal sito 'http://www.allegro.cc/'. Ora viene trattata la versione 4.2.0 ma per le nuove versioni non dovrebbero esserci problemi.

INSTALLAZIONE:
1.Decomprimete il file delle DirectX 'dx80_mgw.zip' nella directory già presente chiamata 'c:\compilatore\mingw32'.
2.Lanciate il file di INSTALLAZIONE di Dev-C++ 'devcpp4980.exe' ed installate il tutto nella directory già presente 'c:\compilatore'. Terminata l'installazione scegliete la lingua italiana.
3.Decomprimete il file 'all420.zip' nella directory già presente 'c:\compilatore'.
Dopo aver installato sul vostro disco fisso questi archivi, sarà presente una cartella chiamata 'c:\compilatore' dove saranno contenuti tutti gli elementi necessari alla compilazione dei vostri videogiochi o applicazioni create con la libreria Allegro.h.
I passaggi non sono ancora terminati. Adesso è necessario compilare tutto quanto perché si verifichi un funzionamento corretto!

COMPILAZIONE:
Creiamo un file chiamato 'go.bat' con il BLOCCO NOTE di Windows e salviamolo in 'c:\compilatore'.
All'interno del file 'go.bat' dobbiamo inserire il seguente contenuto:
path=c:\compilatore\bin;%path%
set MINGDIR=C:\compilatore
path=c:\compilatore\mingw32\bin;%path%
set MINGDIR=c:\compilatore\mingw32
Una volta salvato il file 'go.bat' con il suo contenuto, apriamo una finestra DOS e con il comando 'cd \compilatore' ci spostiamo nella directory che ci interessa.
Ora lanciamo il file 'go.bat', che provvederà alla creazione dei nuovi percorsi necessari alla compilazione.
Ora possiamo verificare se il processo ha avuto successo con il comando 'gcc -v'. Se ci compare un messaggio simile a questo, significa che è tutto ok:
Reading specs from c:/compilatore/bin/...
gcc version 3.2 (mingw special)
Compiliamo la libreria allegro.h nel compilatore Dev-C++, spostandoci nella directory 'c:\compilatore\allegro' con il comando 'cd allegro'.
Digitiamo il comando 'fix mingw32' che indica quale compilatore utilizziamo.
Digitiamo il comando 'make' con il quale inizierà la prima fase della compilazione.
Finita la prima fase dovremo digitare 'make install' per iniziare la seconda fase della compilazione.
Ora la libreria allegro.h è integrata nel compilatore. Possiamo avviare Dev-C++ aprendo il file 'DevCpp.exe' presente 'c:\compilatore'.
Prima di entrare nel editor di compilazione ci verrà chiesto di scegliere il tipo di interfaccia e di confermare.
Con il puntatore clicchiamo su 'Strumenti' e poi su 'Opzioni di Compilazione', nella pagina 'Compilatore'. Selezioniamo la voce 'Aggiungi i comandi seguenti in fase di compilazione:' ed inseriamo nello spazio bianco sottostante il comando necessario per linkare la libreria allegro.h '-lalleg' e confermiamo con il tasto 'OK'
Spostiamoci alla voce 'Cartelle', sempre sotto questa finestra e poi su 'file binari' e inseriamo questo percorso: 'C:\COMPILATORE\Bin' e questo: 'C:\COMPILATORE\lib\gcc-lib\mingw32\3.2'. Ora ci spostiamo su 'Librerie' ed inseriamo questo percorso: 'C:\COMPILATORE\lib' e 'C:\compilatore\allegro\lib\mingw32'. Quindi andiamo su 'Include C' ed inseriamo questo percorso: 'C:\COMPILATORE\include' e 'C:\compilatore\allegro\include'. In fine ci spostiamo su 'Include C++' ed inseriamo questi percorsi: 'C:\COMPILATORE\include\c++' e 'C:\COMPILATORE\include\c++\mingw32' e 'C:\COMPILATORE\include\c++\backward' e 'C:\COMPILATORE\include'. Poi confermiamo cliccando il stato 'ok'. Molti di questi percorsi probabilmente non sono necessari, ma è buona precauzione inserirli!
Ora il compilatore Dev-C++ è perfettamente ingrado di gestire i sorgenti di allegro.h.

PROVARE IL COMPILATORE:
Caricate un file sorgente di esempio nel compilatore premendo sul menù 'File' e poi su 'Apri Progetto o file...'. Ora cliccate su 'Esegui' e poi su 'Compila & Esegui' per compilare e lanciare il programma... PS: Non dimenticate di inserire i vostri sorgenti in una cartella dove sia presente il file 'alleg42.dll' o similari, altrimenti i vostri listati non funzioneranno!
Se intendete creare un vostro sorgente aprite sempre il menù 'File' e poi cliccate su 'Nuovo', 'File sorgente' ed iniziate a scrivere. Una volta finito di scrivere il sorgente cliccate su 'Esegui' e poi su 'Compila & Esegui' per compilare e lanciare il programma... PS: Non dimenticate di inserire i vostri sorgenti in una cartella dove sia presente il file 'alleg42.dll' o similari, altrimenti i vostri listati non funzioneranno!
AVVERTO TUTTI GLI UTENTI CHE QUESTA E' LA TECNICA USATA DA ME PER LINKARE LA LIBRERIA PER COMPILARE ALLEGRO.H IN DEV-C++ E LE DIRECTX. LE VERSIONI DEI FILE USATI RISALGONO AL 2006. E QUINDI LE VERSIONI E LE REGOLE POTREBBERO ESSERE CAMBIATE OGGI. NEL MOMENTO IN CUI SCRIVO QUESTE RIGHE (11-03-06) HO PROVATO A COMPILARE L'ULTIMA VERSIONE DI ALLEGRO.H IN MIO POSSESSO (ver. 4.20) SEGUENDO QUESTO METODO ED HA FUNZIONATO TUTTO PERFETTAMENTE!

Come installare in ambiente Linux la libreria Allegro.h 4.20
Scaricate il file 'all420.zip' di allegro.h dal sito 'http://www.allegro.cc/'. Ora viene trattata la versione 4.2.0 (in realtà all'interno dell'archivio è presente la versione 4.0.3) ma per le nuove versioni non dovrebbero esserci problemi.

INSTALLAZIONE:
Decomprimete il file 'all420.zip' nella directory già presente '/home/nomeutente/', a questo punto dovreste ottenere '/home/nomeutente/allegro-4.0.3/'.
I passaggi sono solo all'inizio... Adesso è necessario compilare tutto quanto perché si verifichi un funzionamento corretto!

COMPILAZIONE:
Spostiamoci con il puntatore nel 'GNOME menù', l'icona a cappello per intenderci, poi su 'strumenti di sistema' e clicchiamo su 'Terminale'.
Ora appare una finestra a linea di comando con cursore lampeggiante. Bene, digitiamo 'bash' e premiano il tasto invio.
Dovrebbe apparire sul prompt della linea di comando la voce '[Nomeutente@localhost Nomeutente$]', prima del cursore lampeggiante.
Con il comando 'cd allegro-4.0.3', premiamo invio, ci spostiamo all'interno della cartella di alleggro.h e digitiamo 'chmod +x fix.sh', premiamo invio.
Ancora './fix.sh unix', premiamo invio, poi './configure', premiamo invio e adesso 'make' e premiamo invio.
Un' altra piccola sequenza di comandi: 'su -c "make install"', 'su -c "make install-man"', 'su -c "make install-info"'. Dopo ogni singolo comando tra apostrofi premiamo invio.
PS: dopo ognuno di questi tre comandi il sistema vi chiederà di inserire la password di root (la stessa che utilizzate per accedere a linux con il vostro profilo utente). Digitatela e premete invio.
Digitiamo './configure --enable-static', './configure --disable-shared', './configure --enable-dbglib', './configure –enable-dbgprog'. Nuovamente va premuto il tasto di invio per quattro volte dopo ogni comando fra apostrofi.
Altri due comandi da digitare: 'autoconf' e 'make depend', sempre seguiti dal tasto di invio.
Scriviamo ancora questo: './configure --prefix=$HOME'
Allegro.h è perfettamente in grado di compilare i nostri listati sotto il sistema operativo linux con questa linea di comando: 'gcc miofile.c -o mioprogramma `allegro-config –libs`'
L'ultimo appunto è quello di inserire il percorso delle librerie di allegro.h, digitando questa riga e premendo invio ogni volta che si apre la finestra del terminale: 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:usr/local/lib'
Per lanciare l'eseguibile digitiamo al prompt dei comandi './mioprogramma' e premere invio.

Installare sotto Windows allegGL.h ver. 0.2.4 in DevCpp ver. 4.9.8.0 con allegro.h ver. 4.20 – Testata su windows XP -

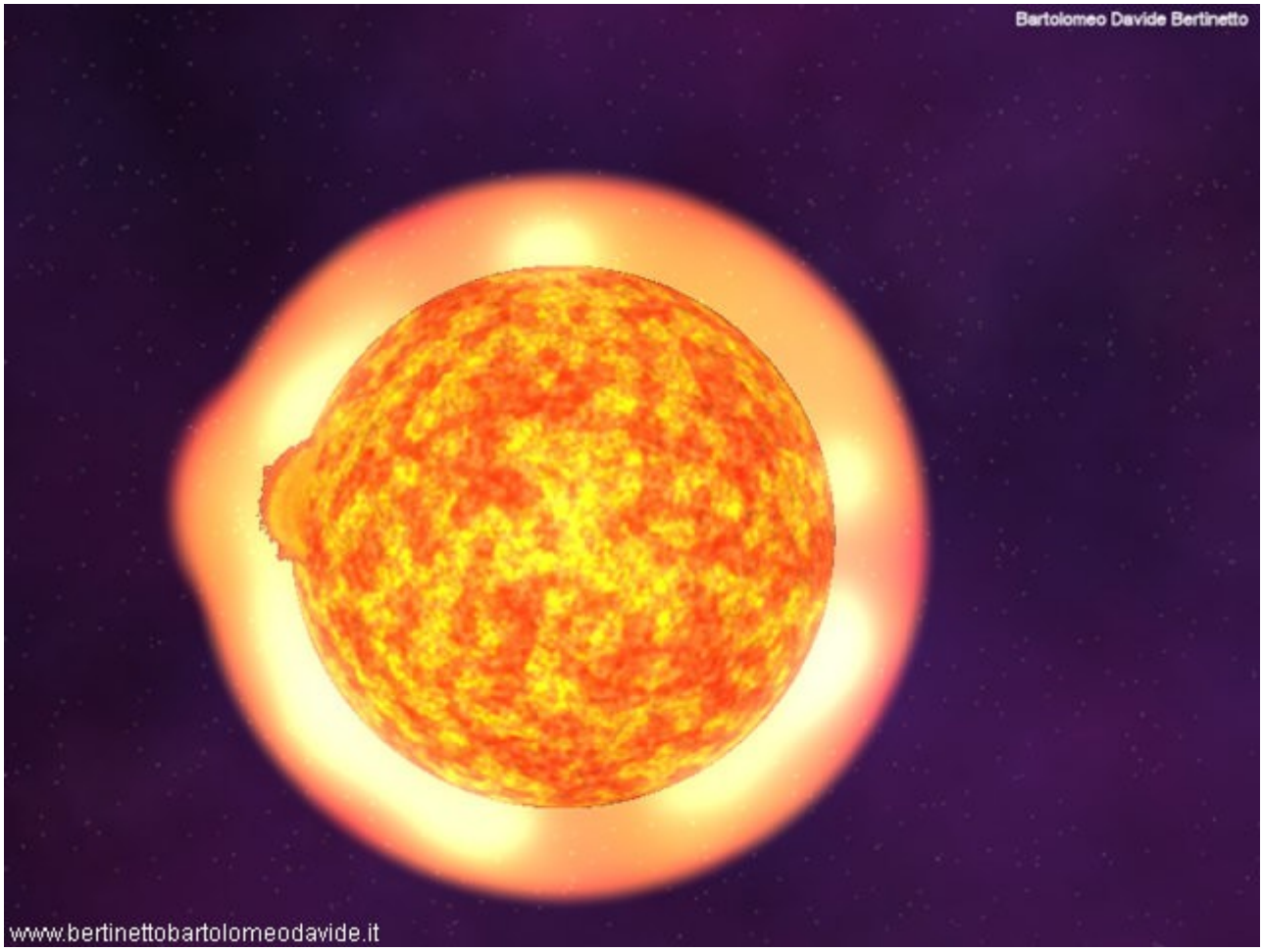
Prima di tutto si deve decomprimere l'archivio di alleggl.h (oggi ver. 0.2.4) nella cartella 'c:\compilatore'
Otterrete così una cartella chiamata 'c:\compilatore\alleggl'.
Aprite una finestra DOS e andate nella cartella 'c:\compilatore' digitando 'cd \compilatore'.
In questa cartella dovrebbe essere presente il file 'go.bat' che era necessario per installare allegro.h in DevCpp. Lanciatelo!
Fatto questo, spostatevi nella directory 'c:\compilatore\alleggl' con il comando 'cd alleggl'
Digitate il comando 'fix mingw32'
A questo punto digitate 'make'
Poi 'make install'
Installazione terminata!

Ora lanciate 'DevCpp' e andate in 'Strumenti' e poi su 'Opzioni di Compilazione'
Alla voce, che dovrebbe essere già selezionata, 'aggiungi i comandi seguenti in fase di compilazione'
I seguenti comandi e nel giusto ordine: '-lagl -luser32 -lgdi32 -lopengl32 -lglu32 -lalleg'
Il compilatore è pronto per eseguire i vostri programmi 3d OpenGL!

In questa spiegazione faccio riferimento alla versione di DevCpp 4.9.8.0, ad Allegro.h 4.2.0 e a Alleggl.h 0.2.4. Dovrebbe essere tutto uguale anche per versioni successive.

ATTENZIONE: Alleggl.h è possibile utilizzarlo anche su Linux e altri sistemi operativi. In questo libro però verrà trattata solamente per quello che riguarda il 3D sotto ambienti Windows. Tutti i listati presenti in queste pagine per la libreria alleggl.h dovrebbero essere identici e completamente compatibili per tutti i sistemi.

01 - Programma che ci permette di caricare un immagine presente in un file con estensione TGA, BMP, PCX e LBM.



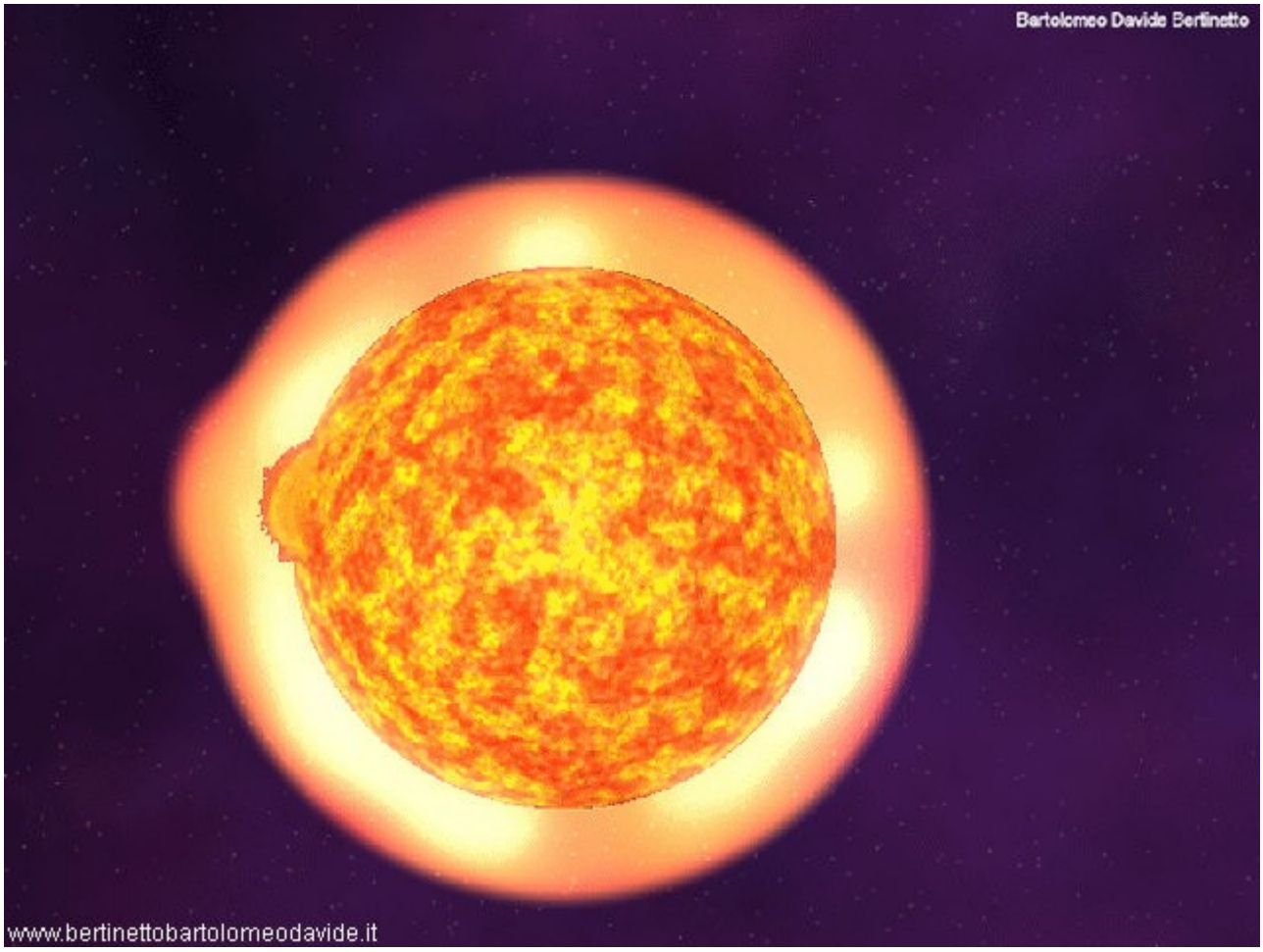

```
/* LISTATO C/C++ DI IMMAGINE
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
*/

#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA
#include "allegro.h" /* CARICA LA LIBRERIA ALLEGRO.H */
int main() /* INIZIO PROCEDURA */
{
BITMAP *immagine; /* DICHIARAZIONE VARIABILE IMMAGINE */
PALETTE colori; /* DICHIARAZIONE VARIABILE COLORI */
allegro_init(); /* INIZIALIZZA LA LIBRERIA ALLEGRO */
install_keyboard(); /* INSTALLA LA TASTIERA */
set_color_depth(32); /* IMPOSTA LA PALETTE DEI COLORI A 32 BIT */
set_gfx_mode(GFX_GDI, 640, 480, 0, 0); /* IMPOSTA LA MODALITA' GRAFICA 640x480 */
immagine=load_bitmap("immagine.bmp", colori); /* CARICA IL FILE IMMAGINE.BMP */
blit(immagine, screen, 0, 0, 0, 0, 640, 480); /* VISUALIZZA L'IMMAGINE SULLO SCHERMO */
destroy_bitmap(immagine); /* ELIMINA L'IMMAGINE DAL BUFFER PRIMA DI USCIRE DAL
PROGRAMMA */
readkey(); /* LEGGE UN TASTO QUALSIASI DA TASTIERA */
allegro_exit(); /* TERMINA LA LIBRERIA ALLEGRO */
}
END_OF_MAIN (); /* FINE DEL PROGRAMMA */
```

Analizziamo le parti salienti del programma '01immagine.c'

Il requisito indispensabile per utilizzare la libreria allegro.h è quello di inserire la riga, subito dopo aver inserito alcune librerie che potrebbero essere necessarie in fase di sviluppo del listato: stdio.h, string.h e time.h, '#include <allegro.h> ...' seguita da 'allegro_init(); ...' e per chiudere 'allegro_exit(); ...' ed alla fine dell'intero programma 'END_OF_MAIN ();'.
Il secondo passo e quello di inserire l'uso della tastiera, quindi digitare 'install_keyboard(); ...', a questo punto il comando 'readkey();' permette la lettura di un carattere da tastiera e quindi ne attende la pressione.
Adesso con il comando 'set_color_depth(32);' viene settata la profondità di colore dell'immagine, che in questo caso è di 32 bit ma i parametri accettati sono anche 16 bit e 8 bit(in questo caso deve essere impostata la palette da usare, come nella immagine in formato PCX e LBM che sono a 256 colori).
Non ci resta che settare la modalità grafica da usare con il comando 'set_gfx_mode(GFX_GDI, 640, 480, 0, 0);' che in questo caso seleziona la modalità automaticamente(francamente consiglio di scegliere sempre modalità automatica evitando così incompatibilità da parte di alcuni computer), altrimenti si può scegliere tra directx, windowed, ecc... Subito dopo si danno le dimensioni in pixel dello schermo o della finestra da visualizzare, in questo caso 640x480 pixel.
Cerchiamo adesso di capire come viene caricata l'immagine in memoria: tramite il comando 'immagine=load_bitmap("immagine.bmp", colori);' viene caricato il file' immagine.bmp'(potrebbe anche essere un *.bmp o *.pcx o un *.lbm), così il contenuto del file viene trasferito alla variabile puntatore per il bitmap 'immagine' e con la variabile 'colori'(parametro indispensabile per le profondità di colore a 8 bit) viene assegnata la palette.
Il comando 'blit(immagine, screen, 0, 0, 0, 0, 640, 480);' visualizza l'immagine contenuta nella variabile puntatore 'immagine' sullo schermo tramite la voce 'screen'(che non è una variabile!). I numeri indicati dopo, tra le virgole, indicano la posizione e la porzione dell'immagine da visualizzare.
Con il comando 'destroy_bitmap(immagine);' ripuliamo dalla memoria del computer la variabile puntatore 'immagine' azzerandola del suo contenuto prima di uscire dal programma.

Formati supportati da allegro.h per i files contenenti immagini:
TARGA (TrueVision): estensione *.tga con profondità di colore a 24 bit. Allegro è capace di leggerlo sia in modalità compressa che non compressa.
Bit Map Windows o OS/2: estensione *.bmp con profondità di colore di 8, 24 e 32 bit. Allegro è capace di leggerlo in 8 e 24 bit in modalità Windows RGB.
Zsoft PaintBrush: estensione *.pcx con profondità di colore a 1 bit, 4 bit e 8 bit. Allegro è in grado di leggere il formato a 8 bit.
Deluxe Paint: estensione *.lbm con profondità di colore a 8 bit. Allegro è capace di leggerlo sia in modalità compressa che non compressa.

02 - Versione ottimizzata del programma che ci permette di caricare un immagine con profondità di colore a 8 bit (palette da 256 colori).




```
/* LISTATO C/C++ DI IMMAGINEOTTOBIT
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
*/

#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA
#include "allegro.h" /* CARICA LA LIBRERIA ALLEGRO.H */
int main() /* INIZIO PROCEDURA */
{
BITMAP *immagine; /* DICHIARAZIONE VARIABILE IMMAGINE */
PALETTE colori; /* DICHIARAZIONE VARIABILE COLORI */
allegro_init(); /* INIZIALIZZA LA LIBRERIA ALLEGRO */
install_keyboard(); /* INSTALLA LA TASTIERA */
set_gfx_mode(GFX_GDI, 640, 480, 0, 0); /* IMPOSTA LA MODALITA' GRAFICA 640x480 */
immagine=load_bitmap("immagine.pcx", colori); /* CARICA IL FILE IMMAGINE.PCX */
set_palette(colori); /* IMPOSTA LA PALETTE DEI COLORI */
blit(immagine, screen, 0, 0, 0, 0, 640, 480); /* VISUALIZZA L'IMMAGINE SULLO SCHERMO */
destroy_bitmap(immagine); /* ELIMINA L'IMMAGINE DAL BUFFER PRIMA DI USCIRE DAL
PROGRAMMA */
readkey(); /* LEGGE UN TASTO QUALSIASI DA TASTIERA */
allegro_exit(); /* TERMINA LA LIBRERIA ALLEGRO */
}
END_OF_MAIN (); /* FINE DEL PROGRAMMA */
```

Analizziamo le parti salienti del programma 'immagineottobit.c'

Per visualizzare correttamente un'immagine a 256 colori è sufficiente apportare poche e semplici modifiche al listato precedente. Quindi eliminiamo il comando 'set_color_depth(32);' che si occupa delle immagine true color. Ed inseriamo il comando 'set_palette(colori);' subito dopo il comando necessario al caricamento del file 'immagine=load_bitmap("immagine.pcx", colori);'. A questo punto il programma è pronto per visualizzare un immagine ad otto bit. L'ultima nota da segnalare è quella di porre particolare attenzione, nel caso in cui si utilizzino più immagini a 256 colori nello stesso momento su schermo, è di limitare la palette di tutte le immagini alla medesima. Altrimenti si correrà il rischio di alterare alcuni colori durante la visualizzazione, viste le risorse grafiche limitate.

03 - Generare una routine di scrolling.


```
/* LISTATO C/C++ DI SCROLLING
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
*/
#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA
#include “allegro.h” /* INCLUSA LA LIBRERIA ALLEGRO.H */
BITMAP *immagine, *buffer, *buffer2; /* DICHIARAZIONI VARIABILI DEI BITMAP */
PALLETE colori; /* DICHIARAZIONE VARIABILE DELLA PALETTE */
int y; /* DICHIARAZIONE VARIABILE Y DI SPOSTAMENTO SCROLLING */
int pattern; /* DICHIARAZIONE DI VARIABILE DEL PATTERN DA RIPETERE */
void doppiobuffering() /* PROCEDURA DI DOPPIO BUFFERING */
{
```

```
vsync(); /* SINCRONIZZA L'AGGIORNAMENTO DELLO SCHERMO */
blit(buffer2, screen, 0, 0, 0, 0, 640, 480); /* VISUALIZZA IL SECONDO BUFFER SULLO SCHERMO
*/
clear(buffer2); /* CANCELLA IL SECONDO BUFFER */
}
void buffering() /* PROCEDURA DI PRIMO BUFFERING */
{
blit(buffer, buffer2, 0, 0, 0, 0, 640, 480); /* TRASFERISCE IL PRIMO BUFFER NEL SECONDO
BUFFER */
}
void blocco() /* PROCEDURA PER RIPETERE BLOCCHI DI PATTERN PER CHEARE IL BITMAP
DI SFONDO */
{
for (pattern=0;pattern<=1920;pattern=pattern+480) { /* CICLO FOR CHE RIPETE IL PATTERN PER 3
VOLTE A PASSI DI 480 PIXEL */
blit(immagine,buffer,0,0,0,pattern,640,480); /* INSERISCE L'IMMAGINE NEL PRIMO BUFFER */
}
}
void scrolling() /* SPOSTA LO SCHERMO IN SCROLLING VERTICALE */
{
for (y=0;y>=-1440;y=y-2) { /* CICLO FOR CHE FA AVANZARE LO SCHERMO SULL ASSE Y DI 2
PIXEL*/
blit (buffer, buffer2, 0, 0, 0, y, 640, 1920); /* INSERISCE IL PRIMO BUFFER NEL SECONDO
BUFFER CREANDO L'ILLUSIONE DELLO SPOSTAMENTO IN Y */
doppiobuffering(); /* CARICA LA PROCEDURA DI DOPPIO BUFFERING VISUALIZZANDO IL
RISULTATO SULLO SCHERMO */
}
}
int main() /* INIZIO PROCEDURA PRINCIPALE DI ALLEGRO */
{
allegro_init(); /* INIZIALIZZAZIONE DI ALLEGRO */
install_keyboard(); /* INSTALLAZIONE DELLA TASTIERA */
set_color_depth(32); /* SETTA LA PROFONDITA' DI COLORE A 32 BIT */
set_gfx_mode(GFX_GDI, 640, 480, 0, 0); /* CONFIGURA AUTOMATICAMENTE LA MODALITA
GRAFICA E IMPOSTA LA RISOLUZIONE A 640x480 PIXEL */
immagine=load_bitmap("scrolling.bmp", colori); /* CARICA IL FILE SCROLLING.BMP E LO
METTE NELLA VARIABILE IMMAGINE */
buffer = create_bitmap(640, 1920); /* CREA LA DIMENSIONE DEL PRIMO BUFFERING */
buffer2 = create_bitmap(640, 1920); /* CREA LA DIMENSIONE DEL DOPPIO BUFFERING */
clear(buffer); /* CANCELLA IL PRIMO BUFFER */
blocco(); /* CARICA LA PROCEDURA DI RIPETIZIONE PEL PATTERN */
buffering(); /* CARICA LA PROCEDURA DI PRIMO BUFFERING */
scrolling(); /* CARICA LA PROCEDURA DI SCROLLING */
destroy_bitmap(immagine); /* DISTRUGGE IL BITMAP IMMAGINE */
destroy_bitmap(buffer); /* DISTRUGGE IL PRIMO BUFFER */
readkey(); /* ATTENDE LA PRESSIONE DI UN TASTO */
allegro_exit(); /* TERMINA LA LIBRERIA ALLEGRO */
}
END_OF_MAIN (); /* CONCLUDE IL PROGRAMMA */
```

Analizziamo le parti salienti del programma 'scrolling.c'

Visto che si deve simulare il movimento dello schermo(scrolling), è necessario utilizzare un procedimento chiamato doppio buffering in cui l'immagine viene creata due volte in due buffer diversi, dove solo uno visualizza il risultato su schermo. Questo procedimento evita lo sfarfallio di aggiornamento

dello schermo. Ciò è realizzato nelle procedure 'void doppiobuffering() {...}' e 'void buffering() {...}'. Ora passiamo alla creazione della superficie bitmap da spostare per creare lo scrolling. Così viene moltiplicata l'immagine tramite il ciclo 'for' sull'asse y. Si ripete il pattern(l'immagine) per X volte. Il risultato viene trasferito con il comando 'blit' nel primo buffer. Tutto questo nella procedura 'void blocco() {...}'. In 'void scrolling() { ... }' è rappresentato il centro del programma in cui viene creato il vero e proprio scrolling. Il comando 'for' permette lo spostamento della finestra 'blit' sull'asse y per un certo numero di pixel portando il risultato dal primo al secondo buffer. IL GIOCO E' FATTO!!! Ancora all'interno della procedura void scrolling() { ... } viene chiamata un'altra procedura denominata 'doppiobuffering();' per visualizzare il risultato su schermo. Anche in questo caso il requisito indispensabile(oltre alle solite librerie di sistema che potrebbero tornare utili) per utilizzare la libreria allegro.h è quello di inserire la riga '#include <allegro.h> ...' seguita da 'allegro_init(); ...' e per chiudere 'allegro_exit(); ...' ed alla fine dell'intero programma 'END_OF_MAIN ();'. Il secondo passo e quello di inserire l'uso della tastiera, quindi digitare 'install_keyboard(); ...' a questo punto il comando 'readkey();' permette la lettura di un carattere da tastiera e quindi ne attende la pressione. Adesso con il comando 'set_color_depth(32);' viene setta la profondità di colore dell'immagine, che in questo caso è di 32 bit ma i parametri accettati sono anche 16 bit e 8 bit(in caso di 8 bit deve essere impostata la palette da usare). Non ci resta che settare la modalità grafica da usare con il comando 'set_gfx_mode(GFX_GDI, 640, 480, 0, 0);' che in questo caso seleziona la modalità automaticamente, altrimenti si può scegliere tra directx, windowed, ecc... Subito dopo si danno le dimensioni in pixel dello schermo o della finestra da visualizzare, in questo caso 640x480 pixel. Cerchiamo adesso di capire come viene caricata l'immagine in memoria: tramite il comando 'immagine=load_bitmap("scrolling.bmp", colori);' viene caricato il file scrolling.bmp(potrebbe anche essere un *.tga o *.pcx o *.lbm), così il contenuto del file viene trasferito alla variabile immagine, mentre con la variabile colori viene assegnata la palette. I comandi 'create_bitmap(640,1920)' su buffer e buffer2 determinano la grandezza della superficie del bitmap da scollare(da spostare). 'clear(buffer)' cancella semplicemente il contenuto di buffer. Sono richiamate le procedure 'blocco();' 'buffering();' 'scrolling();' che vengono ripetute in sequenza. Il comando 'destroy_bitmap(immagine e buffer);' ripulisce la variabile immagine e buffer azzerando il loro contenuto dalla memoria prima di uscire dal programma.

04 - Lo sprite, entità in movimento...

```
/* LISTATO C/C++ DI SPRITE
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
*/
#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA
#include “allegro.h” /* INCLUSA LA LIBRERIA ALLEGRO.H */
BITMAP *sprite, *background, *buffer; // DICHIARAZIONE VARIABILI //
PALLETE colori; // DICHIARAZIONE VARIABILI //
int x; // DICHIARAZIONE VARIABILI //
int y; // DICHIARAZIONE VARIABILI //
```

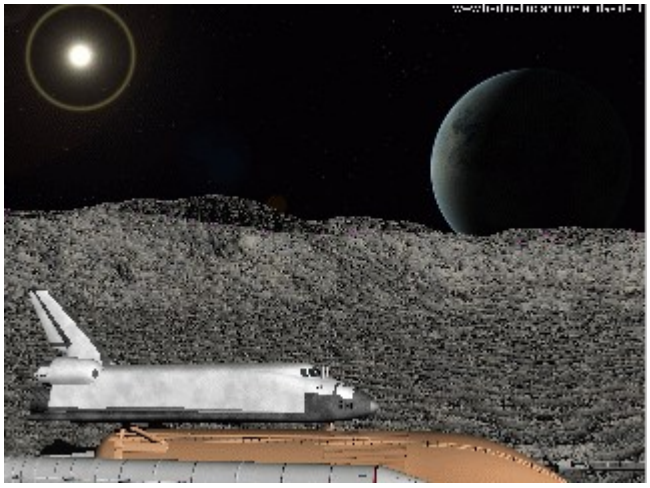
```
void doppiobuffering() // INIZIO PROCEDURA DI DOPPIO BUFFERING //
{
vsync(); // SINCRONIZZAZIONE AGGIORNAMENTO SCHERMO //
blit(buffer, screen, 0, 0, 0, 0, 640, 480); // VISUALIZZA IL CONTENUTO DI BUFFER SU SCHERMO
//
clear(buffer); // PULISCE IL BUFFER //
}
int main() // INIZIO PROCEDURA DI ALLEGRO //
{
allegro_init(); // INIZIALIZZA ALLEGRO //
install_keyboard(); // INSTALLA LA TASTIERA //
set_color_depth(32); // SELEZIONA LA PROFONDITA' DI COLORE A 32 BIT //
set_gfx_mode(GFX_GDI, 640, 480, 0, 0); // SETTA LA MODALITA' GRAFICA COME GDI SU UNA
FINESTRA DI 640x480 PIXEL //
sprite = load_bitmap("sprite.bmp",colori); // CARICA IL FILE SPRITE.BMP DOVE IL VIOLETTO IN
MODALITA' TRUE COLOR E TRAPARENTE (ROSSO=255, VERDE=0, BLU=255) MENTRE PER
LA MODALITA A 256 COLORI IL TRASPARENTE E IL NERO (R=0, V=0, B=0) //
background = load_bitmap("background.bmp",colori); // CARICA IL FILE BACKGROUND.BMP //
buffer = create_bitmap(640, 480); // CREA UN BUFFER BITMAP DI 640x480 PIXEL //
clear(buffer); // CANCELLA IL BUFFER //
x=220; // CENTRA LO STRITE //
y=140; // CENTRA LO SPRITE //
while (!key[KEY_SPACE]) { // CREA UN CICLO INFINO FINO ALLA PRESSIONE DEL TASTO DI
SPAZIO //
blit(background, buffer, 0, 0, 0, 0, 640, 480); // VISUALIZZA L'IMMAGINE DI BACKGROUND SUL
BUFFER //
draw_sprite(buffer, sprite, x, y); // VISUALIZZA LO SPRITE SUL BUFFER //
doppiobuffering(); // CARICA LA PROCEDURA DI DOPPIO BUFFERING //
if (key[KEY_UP]) y=y-3; // MOVIMENTO SPRITE IN ALTO ALLA PRESSIONE DEL TASTO
CURSORE UP //
if (key[KEY_DOWN]) y=y+3; // MOVIMENTO SPRITE IN BASSO ALLA PRESSIONE DEL TASTO
CURSORE DOWN //
if (key[KEY_LEFT]) x=x-3; // MOVIMENTO SPRITE A SINISTRA ALLA PRESSIONE DEL TASTO
CURSORE LEFT //
if (key[KEY_RIGHT]) x=x+3; // MOVIMENTO SPRITE A DESTRA ALLA PRESSIONE DEL
TASTO CURSORE RIGHT //
}
destroy_bitmap(sprite); // DISTRUGGE IL BITMAP //
destroy_bitmap(background); // DISTRUGGE IL BITMAP //
destroy_bitmap(buffer); // DISTRUGGE IL BITMAP //
allegro_exit(); // TERMINA LA LIBRERIA ALLEGRO //
}
END_OF_MAIN (); // FINE DEL PROGRAMMA //
```

Analizziamo le parti salienti del programma 'sprite.c'

Lo sprite nel mondo dei videogiochi rappresenta un'entità “separata dal resto” dello schermo, capace di movimento ed interazione sia con l'utente sia con molti eventi della scena di gioco.
I questo esempio specifico è presente un immagine di sfondo (background.bmp) e una che rappresenta lo sprite (sprite.bmp). Lo sprite in questo programma è rappresentato da una navetta che possiamo comandare a nostro piacimento.
Avremo perciò i soliti comandi presenti negli altri listati, con l'aggiunta di ' draw_sprite(buffer, sprite, x, y);' che permette al computer di disegnare nel buffer lo sprite in una posizione qualsiasi (x,y), che sarà poi visualizzato su schermo con il doppio buffering. All'interno del ciclo 'while...', inltre sono inseriti alcuni comandi necessari ad impartire l'input direzionale da parte dell'utente: 'if (key[KEY_UP]) y=y-3;', 'if (key[KEY_DOWN]) y=y+3;', 'if (key[KEY_LEFT]) x=x-3;', 'if (key[KEY_RIGHT]) x=x+3;'.

Il comando 'draw_sprite...' oltre a visualizzare uno sprite determina o meno la sua trasparenza. Infatti nella modalità true color il colore che determina la trasparenza è il violetto (ROSSO=255,VERDE=0, BLU=255), mentre per quella a 256 colori è il nero (R=0, V=0, B=0).
Da notare il particolare comportamento degli sprite. Mano a mano che vengono visualizzati su schermo, i primi a essere disegnati saranno quelli ad essere coperti dai successivi. Attenzione quindi!
Nel movimento della navicella in cui si identifica lo sprite si deve porre particolare attenzione a non 'attraversare' i bordi dello schermo. In quanto per in questione di semplicità didattica non è stato inserito alcun parametro che eviti questo bug.

05 - Simuliamo l'orizzonte con il parallelasse.



```
/* LISTATO C/C++ DI PARALLELASSE
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
*/

#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA
#include "allegro.h" /* CARICA LA LIBRERIA ALLEGRO.H */

BITMAP *immagine1, *immagine2, *immagine3, *immagine4, *immagine5, *buffer2; /*
DICHIARAZIONE VARIABILI DEI BITMAP */
/* DICHIARAZIONE VARIABILE IMMAGINE */
BITMAP *strato1, *strato2, *strato3, *strato4, *strato5; /* DICHIARAZIONE VARIABILI DEI
BITMAP */
PALETTE colori; /* DICHIARAZIONE VARIABILE COLORI */
int a, b, c, d, e, sposta, sposta1, sposta2, sposta3, sposta4, sposta5; /* DICHIARAZIONE VARIABILI
DEGLI INTERI */

void doppiobuffering() // INIZIO PROCEDURA DI DOPPIO BUFFERING //
{
vsync(); // SINCRONIZZAZIONE AGGIORNAMENTO SCHERMO //
blit(buffer2, screen, 0, 0, 0, 0, 640, 480); // VISUALIZZA IL CONTENUTO DI BUFFER SU
SCHERMO //
clear(buffer2); // PULISCE IL BUFFER //
}
int main() /* INIZIO PROCEDURA */
{
```

```
allegro_init(); /* INIZIALIZZA LA LIBRERIA ALLEGRO */
install_keyboard(); /* INSTALLA LA TASTIERA */
set_color_depth(32); /* IMPOSTA LA PALETTE DEI COLORI A 32 BIT */
set_gfx_mode(GFX_GDI, 640, 480, 0, 0); /* IMPOSTA LA MODALITA' GRAFICA 640x480 */

immagine1=load_bitmap("parallelasse1.bmp", colori); /* CARICA L'IMMAGINE IN FORMATO BMP
*/
immagine2=load_bitmap("parallelasse2.bmp", colori); /* CARICA L'IMMAGINE IN FORMATO BMP
*/
immagine3=load_bitmap("parallelasse3.bmp", colori); /* CARICA L'IMMAGINE IN FORMATO BMP
*/
immagine4=load_bitmap("parallelasse4.bmp", colori); /* CARICA L'IMMAGINE IN FORMATO BMP
*/
immagine5=load_bitmap("parallelasse5.bmp", colori); /* CARICA L'IMMAGINE IN FORMATO BMP
*/
buffer2 = create_bitmap(640, 480); // CREA UN BUFFER BITMAP DI 640x480 PIXEL //
clear(buffer2); /* PULISCE IL BUFFER */
strato5 = create_bitmap(640, 480); // CREA UN BUFFER BITMAP DI 640x480 PIXEL //
clear(strato5); /* PULISCE IL BUFFER */
strato1 = create_bitmap(3840, 480); // CREA UN BUFFER BITMAP DI 3840x480 PIXEL //
clear(strato1); /* PULISCE IL BUFFER */
strato2 = create_bitmap(3840, 480); // CREA UN BUFFER BITMAP DI 3840x480 PIXEL //
clear(strato2); /* PULISCE IL BUFFER */
strato3 = create_bitmap(3840, 480); // CREA UN BUFFER BITMAP DI 3840x480 PIXEL //
clear(strato3); /* PULISCE IL BUFFER */
strato4 = create_bitmap(3840, 480); // CREA UN BUFFER BITMAP DI 3840x480 PIXEL //
clear(strato4); /* PULISCE IL BUFFER */
blit(immagine5, strato5, 0, 0, 0, 0, 640, 480);

for (d=0;d<=3840;d=d+1280) { /* MOLTIPLICA IL BITMAP PER TUTTA LA LUNGHEZZA DELLA
PARTE DA SCROLLARE */
blit(immagine4, strato4, 0, 0, d, 0, 1280, 480);
}
for (c=0;c<=3840;c=c+1280) { /* MOLTIPLICA IL BITMAP PER TUTTA LA LUNGHEZZA DELLA
PARTE DA SCROLLARE */
blit(immagine3, strato3, 0, 0, c, 0, 1280, 480);
}
for (b=0;b<=3840;b=b+1280) { /* MOLTIPLICA IL BITMAP PER TUTTA LA LUNGHEZZA DELLA
PARTE DA SCROLLARE */
blit(immagine2, strato2, 0, 0, b, 0, 1280, 480);
}

for (a=0;a<=3840;a=a+640) { /* MOLTIPLICA IL BITMAP PER TUTTA LA LUNGHEZZA DELLA
PARTE DA SCROLLARE */
blit(immagine1, strato1, 0, 0, a, 0, 640, 480);
}

while (!key[KEY_SPACE]) { // CREA UN CICLO INFINO FINO ALLA PRESSIONE DEL TASTO DI
SPAZIO //

if (key[KEY_LEFT]) { // ATTENDE LA PRESSIONE DEL TASTO CURSORE //
sposta1=sposta1+5; sposta2=sposta2+4; sposta3=sposta3+3; sposta4=sposta4+2; sposta5=sposta5+1;
// SCROLLING DEI LIVELLI DI PARALLELASSE //
}
if (key[KEY_RIGHT]) { // ATTENDE LA PRESSIONE DEL TASTO CURSORE //
sposta1=sposta1-5; sposta2=sposta2-4; sposta3=sposta3-3; sposta4=sposta4-2; sposta5=sposta5-1;
```

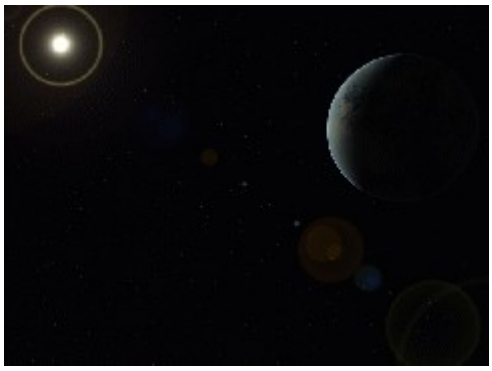
```
// SCROLLING DEI LIVELLI DI PARALLELASSE //
}
// BLOCCA LO SPOSTAMENTO DELLO SCHERMO //

if (sposta5>=0) {
sposta1=sposta1-5; sposta2=sposta2-4; sposta3=sposta3-3; sposta4=sposta4-2; sposta5=sposta5-1;
}
if (sposta5<=-640) {
sposta1=sposta1+5; sposta2=sposta2+4; sposta3=sposta3+3; sposta4=sposta4+2; sposta5=sposta5+1;
}
/* VISUALIZZA L'IMMAGINE SUL DOPPIO BUFFERING - in questa fase è necessario ricordarsi che
le prime immagini visualizzate saranno
anche le prime ad essere coperte dalle successive - */
draw_sprite(buffer2, strato5, 0, 0); // CARICA E SPOSTA IL LIVELLO DI PARALLELASSE - in
questo caso la prima immagine sarà anche quella più lontana e perciò non sarà mossa -//
draw_sprite(buffer2, strato3, sposta3, 183); // CARICA E SPOSTA IL LIVELLO DI
PARALLELASSE //
draw_sprite(buffer2, strato2, sposta2, 200); // CARICA E SPOSTA IL LIVELLO DI
PARALLELASSE //
draw_sprite(buffer2, strato1, sposta1, 448); // CARICA E SPOSTA IL LIVELLO DI
PARALLELASSE //
draw_sprite(buffer2, strato4, sposta4, 285); // CARICA E SPOSTA IL LIVELLO DI
PARALLELASSE //
doppiobuffering(); // CARICA LA PROCEDURA DI DOPPIO BUFFERING //
}

destroy_bitmap(immagine1); /* ELIMINA L'IMMAGINE DAL BUFFER PRIMA DI USCIRE DAL
PROGRAMMA */
destroy_bitmap(immagine2); /* ELIMINA L'IMMAGINE DAL BUFFER PRIMA DI USCIRE DAL
PROGRAMMA */
destroy_bitmap(immagine3); /* ELIMINA L'IMMAGINE DAL BUFFER PRIMA DI USCIRE DAL
PROGRAMMA */
destroy_bitmap(immagine4); /* ELIMINA L'IMMAGINE DAL BUFFER PRIMA DI USCIRE DAL
PROGRAMMA */
destroy_bitmap(immagine5); /* ELIMINA L'IMMAGINE DAL BUFFER PRIMA DI USCIRE DAL
PROGRAMMA */
allegro_exit(); /* TERMINA LA LIBRERIA ALLEGRO */
}
END_OF_MAIN (); /* FINE DEL PROGRAMMA */
```

Analizziamo le parti salienti del programma PARALLELASSE.C

Il requisito indispensabile per utilizzare la libreria 'allegro.h' come al solito è quello di inserire la riga '#include <allegro.h>...' seguita da 'allegro_init();...' e per chiudere 'allegro_exit();...' ed alla fine dell'intero programma 'END_OF_MAIN ();'. Vengono inizializzate tutte le variabili. Prendiamo in esame la procedura di doppio 'buffering' in cui vengono visualizzate le immagini su schermo 'void doppiobuffering() { ... }'. In cui 'vsync()' sincronizza l'aggiornamento dello schermo. 'Blit(...)' prende il contenuto del buffer e lo visualizza su schermo con 'screen', a questo punto il buffer viene cancellato con 'clear(...)' per poi essere rivisualizzato. Il secondo passo e quello di inserire l'uso della tastiera, quindi digitare 'install_keyboard();...' a questo punto il comando 'readkey();' permette la lettura di un carattere da tastiera e quindi ne attende la pressione. Adesso con il comando 'set_color_depth(32);' viene setta la profondità di colore dell'immagine, che in questo caso è di 32 bit ma i parametri accettati sono anche 16 bit e 8 bit(in questo caso ricordo che deve essere impostata la palette da usare). Non ci resta che settare la modalità grafica da usare con il comando 'set_gfx_mode(GFX_GDI, 640, 480, 0, 0);' che seleziona la modalità automaticamente, altrimenti si può scegliere tra directx, windowed, ecc... Subito dopo si assegnano le dimensioni in pixel dello schermo o della finestra da visualizzare di 640x480 pixel.

Cerchiamo adesso di capire come viene caricata l'immagine in memoria: tramite il comando 'immagin1=load_bitmap("1.tga", colori);' viene caricato il file 'parallelasse1.bmp' (potrebbe anche essere un *.tga o *.pcx o *.lbm), così il contenuto del file viene trasferito alla variabile 'immagine1' e con la variabile 'colori' viene assegnata la palette. A questo punto con 'buffer2 = create_bitmap(...)' si crea la dimensione del bitmap da usare per la visualizzazione degli strati di parallelasse o di buffer. Con il comando 'for { ... }' si moltiplica il bitmap per creare tutti i livelli di parallelasse su tutta la lunghezza dell' area da scrollare e il risultato viene inserito nel buffer 'stratoX'. Creiamo un ciclo infinito 'while { ... }' fino alla pressione del tasto di spazio. All'interno del ciclo inseriamo delle condizioni 'if' per determinare il movimento con la pressione dei tasti cursore sinistro e destro. con 'SPOSTAX=SPOSTAX+X' viene determinata la velocità di spostamento degli strati di parallelasse. Le seconde condizioni 'if' determinano la fine dello scorrimento dello schermo bloccandolo. I livelli di parallelasse vengono visti come degli sprite e devono essere visualizzati in ordine di visualizzazione. Da non dimenticare le trasparenze che in questo caso sono in true-color, quindi si settano con il violetto r=255, g=0, b=255. Si carica la procedura di 'doppiobuffering();' che visualizza il tutto su schermo. Vengono distrutti tutti i bitmap dalla memoria con 'destroy_bitmap(immaginex);'

06 - Un comodo file di archivio dati sicuro.



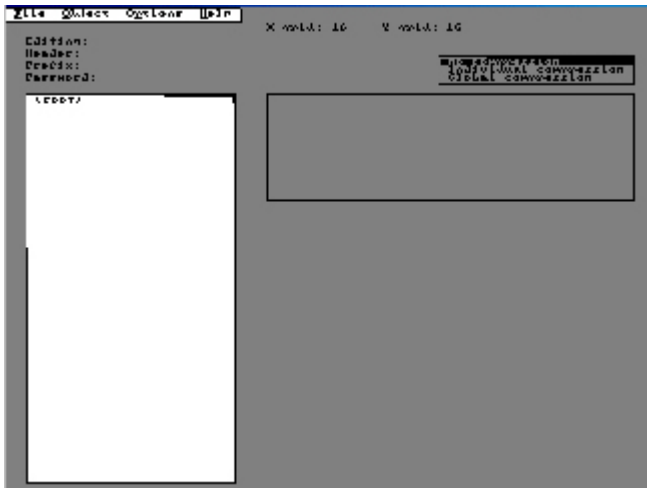
```
/* LISTATO C/C++ DI FILEDAT
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
*/

#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA
#include "allegro.h" /* CARICA LA LIBRERIA ALLEGRO.H */
#include "filedat.h" /* CARICA LA LIBRERIA CHE ABBIAMO CREATO CON IL FILE *.DAT */

DATAFILE *dat; // DICHIARAZIONE VARIABILE DEL FILE DAT //

int main() /* INIZIO PROCEDURA */
{
PALETTE colori; /* DICHIARAZIONE VARIABILE COLORI */
allegro_init(); /* INIZIALIZZA LA LIBRERIA ALLEGRO */
install_keyboard(); /* INSTALLA LA TASTIERA */
set_color_depth(32); /* IMPOSTA LA PALETTE DEI COLORI A 32 BIT */
set_gfx_mode(GFX_GDI, 640, 480, 0, 0); /* IMPOSTA LA MODALITA' GRAFICA 640x480 */
dat = load_datafile("filedat.dat"); // CARICA IL FILE DAT //
blit(dat[immagine1].dat, screen, 0, 0, 0, 0, 640, 480); /* VISUALIZZA L'IMMAGINE1 SULLO
SCHERMO CONTENUTA NEL FILE DAT */
readkey(); /* LEGGE UN TASTO QUALSIASI DA TASTIERA */
clear(screen); // PULISCE LO SCHERMO //
```

```
blit(dat[immagine2].dat, screen, 0, 0, 0, 0, 640, 480); /* VISUALIZZA L'IMMAGINE2 SULLO SCHERMO CONTENUTA NEL FILE DAT */
readkey(); // ATTENDE LA PRESSIONE DI UN TASTO //
clear(screen); /* PULISCE LO SCHERMO */
blit(dat[immagine3].dat, screen, 0, 0, 0, 0, 640, 480); /* VISUALIZZA L'IMMAGINE3 SULLO SCHERMO CONTENUTA NEL FILE DAT */
readkey(); /* LEGGE UN TASTO QUALSIASI DA TASTIERA */
clear(screen); // PULISCE LO SCHERMO //
allegro_exit(); /* TERMINA LA LIBRERIA ALLEGRO */

} END_OF_MAIN ();
```

Analizziamo le parti salienti del programma FILEDAT.C:

Il requisito indispensabile per utilizzare la libreria allegro.h è quello di inserire la riga '#include <allegro.h>...' seguita da 'allegro_init();...' e per chiudere 'allegro_exit();...' con alla fine dell'intero programma 'END_OF_MAIN ();'. Fatto questo si deve caricare un seconda libreria contenente i nostri files, contenuti in questo caso in 'graph.h' con il comando '#include "graph.h"'. Si dichiarano le variabili 'dat', 'palette', ecc... Ora inizializziamo la libreria allegro ed installiamo la tastiera. Adesso con il comando 'set_color_depth(32);' viene setta la profondità di colore dell'immagine, che in questo caso è di 32 bit ma i parametri accettati sono anche 8 bit e 16 bit. Non ci resta che settare la modalità grafica da usare con il comando 'set_gfx_mode(GFX_GDI, 640, 480, 0, 0);' che seleziona la modalità automaticamente altrimenti si può scegliere tra directx, windowed, ecc... Subito dopo si assegnano le dimensioni in pixel dello schermo o della finestra da visualizzare, in questo caso 640x480 pixel. Carichiamo il file '*.DAT' nella variabile 'dat' con il comando 'load_datafile(...)'. Con il comando 'blit(...)' viene caricata tutta la sequenza di immagini che segue visualizzandola su schermo. Il secondo passo e quello di inserire l'uso della tastiera, quindi digitare 'install_keyboard(); ...' ora il comando 'readkey();' permette la lettura di un carattere da tastiera attendendone la pressione. Infine con il comando 'clear(screen);' si pulisce lo schermo. questi ultimi tre comandi si ripetono per il caricamento di ogni immagine.

Creiamo il 'filedat.h' con GRABBER:



Il programma GRABBER.EXE lo troviamo in: C:\ALLEGRO\ALLEGRO\TOOLS

PS: consiglio di copiarlo nella directory in cui dovete lavorare.

Lanciamo il programma GRABBER.EXE poi nella voce 'Editing:' inseriamo il percorso e il nome del 'filedat.dat' (es: c:\allegro\filedat.dat). Subito sotto troviamo la dicitura 'Header:' dove inseriamo il nome 'filedat.h'. Spostiamoci con il puntatore del mouse sul menù in alto e clicchiamo su 'Object', poi su 'New'

in basso e in fine su 'BITMAP' (ad esempio), qui scriviamo il nome che vogliamo assegnare al bitmap (questo sarà poi anche il nome che verrà usato nel listato per richiamare l'immagine). E' arrivato il momento di assegnare il file all'etichetta che abbiamo creato; quindi selezioniamo la nostra etichetta dall'elenco e con il mouse ripuntiamo il menù 'Object' in alto e così premiamo su 'Grab' e selezioniamo il file(tga, bmp, pcx, lbm) che vogliamo inserire. Fatto questo, per scrupolo, selezioniamo nella finestra a destraGLOBAL COMPRESSION così otterremo un file molto meno spazioso. Così per finire possiamo salvare il file, nel menù 'File' appunto, nella directory che ci interessa, specificando il nome e premere ok.
Il passo successivo è quello di aprire il file.h con il programma 'blocco note' di windows.
All' interno del file.h troviamo una o più righe simili a questa:

```
#define immagine 0 /* BMP */
```

Le dicitura 'immagine' e '0' servono per richiamare il file a cui fanno riferimento. Possiamo scegliere se usare la modalità nominativa o numerica. Quella numerica può servire se decidiamo di animare uno sprite per esempio. Per caricare il file 'dat' nel nostro programma riporto le righe di codice più salienti:

```
#include "filedat.h" /* CARICA LA LIBRERIA CHE ABBIAMO CREATO CON IL FILE *.DAT */

...

DATAFILE *dat; // DICHIARAZIONE VARIABILE DEL FILE DAT //

...

dat = load_datafile("filedat.dat"); // CARICA IL FILE DAT //
...

blit(dat[immagine1].dat, screen, 0, 0, 0, 0, 800, 600); /* VISUALIZZA L'IMMAGINE SULLO SCHERMO CONTENUTA NEL FILE DAT */
...
```

Questo procedimento è uguale per tutti i tipi di file (fli/flc, wav, mid, tga, dat, ecc...)

Di seguito propongo il contenuto del file dat utilizzato per questo esempio:

```
/* Allegro datafile object indexes, produced by grabber v4.2.0, MinGW32 */
/* Datafile: c:\Documents and Settings\Davide\Documenti\libro allegro.h\filedat.dat */
/* Date: Tue May 29 23:19:08 2007 */
/* Do not hand edit! */

#define immagine1 0 /* BMP */
#define immagine2 1 /* BMP */
#define immagine3 2 /* BMP */
```

07 - Impariamo a riprodurre un file audio

```
/* LISTATO C/C++ DI SAMPLE
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
*/

#include <stdio.h> // LIBRERIA DI SISTEMA
```

```
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA
#include <allegro.h> // Carica la libreria allegro.h //

int main() // Inizia la procedura //
{
SAMPLE *suono; // Dichiarazione della variabile del sample //
allegro_init(); // Inizializza la libreria allegro.h //
install_keyboard(); // Installa la tastiera //
install_timer(); // Installa il timer per temporizzare i suoni //
install_sound(DIGI_AUTODETECT, MIDI_AUTODETECT, 0); // Seleziona la scheda audio e il midi, il numerino e necessario solo per le vecchie versioni di allegro.h e se da 0 e tutto ok e se da -1 indica errore//
suono = load_sample("sample.wav"); // Carica il file audio .wav //
set_volume(255,255); // Setta il volume dei canali stereo destro e sinistro //
play_sample(suono, 255,128,1000, FALSE); // manda in esecuzione il il contenuto della variabile sample, volume, posizione, frequenza, TRUE o FALSE indica il loop //
readkey(); // Attende la pressione di un tasto //
destroy_sample(suono); // Distrugge il sample //
}
END_OF_MAIN ();
```

Analizziamo le parti salienti del programma SAMPLE.C:

Il requisito indispensabile per utilizzare la libreria allegro.h e quello di inserire la riga '#include <allegro.h>...' seguita da 'allegro_init();...' e per chiudere 'allegro_exit();...' ed alla fine dell'intero programma 'END_OF_MAIN ();'. Il secondo passo è la dichiarazione delle variabili 'SAMPLE' che in questo caso è '*suono'. Successivamente si inserisce l'uso della tastiera, quindi digitare 'install_keyboard();...' a questo punto il comando 'readke();' permette la lettura di un carattere da tastiera e quindi ne attende la pressione. Oltre che il 'timer' per temporizzare i suoni. Ora installiamo la scheda sonora 'install_sound(...)', selezionando la modalità di rilevazione automatica sia per i sample che per i midi. Arrivati a questo punto possiamo caricare il file midi con 'suono = load_sample(...)' dove il contenuto si trasferisce nella variabile 'suono'. Impostiamo il volume dei due canali audio destro e sinistro (255 è il max) con set_volume(x,x). Mandiamo in esecuzione la musica con il comando 'play_sample(...)', i numeri in questo ordine indichiamo volume, posizione, frequenza e con l'opzione FALSE o TRUE si decide se creare si o no un loop infinito. Con 'readkey()' il computer attende la pressione di un tasto. Il comando 'destroy_sample(...);' ripulisce la variabile azzerandola del suo contenuto prima di uscire dal programma.

08 - Carichiamo ed eseguiamo un brano musicale MIDI

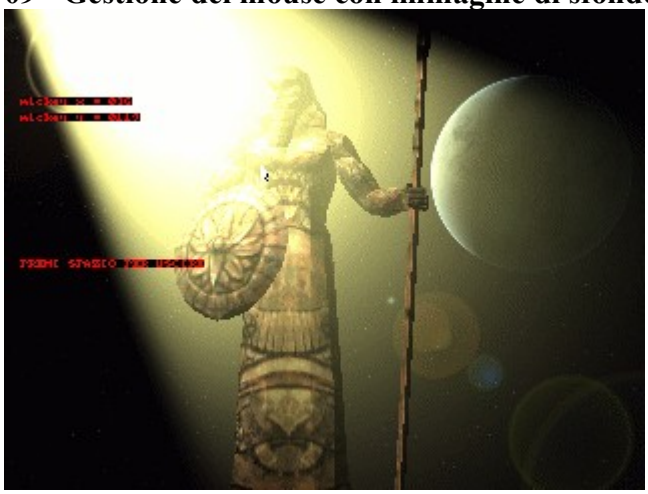
```
/* LISTATO C/C++ DI MIDI
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
*/

#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA
#include <allegro.h> // Carica la libreria allegro.h //

int main() // Inizia la procedura //
{
```

```
MIDI *musica; // Dichiarazione della variabile del MIDI //
allegro_init(); // Inizializza la libreria allegro.h //
install_keyboard(); // Installa la tastiera //
install_timer(); // Installa il timer per temporizzare i suoni //
install_sound(DIGI_AUTODETECT, MIDI_AUTODETECT, 0); // Seleziona la scheda audio e il midi, il numerino e necessario solo per le vecchie versioni di allegro.h e se da 0 e tutto ok e se da -1 indica errore//
musica = load_midi("midi.mid"); // Carica il file midi .mid //
set_volume(255,255); // Setta il volume dei canali stereo destro e sinistro //
play_midi(musica, FALSE); // manda in esecuzione il il contenuto della variabile musica, TRUE o FALSE indica il loop //
readkey(); // Attende la pressione di un tasto //
destroy_midi(musica); // Distrugge il midi //
allegro_exit();
}
END_OF_MAIN ();
```

Analizziamo le parti salienti del programma MIDI.C

Il requisito indispensabile per utilizzare la libreria 'allegro.h' è quello di inserire la riga '#include <allegro.h>...' seguita da 'allegro_init(); ...' e per chiudere 'allegro_exit(); ...' ed alla fine dell'intero programma 'END_OF_MAIN ();'. Il secondo passo è la dichiarazione della variabile MIDI che in questo caso è '*musica'. Successivamente si inserisce l'uso della tastiera, quindi digitare 'install_keyboard(); ...' a questo punto il comando 'readkey();' permette la lettura di un carattere da tastiera e quindi ne attende la pressione. Oltre che il timer per temporizzare i suoni. Ora installiamo la scheda sonora 'install_sound(...)', selezionando la modalità di rilevazione automatica sia per i sample che per i midi. Arrivati a questo punto possiamo caricale il file midi con 'musica = load_midi(...)' dove il contenuto si trasferisce nella variabile 'musica'. Impostiamo il volume dei due canali audio destro e sinistro (255 è il max) con 'set_volume(x,y)'. Mandiamo in esecuzione la musica con il comando 'play_midi(...)' e con l'opzione 'FALSE' o 'TRUE' si decide se creare si o no un loop infinito. Con 'readkey()' il computer attende la pressione di un tasto. Il comando 'destroy_midi(...);' ripulisce la variabile azzerandola del suo contenuto prima di uscire dal programma.

09 - Gestione del mouse con immagine di sfondo: tasti e movimenti



```
/* LISTATO C/C++ DI MOUSE
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
*/
```

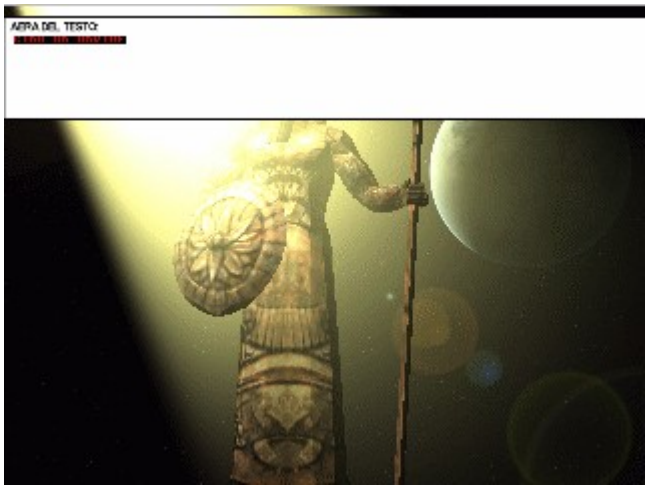
```
#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA
#include "allegro.h" /* CARICA LA LIBRERIA ALLEGRO.H */

BITMAP *immagine; /* DICHIARAZIONE VARIABILE IMMAGINE */
PALETTE colori; /* DICHIARAZIONE VARIABILE COLORI */

int main() /* INIZIO PROCEDURA */
{
int x,y; /* DICHIARAZIONE VARIABILI INTERE PER LE COORDINATE */
char msg[80]; /* DICHIARAZIONE DI UNA STRINGA LUNGA MAX 80 CARATTERI */

allegro_init(); /* INIZIALIZZA LA LIBRERIA ALLEGRO */
install_keyboard(); /* INSTALLA LA TASTIERA */
install_mouse(); /* INIZIALIZZA IL MOUSE */
install_timer(); /* INIZIALIZZA IL TIMER */

set_color_depth(32); /* IMPOSTA LA PALETTE DEI COLORI A 32 BIT */
set_gfx_mode(GFX_GDI, 640, 480, 0, 0); /* IMPOSTA LA MODALITA' GRAFICA 640x480 */

immagine=load_bitmap("testo.bmp", colori); /* CARICA IL FILE TESTO.BMP CHE SARA' USATA
COME SFONDO*/

blit(immagine, screen, 0, 0, 0, 0, 640, 480); /* VISUALIZZA L'IMMAGINE SULLO SCHERMO */
show_mouse(screen); /* MOSTRA IL PUNTATORE DEL MOUSE SU SCHERMO */

textout(screen, font, "PREMI SPAZIO PER USCIRE", 16, 250, 255); /* MOSTRA UN MESSAGGIO
SU SCHERMO */

while (!key[KEY_SPACE]) { /* CICLO WHILE FINO ALLA PRESSIONE DEL TASTO SPAZIO */
if (mouse_b & 1) { /* CONDIZIONE SE VIENE PREMUTO IL TASTO SINISTRO DEL MOUSE */
textout(screen, font, "Tasto sinistro", 100, 150, 255); /* VISUALIZZA UN MESSAGGIO */
}

if (mouse_b & 2) { /* CONDIZIONE SE VIENE PREMUTO IL TASTO DESTRO DEL MOUSE */
textout(screen, font, "Tasto destro", 100, 175, 255); /* VISUALIZZA UN MESSAGGIO SU SCHERMO
*/
}

get_mouse_mickeys(&x, &y); /* AVVERTE GLI SPOSTAMENTI DEL MOUSE SULL' ASSE X E Y
IN ACCELERAZIONE */

sprintf(msg, "mickey_x = %ld", x); /* INSERISCE UN MESSAGGIO NELLA STRINGA MSG */
textout(screen, font, msg, 16, 88, 255); /* VISUALIZZA LA STRINGA MSG IN UNA DATA
POSIZIONE CON UN CERTO COLORE */

sprintf(msg, "mickey_y = %ld", y); /* INSERISCE UN MESSAGGIO NELLA STRINGA MSG */
textout(screen, font, msg, 16, 104, 255); /* VISUALIZZA LA STRINGA MSG IN UNA DATA
POSIZIONE CON UN CERTO COLORE */

}

allegro_exit(); /* TERMINA LA LIBRERIA ALLEGRO */
}
```

END_OF_MAIN (); /* FINE DEL PROGRAMMA */

Analizziamo le parti salienti del programma MOUSE.C:

Il requisito indispensabile per utilizzare la libreria allegro.h è quello di inserire la riga '#include <allegro.h> ...' seguita da 'allegro_init(); ...' e per chiudere 'allegro_exit(); ...' ed alla fine dell'intero programma 'END_OF_MAIN ();'. Si dichiarano le variabili delle stringhe(lunghezza max 80 caratteri) e interi; Qui ho installato la tastiera il mouse e il temporizzatore. Questo è un passo importante per ricevere gli input dall'esterno. Adesso con il comando 'set_color_depth(32);' viene settata la profondità di colore dell'immagine, che in questo caso è di 32 bit ma i parametri accettati sono anche 16 bit e 8 bit(in questo caso come al solito deve essere impostata la palette da usare). Non ci resta che settare la modalità grafica da usare con il comando 'set_gfx_mode(GFX_GDI, 640, 480, 0, 0);' che in questo caso seleziona la modalità automaticamente, altrimenti si può scegliere tra directx, windowed, ecc... Subito dopo si danno le dimensioni in pixel dello schermo o della finestra da visualizzare, in questo caso 800x600 pixel. Con 'show_mouse(screen);' visualizziamo il puntatore del mouse su schermo(che per ora non gestirà alcun evento!). Utilizzando l'ordine 'textout(x,x,x,x,x,x)' visualizziamo dei caratteri su schermo o nel buffer patendo da una variabile di tipo stringa o tra virgolette "XXX" in una data posizione (x,y) con un dato colore da 0 a 255 per la modalità a 8 bit, mentre con 'makecol(r, g, b)' per la modalità true color. Ho inserito un ciclo 'while' infinito fino alla pressione del tasto di 'spazio'. Qui iniziano le condizioni di pressione dei due tasti del mouse (mouse_b & 1), per il tasto di SX e per il tasto DX (mouse_b & 2). Digitando 'get_mouse_mickeys(&x, &y);' si da modo di 'capire' al computer se c'è stato si o no uno spostamento del mouse sull'asse x o y. Ora arriva il momento di visualizzare su schermo i valori di spostamento di x e y. Prima di arrivare a ciò si devono convertire in stringhe i valori di x e y che sono interi con il comando 'sprintf(stringa, "messaggio", intero);' in cui tutto il contenuto viene trasferito alla stringa.
PS: %ld indica la posizione in cui si visualizza l'intero.
In seguito con 'textout(xxxxxx);' viene visualizzato il tutto su schermo nel buffer video. Il comando 'destroy_bitmap(immagine);' ripulisce la variabile immagine azzerandola del suo contenuto prima di uscire dal programma.

10 - Editare del testo in modalità grafica



/* LISTATO C/C++ DI TESTO
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
*/

```
// CARICO LE LIBRERIE

#include "allegro.h" /* CARICA LA LIBRERIA ALLEGRO.H */
#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA

BITMAP *immagine; /* DICHIARAZIONE VARIABILE IMMAGINE */
PALETTE colori; /* DICHIARAZIONE VARIABILE COLORI */

// VARIABILE PER IL DOPPIO BUFFERING

BITMAP *buf;

// STRINGA DEL CARATTERE DEL TASTO PREMUTO

char carattere[1];

// CONTIENE IL TOLALE DEI TASTI PREMUTI IN QUESTA VARIABILE (ES: C+I+A+O)
// MAX OTTANTA CARATTERI

char testo[80];

// CREO UN PUTATATORE CHE CONTERRA' LA STRINGA DA INSERIRE NEL
DOPPIOBUFFERING

char *tottesto;

// EVITA CHE SIANO VISUALIZZATI TROPPI CARATTERI CON UNA SOLA PRESSIONE

int rallentatore;

// POSIZIONE IN PIXEL DELLA SCRITTA SU SCHERMO

int x=10, y=30;

// CREO LA PROCEDURA PER IL DOPPIO BUFFERING

void doppiobuffering(void)
{

// SINCRONIZZO IL SEGNALE DI AGGIORNAMENTO DELLO SCHERMO

vsync();

// TRAFERISCO IL CONTENUTO DEL BUFFER SULLO SCHERMO, VISUALIZZANDOLO

blit(buf, screen, 0, 0, 0, 0, 640, 480);

// RIPULISCO IL BUFFER

clear(buf);


}
```

```
// CREO LA PROCEDURA DI SCRITTURA

void scrivere(void)
{

/* PROIETTA L'IMMAGINE NEL BUFFER */

blit(immagine, buf, 0, 0, 0, 0, 640, 480);

// CONTATORE CHE RALLENTA LA VISUALIZZAZIONE

rallentatore++;

// ASSEGNO IL CARATTERE PREMUTO ALLA VARIABILE STRINGA 'CARATTERE'


if (key[KEY_A]) strcpy(carattere,"A");
if (key[KEY_B]) strcpy(carattere,"B");
if (key[KEY_C]) strcpy(carattere,"C");
if (key[KEY_D]) strcpy(carattere,"D");
if (key[KEY_E]) strcpy(carattere,"E");
if (key[KEY_F]) strcpy(carattere,"F");
if (key[KEY_G]) strcpy(carattere,"G");
if (key[KEY_H]) strcpy(carattere,"H");
if (key[KEY_I]) strcpy(carattere,"I");
if (key[KEY_L]) strcpy(carattere,"L");
if (key[KEY_M]) strcpy(carattere,"M");
if (key[KEY_N]) strcpy(carattere,"N");
if (key[KEY_O]) strcpy(carattere,"O");
if (key[KEY_P]) strcpy(carattere,"P");
if (key[KEY_Q]) strcpy(carattere,"Q");
if (key[KEY_R]) strcpy(carattere,"R");
if (key[KEY_S]) strcpy(carattere,"S");
if (key[KEY_T]) strcpy(carattere,"T");
if (key[KEY_U]) strcpy(carattere,"U");
if (key[KEY_V]) strcpy(carattere,"V");
if (key[KEY_Z]) strcpy(carattere,"Z");
if (key[KEY_SPACE]) strcpy(carattere, " ");

// QUANDO IL CONTATORE DI RALLENTAMENTO E' MAGGIORE OD UGUALE A 15
ALLORA...

if (rallentatore>=15) {

// ...AGGIUNGE UN NUOVO CARATTERE PREMUTO A QUELLI GIA' DIGITATI IN 'TESTO'

strcat(testo,carattere);

// FA RIPARTIRE DA ZERO IL CONTEGGIO DI RALLENTAMENTO

rallentatore=0;
}

// TRASFORMA LA STRINGA CONTENENTE I CARATTERI DIGITATI IN UN PUNTATORE
STRINGA
```

```
// 'TOTTESTO'

tottesto=testo;

// VISUALIZZO NEL BUFFER I CARATTERI PREMUTI MEMORIZZATI NEL PUNTATORE 'TOTTESTO'

textout(buf,font,tottesto,x,y,215);

// SE NON VIENE PREMUTO ALCUN TASTO 'CARATTERE' ASSUME IL VALORE DI NESSUN CARATTERE

strcpy(carattere,"");

}


// INIZIA IL CORPO DEL PROGRAMMA

int main()
{

// INIZIALIZZO LA LIBRERIA ALLEGRO.H

allegro_init();

// INSTALLO LA TASTIERA

install_keyboard();

/* IMPOSTA LA PALETTE DEI COLORI A 32 BIT */

set_color_depth(32);

// DETERMINO LA MODALITA' GRAFICA

set_gfx_mode(GFX_GDI, 640, 480, 0, 0);

/* CARICA IL FILE TESTO2.BMP */

immagine=load_bitmap("testo2.bmp", colori);

// CREO LA DIMENSIONE IN PIXEL DEL BUFFER

buf = create_bitmap(640, 480);

// PULISCO IL BUFFER

clear(buf);

// INIZIO UN CICLO INFINITO, FINO ALLA PRESSIONE DEL TASTO 'ESC'

while (!key[KEY_ESC]) {


// RICHIAMO LA PROCEDURA DI DOPPIO BUFFERING
```

```
doppiobuffering();

// RICHIAMO LA PROCEDURA DI VISUALIZZAZIONE DEL CARATTERE PREMUTO

scrivere();
}

// DISTRUGGO DALL MEMORIA LA VARIABILE BITMAP 'BUF'

destroy_bitmap(buf);

}

// CHIUDO IL PROGRAMMA

END_OF_MAIN ();
```

Analizziamo le parti salienti del programma TESTO.C:

Dopo aver caricato le librerie necessarie ed inizializzato le variabili, creo la solita routine per il doppio buffering. Fatto questo creo una seconda procedura che mi permetterà di inserire in una stringa il contenuto del tasto corrispondente che è stato premuto(es. premo 'a' ed in memoria si archivia la 'a') - comando 'strcopy(...)'. Inoltre in questa procedura controllo che non vengano memorizzati troppi caratteri per singola pressione del tasto nella variabile, grazie alla condizione del rallentatore - 'if (rallentatore<=15) ...' . Con il comando - 'strcat(...)' aggiungo ad una certa stringa (testo) i singoli caratteri premuti per formare delle parole. Per poter visualizzare il tasto premuto con 'textout(...)' è necessario convertire la stringa in un puntatore (tottesto=testo). In seguito vengono riportati i comandi obbligatori per attivare la modalità grafica e viene creato il ciclo 'while(...)' con tutte le procedure all'interno.

11 - Far funzionare il joystick

```
// ESEMPIO DI CREAZIONE DI MOVIMETO JOYSTICK //
// SVILUPPATO DA BARTOLOMEO DAVIDE BERTINETTO //
// www.bertinettobartolomeodavide.it

#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA
```

```
// carico la libreria allegro.h
#include "allegro.h"

// inizializzo la variabile del buffer
BITMAP *buf;

// inizializzo la palette
PALETTE colori;

/* dichiaro variabili di posizione x e y e
che è la variabile di posizione iniziale dei puntini.
*/
int x,y,start=10;

// creo la procedura per il doppio buffering
void doppiobuffering(void)
{
// sincronizzo l'aggiornamento dello schermo
vsync();

// disegno il contenuto del buffer sullo schermo
blit(buf, screen, 0, 0, 0, 0, 640, 480);

// ripulisco il contenuto del buffer dopo averlo visualizzato
clear(buf);
}

// creo la procedura per visualizzare un cerchio pieno su schermo
void pixel(void) {

// Disegno il cerchio pieno nel buffer dove 10 è il raggio
circlefill(buf, x, y, 10, 255);
}

// creo la procedura che contiene la routine del movimento col joystick
void movimento(void) {

// SEGNALO ALL'UTENTE CHE E' POSSIBILE USARE UNA PERIFERICA DI GIOCO
textout(buf, font, "USA IL JOYPAD O IL JOYSTICK!!!", 200, 470, 215);

// legge il joystick
poll_joystick();

// seleziona il tasto uno del joy per il fuoco (pixel sparito)

/* I parametri inseriti nella condizione if(...) servono ad indicare:
-joy[n] determina quale periferica di gioco è da utilizzate in caso
ne esista più di una.
-button[n] determina il tasto da utilizzare
*/
if (joy[0].button[0].b) circlefill(buf, x, y, 9, 0);

// movimento pixel a sinistra

/* I parametri inseriti nella condizione if(...) servono ad indicare:
```

```
-joy[n] determina quale periferica di gioco è da utilizzate in caso
ne esista più di una.
-stick[n] determina quale leva dello stesso joystick è da utilizzare
nel caso ne esista più di una
-axis[n] determina l'asse di spostamento della leva che ci interessa
(esistono due assi: X e Y o 0 e 1)
-d1 e d2 indicano la direzione nell'asse selezionato (esistono solo 2
direzioni andata(d1) e ritorno(d2))
*/
if (joy[0].stick[0].axis[0].d1) x=x-5; if (x<=10) x=10;

// movimento pixel a destra
if (joy[0].stick[0].axis[0].d2) x=x+5; if (x>=629) x=629;

// movimento pixel verso l'alto
if (joy[0].stick[0].axis[1].d1) y=y-5; if (y<=10) y=10;

// movimento pixel verso il basso
if (joy[0].stick[0].axis[1].d2) y=y+5; if (y>=469) y=469;

/* Attezione alla calibrazione del joystick, che deve essere effettuata da Windows
in: PANNELO DI CONTROLLO, PERIFERICHE DI GIOCO, PROPRIETA', IMPOSTAZIONI E
TARATURA...
Se il vostro joystick non è tarato andrà più in una direzione rispetto ad un altra
e probabilmente non troverete mai la condizione di centro (pixel fermo)
*/
}

// creo la procedura principale

int main()
{
// inizializzo la libreria allegro.h
allegro_init();

// installo la tastiera
install_keyboard();

// setto la profondità di colore della palette
set_color_depth(32);

// assegno una variabile alla palette
set_palette(colori);

// determino la risoluzione dello schermo
set_gfx_mode(GFX_GDI, 640, 480, 0, 0);

// installazione joystick
install_joystick(JOY_TYPE_AUTODETECT);

// assegno la dimensione del buffer
buf = create_bitmap(640, 480);

// pulisco il buffer appena creato
clear(buf);
```

```
x=start; y=start;

// inizio del ciclo
while (!key[KEY_ESC]) {

// carico le procedure
doppiobuffering();
pixel();
movimento();
}

// distruggo la varibile buffer
destroy_bitmap(buf);

}

END_OF_MAIN ();
```

Analizziamo le parti salienti del programma JOYSTICK.C:
Vi divertirete nell'inserire l'uso de joystick nei vostri videogiochi. Questa periferica ha segnato un'era per quello che riguarda i videogames su computer. Con allegro è possibile utilizzare qualunque tipo di joypad, anche di ultimissima generazione, visto che il codice è adattabile all'uso di un grande numero di controlli sia di tipo analogico che digitale.
Il codice sopra esposto è ampiamente ampiamente commentato e di facile comprensione. Non faticherete perciò ad adattarlo ai vostri lavori...

12 - Visualizziamo molti sprite con l'array e pochissimo codice.

```
// ESEMPIO DI CREAZIONE DI UN ARRAY GRAFICO //
// SVILUPPATO DA BARTOLOMEO DAVIDE BERTINETTO //
// www.bertinettobartolomeodavide.it

#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA

// carico la libreria allegro.h
```

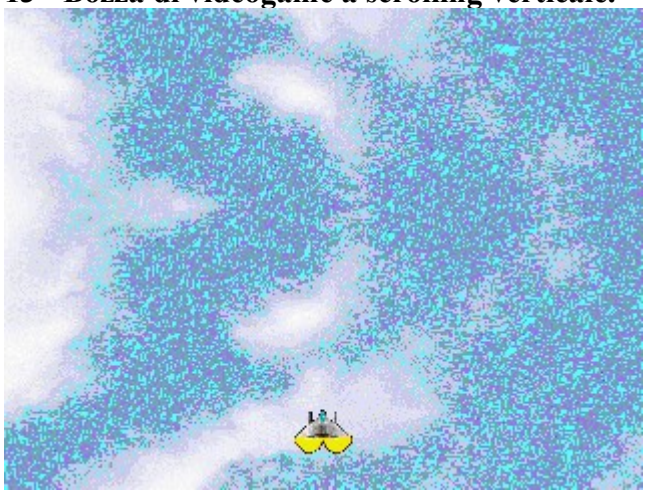
```
#include "allegro.h"

// inizializzo la variabile del buffer
BITMAP *buf, *sprite, *background;

// inizializzo la palette
PALETTE colori;

/* dichiaro variabili di posizione x e y valori di incremento valorex e valorey,
variabile dell'array oltre che la grandezza dell'array, cioè il numero di sprite
da visualizzare(qui sono indicati 5 ma potrebbero arrivare fino a 7 in questo esempio)
per ultima start che è la variabile di posizione iniziale dei mitra.
Le variabili dell'array contengono un dicitura tra parentesi quadre in cui è dichiarato
il valore massimo di variabili di quel tipo creabili.
*/

int x[100],y[100],valorex[100],valorey[100],array, numero=5, start=10;

// creo la procedura per il doppio buffering
void doppiobuffering()
{

// sincronizzo l'aggiornamento dello schermo
vsync();

// disegno il contenuto del buffer sullo schermo
blit(buf, screen, 0, 0, 0, 0, 640, 480);

// ripulisco il contenuto del buffer dopo averlo visualizzato
clear(buf);
}

// creo la procedura per visualizzare un pixel su schermo
void mitra() {

/* viene creato un array con il ciclo for. Per ogni ciclo sono create nuove
variabili x e y. Quindi nuovi sprite visualizzati nel buffer
*/
for (array=0; array<=numero; array++) {

// VISUALIZZA LO SPRITE SUL BUFFER PER OGNI CICLO DI ARRAY
draw_sprite(buf, sprite, x[array], y[array]);
}
}
// creo la procedura che contiene la routine del movimento
void movimento() {

// Creo un array di variabili relative al movimento
for (array=0; array<=numero; array++) {

// spostamento sull'asse x
x[array]=x[array]+1*valorex[array];

// spostamento sull'asse y
y[array]=y[array]+1*valorey[array];
```

```
// incontro con il bordo schermo degli sprite
if (x[array]==576) valorex[array]=-1;
if (x[array]==0) valorex[array]=+1;
if (y[array]==438) valorey[array]=-1;
if (y[array]==0) valorey[array]=+1;
}
}

// creo la procedura principale
int main()
{

// inizializzo la libreria allegro.h
allegro_init();

// installo la tastiera
install_keyboard();

// setto la profondità di colore della palette
set_color_depth(32);

// assegno una variabile alla palette
set_palette(colori);

// determino la risoluzione dello schermo
set_gfx_mode(GFX_GDI, 640, 480, 0, 0);

/* CARICA IL FILE MITRA.BMP DOVE IL VIOLETTO IN MODALITA' TRUE COLOR E
TRASPARENTE (ROSSO=255, VERDE=0, BLU=255) MENTRE PER LA MODALITA' A 256
COLORI
IL TRASPARENTE E' IL NERO (R=0, V=0, B=0) */
sprite = load_bitmap("mitra.bmp",colori);

// CARICA IL FILE BACKGROUND.BMP //
background = load_bitmap("background.bmp",colori);

// assegno la dimensione del buffer
buf = create_bitmap(640, 480);

// pulisco il buffer appena creato
clear(buf);

// creo un nuovo array per le variabili di partenza
for (array=0; array<=numero; array++) {

// posizione di partenza
start=start+60;

// variabili di partenza array
x[array]=start; y[array]=start;
valorex[array]=1; valorey[array]=1;
}

// inizio del ciclo
while (!key[KEY_ESC]) {
```

```
// VISUALIZZA L'IMMAGINE DI BACKGROUND SUL BUFFER //
blit(background, buf, 0, 0, 0, 0, 640, 480);

// carico le procedure
mitra();
movimento();
doppiobuffering();
}

// distruggo la varibile buffer
destroy_bitmap(buf);

}

END_OF_MAIN ();
```

Analizziamo le parti salienti del programma ARRAY.C:
Nella programmazione di un video game molto spesso è necessario visualizzare un oggetto identico molte volte contemporaneamente su schermo. La tecnica più semplice e più breve per raggiungere tale obbiettivo è quella di creare un 'array'. L'array permette la costruzione di una quantità virtualmente infinita di variabili partendo da un'unica entità che possono essere utilizzate nelle situazioni più diverse. Prestate un'attenzione particolare a questo listato ed ai suoi commenti, che vi torneranno sicuramente utile nei vostri video giochi.

13 - Bozza di videogame a scrolling verticale.



```
/* LISTATO C/C++ DI NAVETTA.C
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
HOME: WWW.BERTINETTOBARTOLOMEODAVIDE.IT
E-MAIL: contatto@bertinettobartolomeodavide.it
*/

#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA
#include "allegro.h"
#include "graph.h"
```

```
DATAFILE *dat;
BITMAP *imm;
BITMAP *buf;
int x,y;
int navcontdx, navcontsx, navferco, navfermdx, navfermdx1, navfermsx, navfermsx1;
int las, lasbx, lasby, lasb, lasc, lascx, lascy, las2, lasd, lasdx, lasdy, las3;
int sf, sff, sfw, sfh, sfx, sf1, sf2;
int mis, misbx, misby, misb, misc, miscx, miscy, mis2, misd, misdx, misdy, mis3, miss1, misca, miscb,
miscc;

BITMAP *immagine, *buffer, *buffer2; /* DICHIARAZIONI VARIABILI DEI BITMAP */
PALLETE colori; /* DICHIARAZIONE VARIABILE DELLA PALETTE */
int yc; /* DICHIARAZIONE VARIABILE Y DI SPOSTAMENTO SCROLLING */
int pattern; /* DICHIARAZIONE DI VARIABILE DEL PATTERN DA RIPETERE */

void vis()
{
vsync();
blit (buf, screen,0,0,0,0,640,480);
clear(buf);
}

void navfermadx()
{
navcontdx=navcontdx+1;
navfermdx1=navfermdx1+1;
if (navcontdx>=9) navcontdx=9;
if (navfermdx1<=9) draw_sprite(buf, dat[navcontdx].dat, x, y);


if (navfermdx1>9) {
navfermdx=navfermdx+1;
if (navfermdx>=30) navfermdx=21;
draw_sprite(buf, dat[navfermdx].dat, x, y);
}


}

void navfermasx()
{
navcontsx=navcontsx+1;
navfermsx1=navfermsx1+1;
if (navcontsx>=9) navcontsx=9;
if (navfermsx1<=9) draw_sprite_h_flip(buf, dat[navcontsx].dat, x, y);


if (navfermsx1>9) {
navfermsx=navfermsx+1;
if (navfermsx>=30) navfermsx=21;
draw_sprite_h_flip (buf, dat[navfermsx].dat, x, y);
}

}
```

```
/* FUOCO GIOCATORE NUMERO 1 */

void laser()
{
if (key[KEY_SPACE]) {
las=las+1;
las2=las2+1;
las3=las3+1;
if (las==1) lasb=1;
if (las2>=30) lasc=1;
if (las3>=60) lasd=1;
}

/* FUOCO NUMERO 1 */
if (lasb==1) {
lasby=lasby-4;

draw_sprite (buf, dat[NAVLASER].dat,(lasbx+14),lasby);
draw_sprite (buf, dat[NAVLASER].dat,(lasbx+42),lasby);

if (lasby<=-15) {
lasb=0;
lasby=y; lasbx=x; las=0; las2=29;
}
}
if (lasb==0) {
lasby=y; lasbx=x;
}

/* FUOCO NUMERO 2 */

if (lasc==1) {
lascy=lascy-4;

draw_sprite (buf, dat[NAVLASER].dat,(lascx+14),lascy);
draw_sprite (buf, dat[NAVLASER].dat,(lascx+42),lascy);

if (lascy<=-15) {
lasc=0; las2=0; las3=59;
lascy=y; lascx=x;
}
}
if (lasc==0) {
lascy=y; lascx=x;
}

/* FUOCO NUMERO 3 */

if (lasd==1) {
lasdy=lasdy-4;

draw_sprite (buf, dat[NAVLASER].dat,(lasdx+14),lasdy);
draw_sprite (buf, dat[NAVLASER].dat,(lasdx+42),lasdy);

if (lasdy<=-15) {
```

```
lasd=0; las3=0; las=0;
lasdy=y; lasdx=x;
}
}
if (lasd==0) {
lasdy=y; lasdx=x;
}

}

/* MISSILE SUPER FUOCO GIOCATORE 1 */

void missilea()
{

if (key[KEY_SPACE]) {
mis=mis+1;
mis2=mis2+1;
mis3=mis3+1;
if (mis==1) misb=1;
if (mis2>=30) misc=1;
if (mis3>=60) misd=1;
}

/* MISSILE NUMERO 1 */
if (misb==1) {
misby=misby-6;
misca=misca+1;
draw_sprite (buf, dat[misca].dat,(misbx-60),misby);
draw_sprite (buf, dat[misca].dat,(misbx+60),misby);
if (misca>=50) misca=41;
if (misby<=-15) {
misb=0;
misby=y; misbx=x; mis=0; mis2=29;
}
}
if (misb==0) {
misby=y; misbx=x;
}

/* MISSILE NUMERO 2 */

if (misc==1) {
miscy=miscy-6;
miscb=miscb+1;
draw_sprite (buf, dat[miscb].dat,(miscx-60),miscy);
draw_sprite (buf, dat[miscb].dat,(miscx+60),miscy);
if (miscb>=50) miscb=41;
if (miscy<=-15) {
misc=0; mis2=0; mis3=59;
miscy=y; miscx=x;
}
}
if (misc==0) {
miscy=y; miscx=x;
}
```

```
/* MISSILE NUMERO 3 */

if (misd==1) {
misdy=misdy-6;
miscc=miscc+1;
draw_sprite (buf, dat[miscc].dat,(misdx-60),misdy);
draw_sprite (buf, dat[miscc].dat,(misdx+60),misdy);
if (miscc>=50) miscc=41;
if (misdy<=-15) {
misd=0; mis3=0; mis=0;
misdy=y; misdx=x;
}
}
if (misd==0) {
misdy=y; misdx=x;
}

}
missile()
{
if (miss1==1) missilea();
else {
if ((mis>=1) || (mis2>=30) || (mis3>=60)) missilea();

}
}


/* ARMA SPECIALE SUPERFUOCO */

void superfuoco()
{
if (key[KEY_ENTER]) {
sf1=1;
}
/* COMPARE */
if (sf1==1) {
sff=sff+1; sfw=sfw+1; sfh=sfh+1; sfx=sfx+1;
stretch_sprite(buf, dat[sff].dat, (x+sfw), y, sfw, sfh);
stretch_sprite(buf, dat[sff].dat, (x-sfw), y, sfw, sfh);
if (sfx==50) sfx=49;
if (sfw==60) sfw=59;
if (sfh==45) sfh=44;
if (sff==41) sff=31;
if ((sfx>=49) && (sfw>=59) && (sfh>=44)) {
miss1=1;
if (key[KEY_ENTER]) sf2=1;
}
}
/* SCOMPARE */
if (sf2==1) {
sff=sff+1; sfw=sfw-1; sfh=sfh-1; sfx=sfx-1; sf1=0;
stretch_sprite(buf, dat[sff].dat, (x+sfw), y, sfw, sfh);
stretch_sprite(buf, dat[sff].dat, (x-sfw), y, sfw, sfh);
if (sfx<=0) sfx=0;
```

```
if (sfw<=0) sfw=0;
if (sfh<=0) sfh=0;
if (sff==41) sff=31;
if ((sfx<=0) && (sfw<=0) && (sfh<=0)) sf2=0; miss1=0;
}

}

/* GIOCATORE NUMERO 1 */

void navetta()

{
navferco=navferco+1;
navfermdx=navfermdx+1;

/* SPOSTAMENTO IN DIAGONALE */

if ((key[KEY_UP]) && (key[KEY_LEFT])) {
y=y-2; x=x-2;
if (navcontdx>0) {
navcontdx=navcontdx-1; navfermsx1=0; navfermdx1=0;
draw_sprite(buf, dat[navcontdx].dat, x, y);
}
else {
if ((navcontdx==0) && (navcontsx<0)) {
navcontdx=0; navcontsx=0;
draw_sprite(buf, dat[10].dat, x, y);
}
else {
navfermasx();
}
}
}

else {
/* SPOSTAMENTO IN DIAGONALE */

if ((key[KEY_UP]) && (key[KEY_RIGHT])) {
y=y-2; x=x+2;
if (navcontsx>0) {
navcontsx=navcontsx-1; navfermsx1=0; navfermdx1=0;
draw_sprite_h_flip(buf, dat[navcontsx].dat, x, y);
}
else {
if ((navcontsx==0) && (navcontdx<0)) {
navcontdx=0; navcontsx=0;
draw_sprite(buf, dat[10].dat, x, y);

}
else {
navfermadx();
}
}
}
else {
```

```
/* SPOSTAMENTO IN DIAGONALE */

if ((key[KEY_DOWN]) && (key[KEY_LEFT])) {
y=y+2; x=x-2;
if (navcontdx>0) {
navcontdx=navcontdx-1; navfermsx1=0; navfermdx1=0;
draw_sprite(buf, dat[navcontdx].dat, x, y);
}
else {
if ((navcontdx==0) && (navcontsx<0)) {
navcontdx=0; navcontsx=0;
draw_sprite(buf, dat[10].dat, x, y);

}
else {
navfermasx();
}
}
}
else {
/* SPOSTAMENTO IN DIAGONALE */

if ((key[KEY_DOWN]) && (key[KEY_RIGHT])) {
y=y+2; x=x+2;
if (navcontsx>0) {
navcontsx=navcontsx-1; navfermsx1=0; navfermdx1=0;
draw_sprite_h_flip(buf, dat[navcontsx].dat, x, y);
}
else {
if ((navcontsx==0) && (navcontdx<0)) {
navcontdx=0; navcontsx=0;
draw_sprite(buf, dat[10].dat, x, y);

}
else {
navfermadx();
}
}
}
else {
/* SPOSTAMENTO VERSO IL BASSO */

if (key[KEY_DOWN]) {
y=y+2;
if (navcontdx>0) {
navcontdx=navcontdx-1; navfermsx1=0; navfermdx1=0;
draw_sprite(buf, dat[navcontdx].dat, x, y);
}
else {
if ((navcontdx==0) && (navcontsx<0)) {
navcontdx=0; navcontsx=0;
if (navferco>=19) navferco=10;
draw_sprite(buf, dat[navferco].dat, x, y);

}
else {
```

```
if (navcontsx>0) {
navcontsx=navcontsx-1; navfermsx1=0; navfermdx1=0;
draw_sprite_h_flip(buf, dat[navcontsx].dat, x, y);
}
else {
if ((navcontsx==0) && (navcontdx<0)) {
navcontdx=0; navcontsx=0;
if (navferco>=19) navferco=10;
draw_sprite(buf, dat[navferco].dat, x, y);

}

else {
if (navferco>=19) navferco=10;
draw_sprite(buf, dat[navferco].dat, x, y);
if (navferco>=19) navferco=10;
}
}
}
}
}
else {
/* SPOSTAMENTO VERSO SINISTRA */

if (key[KEY_LEFT]) {
x=x-2;
if (navcontdx>0) {
navcontdx=navcontdx-1; navfermsx1=0; navfermdx1=0;
draw_sprite(buf, dat[navcontdx].dat, x, y);
}
else {
if ((navcontdx==0) && (navcontsx<0)) {
navcontdx=0; navcontsx=0; navferco=10;
draw_sprite(buf, dat[10].dat, x, y);

}
else {
navfermasx();

}
}
}

else {
/* SPOSTAMENTO VERSO DESTRA */

if (key[KEY_RIGHT]) {
x=x+2;
if (navcontsx>0) {
navcontsx=navcontsx-1; navfermsx1=0; navfermdx1=0;
draw_sprite_h_flip(buf, dat[navcontsx].dat, x, y);
}
else {
if ((navcontsx==0) && (navcontdx<0)) {
navcontdx=0; navcontsx=0; navferco=10;
draw_sprite(buf, dat[10].dat, x, y);
```

```
}
else {
navfermadx();
}
}
}
else {
/* SPOSTAMNETO VERSO L'ALTO */
if (key[KEY_UP]) {
y=y-2;
if (navcontdx>0) {
navcontdx=navcontdx-1; navfermsx1=0; navfermdx1=0;
draw_sprite(buf, dat[navcontdx].dat, x, y);
}
else {
if ((navcontdx==0) && (navcontsx<0)) {
navcontdx=0; navcontsx=0;
if (navferco>=19) navferco=10;
draw_sprite(buf, dat[navferco].dat, x, y);

}
else {
if (navcontsx>0) {
navcontsx=navcontsx-1; navfermsx1=0; navfermdx1=0;
draw_sprite_h_flip(buf, dat[navcontsx].dat, x, y);
}
else {
if ((navcontsx==0) && (navcontdx<0)) {
navcontdx=0; navcontsx=0;
if (navferco>=19) navferco=10;
draw_sprite(buf, dat[navferco].dat, x, y);

}

else {
if (navferco>=19) navferco=10;
draw_sprite(buf, dat[navferco].dat, x, y);

}
}
}
}
}

else {
/* SE NON VIENE PREMUTO NESSUN TASTO */
if (navcontdx>0) {
navcontdx=navcontdx-1;
draw_sprite(buf, dat[navcontdx].dat, x, y);
}
else {
if ((navcontdx==0) && (navcontsx<=0)) {
navcontsx=0; navcontdx=0; navfermsx1=0;
if (navferco>=19) navferco=10;
draw_sprite(buf, dat[navferco].dat, x, y);
```

```
}
}

if (navcontsx>0) {
navcontsx=navcontsx-1;
draw_sprite_h_flip(buf, dat[navcontsx].dat, x, y);
}
else {
if ((navcontsx==0) && (navcontdx<=0)) {
navcontdx=0; navcontsx=0;
if (navferco>=19) navferco=10;
draw_sprite(buf, dat[navferco].dat, x, y);

}
}



}

}
}
}
}
}
}
}
}

void blocco()
{
if (yc>=0) {
yc=yc-1;
}
}

/* SPOSTA LO SCHERMO IN SCROLLING VERTICALE */
void scrolling()
{

/* CICLO FOR CHE FA AVANZARE LO SCHERMO SULL ASSE Y DI 1 PIXEL*/
yc=yc+1;

/* INSERISCE IL PRIMO BUFFER NEL SECONDO BUFFER CREANDO L'ILLUSIONE DELLO
SPOSTAMENTO IN Y */
blit (buffer, buf, 0, 0, 0, yc, 640, 1920);

}


int main()
{

allegro_init();
install_keyboard();
```

```
/* SETTA LA PROFONDITA' DI COLORE A 8 BIT */
set_color_depth(8);
set_gfx_mode(GFX_GDI, 640, 480, 0, 0);

dat = load_datafile("graph.dat");
set_palette(dat[PAL].dat);
buffer = create_bitmap(640, 1920);
buf = create_bitmap(640,480);
clear(buffer);
clear(buf);

/* CICLO FOR CHE RIPETE IL PATTERN PER 3 VOLTE A PASSI DI 480 PIXEL */
for (pattern=0;pattern<=1920;pattern=pattern+960) {

/* INSERISCE L'IMMAGINE NEL PRIMO BUFFER */
blit(dat[zcielo].dat,buffer,0,0,0,pattern,640,960);
}

yc=-1440;
navcontsx=0;
navcontdx=0;
x=290; y=400;
sff=31;
navferco=9;
navfermdx=20;
navfermsx=20;
misca=41;
miscb=41;
miscc=41;


while (!key[KEY_ESC]) {

blocco();
scrolling();
navetta();

laser();
superfuoco();
missile();
vis();
}

destroy_bitmap(imm);
destroy_bitmap(immagine);

readkey();

return 0;
}
END_OF_MAIN();
```

Questo è solo un piccolo esempio di quello che si può realizzare con le tecniche di programmazione spiegate in questo libro. Naturalmente le potenzialità sono enormi e dipende solo dalla genialità e dalla

fantasia che possedete.

14 - Un altro classico della storia dei videogiochi: INVADER.


```
// VIDEOGIOCO INVADER //
// SVILUPPATO DA BARTOLOMEO DAVIDE BERTINETTO //
// NEL MESE DI LUGLIO 2002 //
// QUESTO VIDEOGIOCO E' IL PRIMO CHE IO ABBIA CREATO //
// E' MOLTO SEMPLICE E PROPRIO PER QUESTO E' NATO COME ESEMPIO //
// PER CHIUNQUE VOGLIA IMPARARE A CREARE UN VIDEOGIOCO USANDO IL C/C++ //
// www.bertinettobartolomeodavide.it

#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA
#include "allegro.h"

BITMAP *sprite, *buf, *fuoco, *nemico[25], *fnemico, *esplosione, *logo;
PALLETE co, image;
int x, y; // coordinate giocatore //
int fx, fy, fok; // coordinate e variabili fuoco giocatore //
int ex, ey, ritespl, esplode; // ex e ey=pos. esplosione player, ritespl=tempo di esplosione, explode=start per l'esplosione //
int nx[25], ny[25], na[25], cattivovince, ritardogameover, vel; // coordinate e variabili nemico in array fino a 25, incremento velocita' //
int fnr[25], fn[25], fnx[25], fny[25], fnstop; // fnr[a]= numero casuale, fn[a]= permette lo sparo del proiettile, fnstop [a]= fa si che non vengano sparati altri proiettili, fny[a] e fny[a]= posizione proiettile //
int espx[25], espy[25]; // coordinate in array per esplosione //
int a, b, c, e[25], f[25], g[25]; // variabili di array e posizione di partenza nemici //
int d=25, contapunti, meno, vite=3, ini; // numero di nemici - dimensione max di array - variabile conta punti e contatore di esaurimento nemici, vite giocatore settato a 3, variabile di inizio //
char score[80]; // crea la dimensione max in cifre per il punteggio //
int ritardo; // variabile temporizzatrice //
SAMPLE *suono, *gameov, *vitameno; // variabili per il sonoro //


// aggiornamento dello schermo in doppio buffering //
void vai()
{
set_palette(co); // setta la palette degli sprites //
vsync();
```

```
blit(buf, screen, 0, 0, 0, 0, 320, 200);
clear(buf);
}

void inizio() {
if (ini==0) { // variabile di inizio o di gioco: 0=inizio - 1=inizio //
textout(buf, font, "PREMI ENTER PER INIZIARE", 65, 95, 215);
if (key[KEY_ENTER]) {
vite=3;
ini=1;
meno=0;
}
}
}
void partenzadazero() {
// valori delle variabili di partenza //
if (ini==1) { // variabile di inizio o di gioco: 0=inizio - 1=gioco //
if (meno==0) { // vengono assegnati i seguanti valori solo se meno e uguale a zero: cioe' all'inizio e
quando vengono distrutti tutti i nemici, cosi si ricomincia a giocare //
meno=16;
d=15;
fx=x;
fy=y;
x=150;
y=180;
fok=0;
a=0, b=0, c=20;

for (a=0; a<=d; a++) { // ciclo che crea un array di nuove variabili per la posizione di partenza di ogni
nemico //
if (a>1) {
b=b+40;
}
if (b>=200) {
b=0;
c=c+40;
}

na[a]=1;
ny[a]=c;
nx[a]=b;
e[a]=0; f[a]=0; g[a]=0;
espx[a]=0; espy[a]=0;
vel=1;
}
}
}
}

// tasti di spostamento del giocatore sinistra destra e visualizzazione giocatore //
void tasti()
{
if (ini==1) { // variabile di inizio o di gioco: 0=inizio - 1=gioco //
draw_sprite(buf, sprite, x, y); // disegna il giocatore //
if (key[KEY_LEFT]) x=x-2; if (x<=0) x=0; // sposta a sx //
if (key[KEY_RIGHT]) x=x+2; if (x>=300) x=300; // sposta a dx //
```

```
}
}

// Fuoco giocatore //
void fire()
{
if (ini==1) { // variabile di inizio o di gioco: 0=inizio - 1=gioco //
if (fok==0) { // se fok e' uguale a 0 allora permette di premere il tasto di fuoco //
if (key[KEY_SPACE]) { // se viene premuto lo spazio //
fok=1; // fok comincia a incrementate di 1 //
play_sample(suono, 255,128,1000, FALSE); // emette un suono //
}
}
if (fok==1) { // se fok e' uguale a 1 //
fy=fy-3; // fy decrementa di 3 facendo avanzare il proiettile //
draw_sprite(buf, fuoco, 7+fx, fy); // disegna il proiettile, il +7 serve per centrare il proiettile sul
giocatore //
if (fy==0) { // se fx(il proiettile) va a 0(a bordo schermo in alto) //
fok=0; // allora fok ritorna a 0 //
fy=y; fx=x; fok=0; } } // tutte le variabili di posizione diventano uguali a quelle del giocatore //
if (fok==0) { // se fok e' uguale a zero //
fx=x; fy=y; // le variabili sono uguali a quelle del giocatore //
}
}
}

// visualizzazione e movimento nemici //
void cattivo()
{
if (ini==1) { // variabile di inizio o di gioco: 0=inizio - 1=gioco //
for (a=0; a<=d; a++) { // ciclo che crea un array di nuove variabili //
if (e[a]==0) { // se e[a] e' diverso da 0 significa che il nemico e' stato colpito e quindi non viene
visualizzato //

draw_sprite(buf, nemico[a], nx[a], ny[a]); // disegna il nemico //
if (na[a]==1) { // se na[a] e' ugale a 1 //
nx[a]=nx[a]+vel; // nx[a] incrementa di 1 e il nemico avanza a dx //
if (nx[a]>=300) { // se nx supera 300 ny[a] incrementa di 25 e il nemico scende di una riga.
ny[a]=ny[a]+20; na[a]=2; } // na[a] diventa uguale a 2 //
}
if (na[a]==2) { // se na[a] e' uguale a 2 //
nx[a]=nx[a]-vel; // allora nx[a] decrementa di 1 e il nemico avanza verso sinistra //
if (nx[a]<=0) { // se nx[a] e minore o uguale a 0 //
ny[a]=ny[a]+20; na[a]=1; } // allora ny[a] incrementa di 25 e quindi il nemico scende di una riga. Così
na[a] diventa uguale a 1 //
}
if (ny[a]>=180) { // se ny[a] e' uguale maggiore o uguale a 175 //
cattivovince=1; // IL NEMICO VINCE GAME OVER PERCHE' IL CATTIVO TOCCA IL TERRENO
QUINDI CATTIVO VINCE DIVENTA UGUALE A 1 //
play_sample(gameov, 255,128,1000, FALSE); // emette un suono //
}
} } } }

// collisione con alieno //
void collisione()
{ // variabile di inizio o di gioco: 0=inizio - 1=gioco //
```

```
if (ini==1) {
for (a=0; a<=d; a++) { // ciclo che crea un array di nuove variabili //
if (e[a]==0) { // se e[a] e' uguale a 0 allora esegue //
if ((fy<=(ny[a]+14)) && (fy>=(ny[a]-14))) { // se fy(fuoco giocatore) e' minore o uguale a ny[a]
(posizione nemico) più 14(altezza nemico) e fy e' maggiore o uguale a ny[a] meno 14 allora //
if ((fx>=(nx[a]-9)) && (fx<=(nx[a]+10))) { // se fx(fuoco giocatore) e' maggiore o uguale a nx[a]
(posizione nemico] meno 9(larghezza nemico) e fx e' minore o uguale a nx[a] più 10 allora //
fok=0; e[a]=1; fx=x; fy=y; // variabili fuoco giocatore a zero ed e[a] diventa uguale a 1 così non ci sarà
un'altra collisione con il nemico già esploso //
contapunti++; // incrementa il punteggio di 1 //
meno--; // conta quanti nemici vengono distrutti //
play_sample(vitameno, 255,128,1000, FALSE); // emette un suono //
}
}
} } } }

// esplosione alieno //
void explode()
{
if (ini==1) { // variabile di inizio o di gioco: 0=inizio - 1=gioco //
for (a=0; a<=d; a++) { // ciclo array esplosione //
if (e[a]==1) { // se e[a] e' a 1 allora significa che il nemico e stato colpito e deve esplodere //
if (f[a]==0) { // se f[a] e' uguale a 0 vuol dire che che il nemico non e' ancora esploso //
espx[a]=nx[a]; espy[a]=ny[a]; // l'esplosione avviene nella stessa posizione dell'alieno colpito cioè espx[a]
(posizione esplosione) e' uguale a nx[a](posizione alieno) e espy[a](posizione esplosione) e' uguale a
ny[a](posizione alieno) //
f[a]=1; // se f[a] e' uguale a 1 l'esplosione di questo alieno non avverrà più //
nx[a]=0; ny[a]=0; // mette a zero la posizione dell' alieno esploso //
}
if (f[a]==1) { // se f[a] e' uguale a 1 parte la sequenza dell' esplosione //
g[a]=g[a]+1; // contatore di ritardo per il tempo di permanenza dell' esplosione //
draw_sprite(buf, esplosione, espx[a], espy[a]); // visualizza l'esplosione //
if (g[a]>=50) { // se g[a] e' maggiore o uguale a 50 significa che e' stato superato il tempo limite di
visualizzazione dell' esplosione //
f[a]=2; e[a]=2; // f[a] e e[a] sono impostati a 2 cosi non verrà colpito un alieno già colpito e non esploderà
un alieno già esploso //
}
}
} }
} }

// indicatore punteggio //
void puntivite() {
if (ini==1) { // variabile di inizio o di gioco: 0=inizio - 1=gioco //
sprintf(score, "PUNTI: %ld VITE: %ld", (long)contapunti, (long)vite); // da il valore a score e a vite da
visualizzare //

textout(buf, font, score, 1, 1, 215); // inserisce nel doppio buffer dello schermo il valore di score e vite che
verrà poi visualizzato //
}
}

// routine di incremento velocità del nemico quando diminuisce il numero //
void incrementovelocitanemico() {

if (ini==1) {
```

```
if ((meno<=8) && (meno>=4)) { // se ci sono tra 4 e 8 nemici allora //
vel=2; // la velocità del nemico aumenta di 1 //
}
if ((meno<=3) && (meno>=2)) { // se ci sono tra 3 e 2 nemici allora //
vel=3; // la velocita' del nemico aumenta di 2 //
}
if (meno==1) { // se ce un solo nemico allora //
vel=4; // la velocita' del nemico aumenta di 3 //
}
}
}

// game over quando il cattivo vince o il player viene colpito //
void gameover() {
if (ini==1) { // variabile di inizio o di gioco: 0=inizio - 1=gioco //
if (cattivovince==1) { // se il cattivo tocca terra cattivi vince diventa = a 1 allora //
textout (buf, font, "GAME OVER", 120, 95, 215); // VISUALLIZZA GAME OVER //
ritardogameover++; // INCREMENTA DI 1 IL CONTATORE PER LA VISUALIZZAZIONE DI
GAME OVER //
}
if (ritardogameover>=100) { // DETERMINA IL TEMPO DI VISUALIZZAZIONE DELLA SCRITT
GAME OVER SE E' UGUALE A 100 ALLORA //
cattivovince=0; // CATTIVO VINCE RITORNA AL VALORE DI PARTENZA //
meno=0; // MENO RITORNA AL VALORE DI PARTENZA E FA RICOMINCIARE IL GIOCO DA
CAPO //
vite=0; // LE VITE VENGONO PORTATE A ZERO //
contapunti=0; // I PUNTI VENGONO AZZERA IL PUNTEGGIO //
ritardogameover=0; // AZZERA IL CONTATORE DI PERMANENZA DELLA SCRITTA //
ini=0; // azzera ini per ricominciare da capo //
esplode=0; // azzera lo start dell'esplosione per eliminare lo scoppio //
ritespl=0; // azzera il contatore di permanenza dell'esplosione //
}
}
if (esplode==1) { // se e' uguale a 1 parte la routine dell'esplosione del player //
draw_sprite(buf, esplosione, ex, ey); // disegna l'esplosione //
ritespl++; // contatore di permanenza dell'esplosione //
if (ritespl>=100) { // il contatore e' uguale o maggiore di 100 allora //
esplode=0; // lo start dell esplosione ritorna a zero //
x=150; y=180; // le coordinate del player ritornano centrali //
ritespl=0; // azzera il contatore dell'esplosione //
}
}
}

// routine di sparo del nemico //
void sparonemico() {
if (ini==1) {
for (a=0; a<=d; a++) { // se e' a 1 esegue la routine di sparo nemico //
if (e[a]==0) { // se e' a 0 significa che il nemico non e' stato colpito //
if (fnstop==0) { // Controlla che i nemici non sparino tutti contemporaneamente //
if ((fnr[a]>=20000) && (fnr[a]<=21000)) { // Fa si che avvenga lo sparo solo se fnr e' tra 20000 e
21000 //
play_sample(suono, 255,128,1000, FALSE); // emette un suono //
fn[a]=1; fnx[a]=nx[a]; fny[a]=ny[a]; fnstop=1; // Setta tutte le variabili per la partenza del proiettile //
}
fnr[a]=rand(); // fnr assume un numero casuale //
```

```
}
}
if (fn[a]==1) { // se fn[a] e' uguale a 1 il proiettile parte //
draw_sprite(buf, fnemico, fnx[a]+10, fny[a]+7); // disegna il proiettile centrato sul nemico //
fny[a]++; // incrementa fny di 1 facendo scorrere il proiettile verso il basso //
if (fny[a]>=200) { // se fny(il proiettile) e' uguale o maggiore di 200 allora //
fny[a]=ny[a]; // fny[a] riprende il valore di ny[a] //
fnx[a]=nx[a]; // fnx[a] riprende il valore di nx[a] //
fnr[a]=0; // il numero casuale viene riportato a 0 //
fn[a]=0; // fn[a] viene azzerato cosi che il proiettile non si muova e non venga disegnato //
fnstop=0; // fnstop ritorna a 0 cosiche' possano partire altri proiettili //
}
}
}
}
}

// collisione del proiettile nemico con il player //
void collisioneconproiettilenemico() {
if (ini==1) { // se e' uguale a 1 allora esegue la routine //
for (a=0; a<=d; a++) { // crea l'array //
if (fn[a]==1) { // se e' uguale a 1 significa che il proiettile e' stato sparato //
if ((fny[a]>=y-10) && (fny[a]<=y)) { // punto di collisione con il player y //
if ((fnx[a]>=x-12) && (fnx[a]<=x+8)) { // punto di collisione con il player x //
fnx[a]=nx[a]; fny[a]=ny[a]; // se il player viene colpito il proiettile viene trasferito nella posizione di
partenza //
vite--; // decremento di una vita //
esplode=1; // se e uguale a 1 fa partire l'esplosione del player //
ex=x; ey=y; // le variabili dell'esplosione diventano uguali a quelle del player //
x=-1000; y=-1000; // il player scompare fuori dallo schermo //
play_sample(vitameno, 255,128,1000, FALSE); // emette un suono //
if (vite==0) { // se le vite sono uguali a 0 allora //
cattivovince=1; // parte la routine di game over //
play_sample(gameov, 255,128,1000, FALSE); // emette un suono //
}
}
}
}
}
}
}

// inizializzazione della libreria allegro //
int main()
{
allegro_init();
install_keyboard();

sprite = load_tga("player.tga",co); // carica il giocatore //
fuoco = load_tga("fuoco.tga",co); // carica il fuoco //
fnemico = load_tga("fuoco.tga",co); // carica il fuoco nemico //

for (a=0; a<=d; a++) { // ciclo che crea un array di nuove variabili //
nemico[a] = load_tga("nemico1.tga",co); // carica l'immagine dei nemici //
}
```

```
esplosione = load_tga("esplosione.tga",co); // carica l'immagine dell'esplosione //
buf = create_bitmap(320, 200); // crea un una finestra di buffer 320 x 200 pixel //
clear(buf); // pulisce il buffer dello schermo //

set_gfx_mode(GFX_SAFE, 320, 200, 0, 0); // inizializza il formato grafico //
install_sound(DIGI_GDI, MIDI_AUTODETECT, 0); // inizializza la scheda audio //

suono = load_sample("fuoco.wav"); // carica il file wav //
gameov = load_sample("gameover.wav"); // carica il file wav //
vitameno = load_sample("vitameno.wav"); // carica il file wav //

set_volume(255,255); // setta i parametri di volume //

// logo di presentazione //
logo = load_tga("davide1.tga",image); // carica l'immagine logo //
set_palette(image); // setta la palette della immagine della presentazione //
blit (logo, screen, 0, 0, 70, 25, 320,200); // visualizza l'immagine //
textout(screen, font, "INVADER 2002", 0, 0, 255); // TITOLO DEL VIDEOGIOCO //
textout(screen, font, "WWW.BERTINETTOBARTOLOMEODAVIDE.IT", 20, 10, 255);
textout(screen, font, "SVILUPPATO DA", 100, 175, 255);
textout(screen, font, "BARTOLOMEO DAVIDE BERTINETTO", 40, 185, 255);

for (ritardo=0; ritardo<=250; ritardo++) { // tempo di visualizzazione della presentazione //
vsync(); // sincronizzazione //
}

destroy_bitmap(logo); // distrugge l'immagine //

// ciclo while che finisce con la pressione di ESC //
// all' interno di questo cicle sono racchiuse tutti gli oggetti costruiti in precedenza //
while (!key[KEY_ESC]) {
partenzadazero();
vai();
tasti();
fire();
cattivo();
collisione();
explode();
puntivite();
gameover();
inizio();
incrementovelocitanemico();
sparonemico();
collisioneconproiettilenemico();
}

// vengono distrutti tutti i bitmap caricati prima della chiusura del programma //
destroy_bitmap(sprite);
destroy_bitmap(fuoco);
destroy_bitmap(fnemico);
for (a=0; a<=d; a++) { //ciclo che crea un array di nuove variabili //
destroy_bitmap(nemico[a]);
}
destroy_bitmap(buf);
return 0;
}
```

```
END_OF_MAIN ();
```

Analizziamo le parti salienti del programma INVADER.C:
Sicuramente i lettori più anziani si ricorderanno di questo coin-op nelle sale giochi da bar. Forse a queste persone farà piacere scrivere il codice sopra esposto per ricreare con le proprie capacità il loro videogioco preferito nella loro infanzia.

15 - Il primo grande classico della storia dei videogiochi: PONG 2002.


```
// VIDEOGIOCO PONG2002 //
// SVILUPPATO DA BARTOLOMEO DAVIDE BERTINETTO //
// NEL MESE DI AGOSTO 2002 //
// QUESTO VIDEOGIOCO E' IL SECONDO CHE IO ABBIA CREATO //
// E' MOLTO SEMPLICE E PROPRIO PER QUESTO E' NATO COME ESEMPIO //
// PER CHIUNQUE VOGLIA IMPARARE A CREARE UN VIDEOGIOCO USANDO IL C/C++ //
// SITO INTERNET: WWW.BERTINETTOBARTOLOMEODAVIDE.IT//
// EMAIL: contatto@bertinettobartolomeodavide.it //

#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA

#include "allegro.h" // carica la libreria allegro //

BITMAP *buf, *logo; // variabili dei bitmap //
PALETTE image; // variabile palette dei bitmap //
int ritardo, reset, contapunti, vite=3, fine; // variabili di gioco //
int x, y; // coordinate giocatore //
float px, py, incx, incy; // variabili reali per le coordina della pallina //
int nx, ny; // coordinate nemico //
char score[80]; // variabile per visualizzare i punti //
SAMPLE *suono, *gameov, *vitameno; // variabili di suono //

// aggiornamento dello schermo in doppio buffering //

void avvio() { // routine di avvio //
if (reset==0) { // se e' uguale a 0 allora //
reset=1; // reset diventa 1 //
x=0; y=10; // punto di partenza del giocatore //
incx=2; incy=.1; // accelerazione iniziale della pallina //
```

```
px=6; py=10; // posizione iniziale della pallina //
nx=0, ny=10; // posizione iniziale del nemico //
}

}

void vai() // routine di doppio buffering //
{

vsync(); // temporizza l'aggiornamento //

blit(buf, screen, 0, 0, 0, 0, 320, 200); // finestra da visualizzare ( schermo intero ) //
clear(buf); // cancella il buffer dello schermo //
}

void giocatore() { // routine giocatore //

rectfill(buf, 0, 0+y, 5, 30+y, 255); // primitiva che disegna il giocatore //
if (key[KEY_UP]) { // premuta freccia su allora //
y=y-3; // sposta la tavoletta di tre pixel //
}
if (key[KEY_DOWN]) { // premuta freccia giu' //
y=y+3; // sposta la tavoletta di tre pixel //
}
if (y>=169) { // se y e maggiore o uguale a 169 significa che la tavoletta e contro lo schermo //
y=169; // quindi non si muove //
}
if (y<=10) { // se y e minore od uguale a 10 significa che la tavoletta e' contro lo schermo //
y=10; // quindi la tavoletta non si muove //
}
}

void avversario() { // routine dell' avversario //

rectfill(buf, 314, 0+ny, 319, 30+ny, 255); // primitiva che disegna l'avversario //
ny=py-15; // l'avversario segue sempre la pallina e la centra //
if (ny>=169) { // se ny e' maggiore od uguale a 169 significa che e contro lo schermo //
ny=169; // quindi l'avversario e fermo //
}
if (ny<=10) { // se ny e minore od uguale a 10 significa che l'avversario e contro lo schermo //
ny=10; // quindi e fermo //
}
}


void pallina() { // routine di movimento della pallina //

circlefill(buf, px, py, 3, 255); // primitiva che disegna la pallina //
px=px+incx; // avanzamento pallina sulla x con accelerazione //
py=py+incy; // avanzamento pallina sulla y con accelerazione //

if (py>=200) { // se la pallina tocca il bordo in basso dello schermo allora //
incy=incy*-1; // rimbalza a specchio e cambia direzione //
}
if (py<=10) { // se la pallina tocca il bordo dello schermo superiore allora //
incy=incy*-1; // rimbalza a specchio e cambia di direzione //
```

```
}
if (px>320) { // se px e maggiore od uguale di 320 significa che la pallina non e stata presa dalla racchetta del nemico allora //
reset=0; // il gioco ricomincia //
}
if (px<0) { // se px e minore od uguale a 0 significa che il giocatore non ha preso la pallina allora //
reset=0; // il gioco ricomincia //
vite--; // viene tolta una vita al giocatore //
play_sample(vitameno, 255,128,1000, FALSE); // emette un suono //
}
}

void collisione() { // routine di collisione della pallina //
if (px>=314) { // la pallina tocca la racchetta del nemico //
if ((py>=ny-3) && (py<=ny+33)) { // controlla che non sia fuori dai bordi laterali della racchetta ////
incx=incx*-1; // rimbalza a specchio //
play_sample(suono, 255,128,1000, FALSE); // emette un suono //

}
}

if (px<=5) { // la pallina tocca la racchetta del giocatore //
if ((py>=y-3) && (py<=y+33)) { // controlla che non sia fuori dai bordi laterali della racchetta del giocatore //
incx=incx*-1+0.35; // rimbalza a specchio e varia l'accelerazione //
if (py>y+15) { // se tocca la parte inferiore della racchetta allora //
incy=incy+0.2; // allora varia l'angolo della pallina //
}
if (py<y+15) { // se tocca la parte inferiore della racchetta allora //
incy=incy-0.2; // varia l'angolo //
}
contapunti++; // ad ogni rimbalzo incrementa di un punto //
play_sample(suono, 255,128,1000, FALSE); // emette un suono //

}
}
}

// indicatore punteggio //

void puntivite() {
sprintf(score, "PUNTI: %ld VITE: %ld", (long)contapunti, (long)vite); // da il valore a score e a vite da visualizzare //
textout(buf, font, score, 1, 1, 255); // inserisce nel doppio buffer dello schermo il valore di score e vite che verra poi visualizzato //

}

void gameover() { // routine di game over //
if (vite==0) { // se le vite sono uguali a 0 allora //
play_sample(gameov, 255,128,1000, FALSE); // emette un suono //
for (fine=0; fine<=200; fine++) { // ritarda la permanenza della scritta di game over //
vsync(); textout(screen, font, "GAME OVER", 120, 90, 255); // visualizza game over //
if (fine>=199) { // quando game over scompare allora //
vite=3; reset=0; contapunti=0; // vengono resettate tutte le variabili e la partita riparte da zero //
}
```

```
}
}
}


// inizializzazione della libreria allegro //

int main() // parte l'inizializzazione di allegro //
{


allegro_init(); // fa partire allegro //
install_keyboard(); // installa la tastiera //

buf = create_bitmap(320, 200); // crea un una finestra di buffer 320 x 200 pixel //
clear(buf); // pulisce il buffer dello schermo //

set_gfx_mode(GFX_GDI, 320, 200, 0, 0); // inizializza il formato grafico //
install_sound(DIGI_AUTODETECT, MIDI_AUTODETECT, 0); // inizializza la scheda audio //
suono = load_sample("pallina.wav"); // carica un suono //
gameov = load_sample("gameover.wav"); // carica un suono //
vitameno = load_sample("vitameno.wav"); // carica un suono //
set_volume(255,255); // imposta il volume //



// logo di presentazione //


logo = load_tga("davide1.tga",image); // carica l'immagine logo //
set_palette(image); // setta la palette della immagine della presentazione //
blit (logo, screen, 0, 0, 70, 25, 320,200); // visualizza l'immagine //
textout(screen, font, "PONG 2002", 0, 0, 255); // TITOLO DEL VIDEOGIOCO //
textout(screen, font, "WWW.BERTINETTOBARTOLOMEODAVIDE.IT", 30, 10, 255);
textout(screen, font, "SVILUPPATO DA", 100, 175, 255);
textout(screen, font, "BARTOLOMEO DAVIDE BERTINETTO", 40, 185, 255);
for (ritardo=0; ritardo<=250; ritardo++) { // tempo di visualizzazione della presentazione //
vsync(); // sincronizzazione //
}
destroy_bitmap(logo); // distrugge l'immagine //

set_palette(default_palette); // utilizza la palette del BIOS //


// ciclo while che finisce con la pressione di ESC //
// all' interno di questo cicle sono racchiuse tutti gli oggetti costruiti in precedenza //

while (!key[KEY_ESC]) { // quando viene premuto ESC il gioco finisce //

// carica le routine //

avvio();
vai();
giocatore();
avversario();
pallina();
```

```
collisione();
puntivite();
gameover();
}

// fine programma //
return 0;
}
END_OF_MAIN ();
```

Analizziamo le parti salienti del programma PONG2002.C:
Questo videogioco è un esempio su come realizzare un eseguibile partendo da uno spunto semplice come il vecchio PONG; ritengo che per iniziare si deve partire dalle basi gettate ormai più di 25 anni fa con i videogames di tipo 'HOME'.
Il listato sopra esposto è ampiamente commentato e spiegato. Provate a modificarlo come preferite, in questo modo affinerete le vostre capacità di programmazione.

16 - Ripropongo un famosissimo gioco che ha segnato la storia di un noto computer a 16 bit.

```
/* LISTATO C/C++ BEAST
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
*/

#include "allegro.h" /* CARICA LA LIBRERIA ALLEGRO.H */
#include "beast.h" /* CARICA LA LIBRERIA CHE ABBIAMO CREATO CON IL FILE *.DAT */
#include "time.h"

SAMPLE *suono;
BITMAP *buffer; // DICHIARAZIONE VARIABILE DEL BUFFER SCHERMO //
DATAFILE *dat; // DICHIARAZIONE VARIABILE DEL FILE DAT //
int beastani,beastaniaccosciata, beastfermo, destra, sinistra, beastanipugno; // VARIABILI RELATIVE ALLO SPRITE DEL PROTAGONISTA //
int temporizzatore,intervallo, tempriparte; // VARIABILE NECESSARIE ALLA SCANSIONE DEL TEMPO DIANIMAZIONE //
int tastopremuto; // CONTROLLA I TASTI CHE SONO STATI PREMUTI //
int x,y, corretto;
long frame, fps, tempo1;
```

```
char score[80];

BITMAP *immagine1, *immagine2, *immagine3, *immagine4, *immagine5; /* DICHIARAZIONE
VARIABILI DEI BITMAP */ /* DICHIARAZIONE VARIABILE IMMAGINE */
BITMAP *strato1, *strato2, *strato3, *strato4, *strato5; /* DICHIARAZIONE VARIABILI DEI
BITMAP */
PALETTE colori; /* DICHIARAZIONE VARIABILE COLORI */
int a, b, c, d, e, sposta, sposta1, sposta2, sposta3, sposta4, sposta5; /* DICHIARAZIONE VARIABILI
DEGLI INTERI */

void scorrimento()
{
if (tastopremuto==0 || tastopremuto==2) {
if (key[KEY_LEFT]) { // ATTENDE LA PRESSIONE DEL TASTO CURSORE //
sposta1=sposta1+5; sposta2=sposta2+4; sposta3=sposta3+3; sposta4=sposta4+2; sposta5=sposta5+1; //
SCROLLING DEI LIVELLI DI PARALLELASSE //
} }
if (tastopremuto==0 || tastopremuto==1) {
if (key[KEY_RIGHT]) { // ATTENDE LA PRESSIONE DEL TASTO CURSORE //
sposta1=sposta1-5; sposta2=sposta2-4; sposta3=sposta3-3; sposta4=sposta4-2; sposta5=sposta5-1; //
SCROLLING DEI LIVELLI DI PARALLELASSE //
} }

// BLOCCA LO SPOSTAMENTO DELLO SCHERMO //
if (sposta5>=0) {
sposta1=sposta1-5; sposta2=sposta2-4; sposta3=sposta3-3; sposta4=sposta4-2; sposta5=sposta5-1;
}
if (sposta5<=-320) {
sposta1=sposta1+5; sposta2=sposta2+4; sposta3=sposta3+3; sposta4=sposta4+2; sposta5=sposta5+1;
}
/* VISUALIZZA L'IMMAGINE SUL DOPPIO BUFFERING */
draw_sprite(buffer, strato5, sposta5, 0); // CARICA E SPOSTA IL LIVELLO DI PARALLELASSE //
draw_sprite(buffer, strato4, sposta4, 40); // CARICA E SPOSTA IL LIVELLO DI PARALLELASSE //
draw_sprite(buffer, strato3, sposta3, 400); // CARICA E SPOSTA IL LIVELLO DI PARALLELASSE //
draw_sprite(buffer, strato2, sposta2, 440); // CARICA E SPOSTA IL LIVELLO DI PARALLELASSE //
draw_sprite(buffer, strato1, sposta1, 460); // CARICA E SPOSTA IL LIVELLO DI PARALLELASSE //
}

void spriteprincipale() // INIZIO PROCEDURA SPRITEPRINCIPALE //
{
if (beastfermo==1) beastfermo=0; // CONTROLLA CHE LO SPRITE PRINCIPALE SIA FERMO //
if (key[KEY_RIGHT]) { // CONTROLLA CHE IL TASTO DIREZIONALE 'DESTRO' SIA PREMUTO
//
if (tastopremuto==0 || tastopremuto==1) { // CONTROLLA CHE LO SPRITE PRINCIPALE SIA
FERMO O CHE SIA PREMUTA SOLAMENTE LA DIREZIONE DESTRA //
tastopremuto=1; // INDICA E' PREMUTA LA DIREZIONE DESTRA //
beastfermo=1; // LA VARIABILE ASSUME IL VALORE 1 PERCHE LO SPRITE PRINCIPALE E' IN
MOVIMENTO //

destra=1; sinistra=0; // INDICA CHE LO SPRITE PRINCIPALE E' RIVOLTO VERSO DESTRA //

if (tempriparte==0) temporizzatore=0; tempriparte=1; // SE IL TEMPORIZZATORE NON E' A ZERO
LO METTE A ZERO //
if (beastani>=7) beastani=1; // SE SUPERA IL SETTIMO FRAME DI ANIMAZIONE DELLO SPRITE
PRINCIPALE RIPARTE DA 1 //
```

```
draw_sprite(buffer, dat[beastani].dat, x, y); // DISEGNA IL FRAME CORRENTE SUL BUFFER //

if (temporizzatore>=intervallo) { // SE IL TEMPORIZZATORE E' SUPERIORE O UGUALE ALLA PAUSA ...//
beastani=beastani+1; temporizzatore=0; // ... AVANZA DI UN FRAME E RIPONE IL TEMPORIZZATORE A ZERO //
}
}
}
if (key[KEY_LEFT]) { // CONTROLLA CHE IL TASTO DIREZIONALE 'SINISTRO' SIA PREMUTO //
if (tastopremuto==0 || tastopremuto==2) { // CONTROLLA CHE LO SPRITE PRINCIPALE SIA FERMO O CHE SIA PREMUTA SOLAMENTE LA DIREZIONE SINISTRA //
tastopremuto=2; // INDICA E' PREMUTA LA DIREZIONE SINISTRA //
beastfermo=1; // LA VARIABILE ASSUME IL VALORE 1 PERCHE LO SPRITE PRINCIPALE E' IN MOVIMENTO //
sinistra=1; destra=0; // INDICA CHE LO SPRITE PRINCIPALE E' RIVOLTO VERSO SINISTRA //

if (tempriparte==0) temporizzatore=0; tempriparte=1; // SE IL TEMPORIZZATORE NON E' A ZERO LO METTE A ZERO //
if (beastani>=7) beastani=1; // SE SUPERA IL SETTIMO FRAME DI ANIMAZIONE DELLO SPRITE PRINCIPALE RIPARTE DA 1 //

draw_sprite_h_flip(buffer, dat[beastani].dat, x, y); // DISEGNA IL FRAME CORRENTE SUL BUFFER A SPECCHIO //
if (temporizzatore>=intervallo) { // SE IL TEMPORIZZATORE E' SUPERIORE O UGUALE ALLA PAUSA ...//
beastani=beastani+1; temporizzatore=0; // ... AVANZA DI UN FRAME E RIPONE IL TEMPORIZZATORE A ZERO //
}
}
}

if (key[KEY_DOWN]) { // CONTROLLA CHE IL TASTO DIREZIONALE 'BASSO' SIA PREMUTO //
if (tastopremuto==0 || tastopremuto==3) { // CONTROLLA CHE LO SPRITE PRINCIPALE SIA FERMO O CHE SIA PREMUTA SOLAMENTE LA DIREZIONE BASSO //
tastopremuto=3; // INDICA E' PREMUTA LA DIREZIONE BASSO //
beastfermo=1; // LA VARIABILE ASSUME IL VALORE 1 PERCHE LO SPRITE PRINCIPALE E' IN MOVIMENTO //
if (tempriparte==0) temporizzatore=0; tempriparte=1; // SE IL TEMPORIZZATORE NON E' A ZERO LO METTE A ZERO //


if (destra==1) { // SE LO SPRITE PRINCIPALE E RIVOLTO A DESTRA ... //
draw_sprite(buffer, dat[beastaniaccosciata].dat, x, y+15); // DISEGNA IL FRAME CORRENTE SUL BUFFER //
}
if (sinistra==1) { // SE LO SPRITE PRINCIPALE E RIVOLTO A SINISTRA ... //
draw_sprite_h_flip(buffer, dat[beastaniaccosciata].dat, x, y+15); // DISEGNA IL FRAME CORRENTE SUL BUFFER A SPECCHIO //
}
if (temporizzatore>=intervallo) { // SE IL TEMPORIZZATORE E' SUPERIORE O UGUALE ALLA PAUSA ...//

beastaniaccosciata=beastaniaccosciata+1;temporizzatore=0; // ... AVANZA DI UN FRAME E RIPONE IL TEMPORIZZATORE A ZERO //
```

```
if (beastaniaccosciata>=9) beastaniaccosciata=8; // SE SUPERA IL NONO FRAME DI ANIMAZIONE DELLO SPRITE PRINCIPALE RIPARTE DA OTTO - CIOE' RIMANE IN ACCOSCIATA //
}
}
}
else {
if (beastaniaccosciata>=8) beastaniaccosciata=7; // SE E' UGUALE O MAGGIORE ALL'OTTAVO FRAME RIPARTE DAL SETTIMO //
}

if (key[KEY_UP]) { // CONTROLLA CHE IL TASTO DIREZIONALE 'SALTO' SIA PREMUTO //
if (tastopremuto==0 || tastopremuto==4) { // CONTROLLA CHE LO SPRITE PRINCIPALE SIA FERMO O CHE SIA PREMUTA SOLAMENTE LA DIREZIONE ALTO //
tastopremuto=4; // INDICA E' PREMUTA LA DIREZIONE ALTO //
beastfermo=1; // LA VARIABILE ASSUME IL VALORE 1 PERCHE LO SPRITE PRINCIPALE E' IN MOVIMENTO //

if (tempriparte==0) temporizzatore=0; tempriparte=1; // SE IL TEMPORIZZATORE NON E' A ZERO LO METTE A ZERO //
if (destra==1) { // SE LO SPRITE PRINCIPALE E RIVOLTO A DESTRA ... //
draw_sprite(buffer, dat[9].dat, x, y); // DISEGNA IL FRAME CORRENTE SUL BUFFER //
}
if (sinistra==1) { // SE LO SPRITE PRINCIPALE E RIVOLTO A SINISTRA ... //
draw_sprite_h_flip(buffer, dat[9].dat, x, y); // DISEGNA IL FRAME CORRENTE SUL BUFFER A SPECCHIO//
}
}
}

if (key[KEY_SPACE]) { // CONTROLLA CHE IL TASTO DIREZIONALE 'BASSO' SIA PREMUTO //
if (tastopremuto==0 || tastopremuto==5) { // CONTROLLA CHE LO SPRITE PRINCIPALE SIA FERMO O CHE SIA PREMUTA SOLAMENTE LA DIREZIONE BASSO //
tastopremuto=5; // INDICA E' PREMUTA LA DIREZIONE BASSO //
beastfermo=1; // LA VARIABILE ASSUME IL VALORE 1 PERCHE LO SPRITE PRINCIPALE E' IN MOVIMENTO //
if (tempriparte==0) temporizzatore=0; tempriparte=1; // SE IL TEMPORIZZATORE NON E' A ZERO LO METTE A ZERO //
if (destra==1) { // SE LO SPRITE PRINCIPALE E RIVOLTO A DESTRA ... //
draw_sprite(buffer, dat[beastanipugno].dat, x, y+6); // DISEGNA IL FRAME CORRENTE SUL BUFFER //
}
if (sinistra==1) { // SE LO SPRITE PRINCIPALE E RIVOLTO A SINISTRA ... //
if (beastanipugno==18) corretto=-32;
draw_sprite_h_flip(buffer, dat[beastanipugno].dat, x+8+corretto, y+6); // DISEGNA IL FRAME CORRENTE SUL BUFFER A SPECCHIO//
}
if (temporizzatore>=intervallo) { // SE IL TEMPORIZZATORE E' SUPERIORE O UGUALE ALLA PAUSA ...//
beastanipugno=beastanipugno+1;temporizzatore=0; // ... AVANZA DI UN FRAME E RIPONE IL TEMPORIZZATORE A ZERO //
if (beastanipugno>=19) beastanipugno=18; // SE SUPERA IL NONO FRAME DI ANIMAZIONE DELLO SPRITE PRINCIPALE RIPARTE DA OTTO - CIOE RIMANE IN ACCOSCIATA //
}
}
}
else {
```

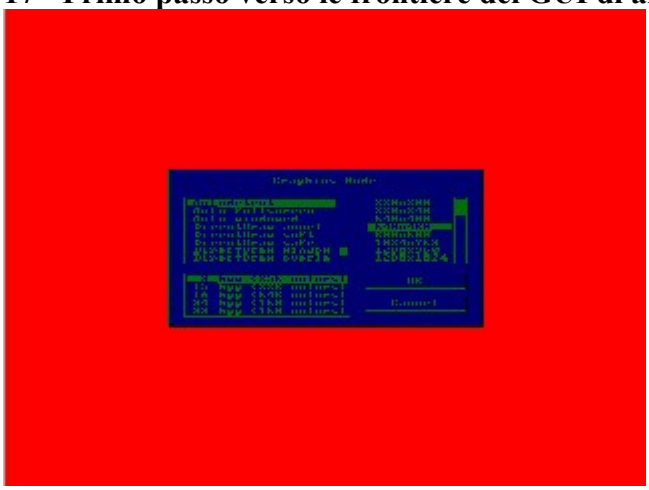
```
if (beastanipugno>=17) beastanipugno=16; // SE E' UGUALE O MAGGIORE ALL'OTTAVO FRAME
RIPARTE DAL SETTIMO //
}
if (beastfermo==0) { // SE LO SPRITE PRINCIPALE E' FERMO ... //
corretto=0;
tempriparte=0; // INTERROMPE LA PARTENZA DA ZERO DEL TEMPORIZZATORE //
tastopremuto=0; // INDICA CHE NON E' PREMUTO ALCUN TASTO //
if (destra==1) { // SE LO SPRITE PRINCIPALE E RIVOLTO A DESTRA ... //
draw_sprite(buffer, dat[10].dat, x, y); // DISEGNA IL FRAME CORRENTE SUL BUFFER //
}
if (sinistra==1) { // SE LO SPRITE PRINCIPALE E RIVOLTO A SINISTRA ... //
draw_sprite_h_flip(buffer, dat[10].dat, x, y); // DISEGNA IL FRAME CORRENTE SUL BUFFER A
SPECCHIO//
}
}
}
void doppiobuffering() // INIZIO PROCEDURA DI DOPPIO BUFFERING //
{
vsync(); // SINCRONIZZAZIONE AGGIORNAMENTO SCHERMO //
blit(buffer, screen, 0, 0, 0, 0, 640, 480); // VISUALIZZA IL CONTENUTO DI BUFFER SU SCHERMO
//
clear(buffer); // PULISCE IL BUFFER //
}

void fotogrammi() {
if (frame==0) {
if (tempo1==0) tempo1= time(0);
if (time(0)-10>tempo1) {
tempo1=0; frame=1; }
fps++; intervallo =fps /60;
}
sprintf(score, "FP 10 S: %ld", (long)fps); // da il valore a score e a vite da visualizzare //
textout(buffer, font, score, 1, 400, 2000000); // inserisce nel doppio buffer dello schermo il valore di score
e vite che verra poi visualizzato //
}



int main() /* INIZIO PROCEDURA */
{
PALETTE colori; /* DICHIARAZIONE VARIABILE COLORI */
allegro_init(); /* INIZIALIZZA LA LIBRERIA ALLEGRO */
install_keyboard(); /* INSTALLA LA TASTIERA */

install_sound(DIGI_AUTODETECT, MIDI_AUTODETECT, 0);
set_color_depth(32); /* IMPOSTA LA PALETTE DEI COLORI A 32 BIT - TRUE COLOR */
set_gfx_mode(GFX_GDI, 640, 480, 0, 0); /* IMPOSTA LA MODALITA' GRAFICA 640x480 */
dat = load_datafile("beast.dat"); // CARICA IL FILE DAT //

set_palette(colori); // PRENDE LA PALETTE CONTENUTA NEL FILE DAT - PER IL TRUE COLOR
NON E' NECESSARIO //
buffer = create_bitmap(640, 480); // CREA LA DIMENSIONE IN PIXEL DEL BUFFER DA
VISUALIZZARE SULLO SCHERMO //
set_volume(255,255);

play_midi(dat[wmidimusic].dat, TRUE);
```

```
strato5 = create_bitmap(960, 200); // CREA UN BUFFER BITMAP DI 640x200 PIXEL //
clear(strato5); /* PULISCE IL BUFFER */
strato1 = create_bitmap(3200, 200); // CREA UN BUFFER BITMAP DI 3200x200 PIXEL //
clear(strato1); /* PULISCE IL BUFFER */
strato2 = create_bitmap(2560, 200); // CREA UN BUFFER BITMAP DI 2560x200 PIXEL //
clear(strato2); /* PULISCE IL BUFFER */
strato3 = create_bitmap(1920, 200); // CREA UN BUFFER BITMAP DI 1920x200 PIXEL //
clear(strato3); /* PULISCE IL BUFFER */
strato4 = create_bitmap(1280, 200); // CREA UN BUFFER BITMAP DI 1280x200 PIXEL //
clear(strato4); /* PULISCE IL BUFFER */

for (d=0;e<=960;e=e+320) { /* MOLTIPLICA IL BITMAP PER TUTTA LA LUNGHEZZA DELLA
PARTE DA SCROLLARE */
blit(dat[15].dat, strato5, 0, 0, e, 0, 960, 200);
}

for (d=0;d<=1280;d=d+256) { /* MOLTIPLICA IL BITMAP PER TUTTA LA LUNGHEZZA DELLA
PARTE DA SCROLLARE */
blit(dat[14].dat, strato4, 0, 0, d, 0, 640, 200);
}

for (c=0;c<=1920;c=c+128) { /* MOLTIPLICA IL BITMAP PER TUTTA LA LUNGHEZZA DELLA
PARTE DA SCROLLARE */
blit(dat[13].dat, strato3, 0, 0, c, 0, 640, 200);
}

for (b=0;b<=2560;b=b+64) { /* MOLTIPLICA IL BITMAP PER TUTTA LA LUNGHEZZA DELLA
PARTE DA SCROLLARE */
blit(dat[12].dat, strato2, 0, 0, b, 0, 640, 200);
}

for (a=0;a<=3200;a=a+32) { /* MOLTIPLICA IL BITMAP PER TUTTA LA LUNGHEZZA DELLA
PARTE DA SCROLLARE */
blit(dat[11].dat, strato1, 0, 0, a, 0, 640, 200);
}
beastani=1; // NUMERO DEL FRAME DI PARTENZA DELLO SPRITE PRINCIPALE NELLA
CORSA //
beastaniaccosciata=7; // NUMERO DEL FRAME DI PARTENZA DELLO SPRITE PRINCIPALE
NELL'ACCOSCIATA //
beastanipugno=16;
intervallo=15; // VALORE DI INTERVALLO TRA UN FRAME E L'ALTRO //
beastfermo=0; // VALORE CHE ATTIVA LA POSIZIONE DI FERMO NELLO SPRITE
PRINCIPALE //
destra=1; sinistra=0; // VALORI CHE INDICANO ALLO SPRITE PRICIPALE LA POSIZIONE IN
CUI ESSO E' RIVOLTO //
tempriparte=1; // VALORE CHE INDICA AL TEMPORIZZATORE DI RIPARTIRE A CONTARE DA
0 //
tastopremuto=0; // IMPOSTA LA VARIABILE A ZERO (NESSUN TASTO PREMUTO) //
x=300; y=370;
tempo1=0;
frame=0;
while (!key[KEY_ESC]) {
draw_sprite(buffer, dat[titoli].dat, 160, 50);
temporizzatore++; // TEMPORIZZATORE CHE INCREMENTA DI UN'UNITA' AD OGNI CICLO //
fotogrammi();
```

```
scorrimento();
spriteprincipale(); // CARICA LA PROCEDURA DELLO SPRITE PRINCIPALE //
doppiobuffering(); // CARICA LA PROCEDURA DI DOPPIOBUFFERING PER LA
VISUALIZZAZIONE SU SCHERMO //
}
destroy_bitmap(immagine1); /* ELIMINA L'IMMAGINE DAL BUFFER PRIMA DI USCIRE DAL
PROGRAMMA */
destroy_bitmap(immagine2); /* ELIMINA L'IMMAGINE DAL BUFFER PRIMA DI USCIRE DAL
PROGRAMMA */
destroy_bitmap(immagine3); /* ELIMINA L'IMMAGINE DAL BUFFER PRIMA DI USCIRE DAL
PROGRAMMA */
destroy_bitmap(immagine4); /* ELIMINA L'IMMAGINE DAL BUFFER PRIMA DI USCIRE DAL
PROGRAMMA */
destroy_bitmap(immagine5); /* ELIMINA L'IMMAGINE DAL BUFFER PRIMA DI USCIRE DAL
PROGRAMMA */
destroy_sample(suono);
allegro_exit(); /* TERMINA LA LIBRERIA ALLEGRO */
}
END_OF_MAIN (); /* FINE DEL PROGRAMMA */
```

Analizziamo le parti salienti del programma BEAST.C:
Il demo che ho presentato è molto semplice, nonostante questo riunisce vari sotto programmi di natura diversa per creare “qualcosa che funziona”. Un componente analizzato in questo listato è la parte che per mezzo di un conteggio dei fotogrammi della durata di 10 secondi viene determinata la velocità del computer su cui è avviato il programma. Tale calcolo garantisce la velocità di esecuzione ideale su tutti i terminali nei quali è eseguito.

17 - Primo passo verso le frontiere del GUI di allegro.h



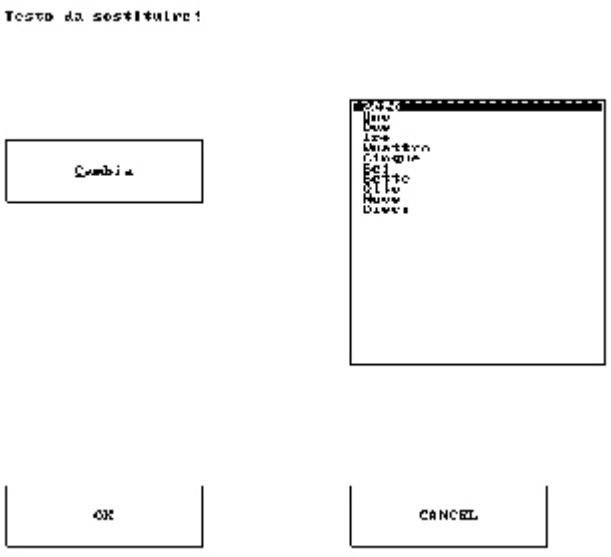
```
// Programma minigui.c
// Realizzato da Bartolomeo Davide Bertinetto
// www.bertinettobartolomeodavide
// contatto@bertinettobartolomeodavide.it

#include <stdio.h>
#include "allegro.h"

// Testo della stringa da visualizzare
char stringa[80] = "La modalita' video va bene?";
```

```
int b=0;
// Procedura principale
main()
{
// Seconda stringa per il passaggio da 'sprintf' ad 'alert'
char messaggio1[80];

// Inizializzaioni necessarie
allegro_init();
install_keyboard();
install_mouse();
install_timer();

// Modalità dello schermo per il menù di scelta
set_gfx_mode(GFX_GDI, 640, 480, 0, 0);

// Comando che permette il trasferimento della stringa al comando finale di visualizzazione
sprintf(messaggio1, "Messaggio: %s", stringa);

// ALETERNATIVA:
// assegna la palette di desktop classica di allegro (palette base del vecchio atari ST in modalità desktop!) che dopo la scelta ritornerà quella di schermo per il bit plane selezionato
// set_palette(desktop_palette);

// ALTERNATIVA:
// seguono alcuni comandi che assegnano un colore diverso dal desktop classico di allegro.h in GUI(in questo caso non è necessario utilizzare 'set_palette(desktop_palette);')
// colore testo e bordi della finestra di dialogo (r,g,b)
gui_fg_color = makecol(0, 128, 0);
// colore sfondo della finestra di dialogo (r,g,b)
gui_bg_color = makecol(0, 0, 128);
// colore di sfondo schermo (r,g,b)
clear_to_color(screen, makecol(255, 0, 0));

// inizializza la variabile per la scelta della modalità video
int c = GFX_AUTODETECT;
// assegna la massima risoluzione video possibile asse X
int w = SCREEN_W;
// assegna la massima risoluzione video possibile asse Y
int h = SCREEN_H;
// assegna la profondità di colore
int bpp = bitmap_color_depth(screen);
// comando che permette la creazione del menù: Modalità video
gfx_mode_select_ex(&c, &w, &h, &bpp);
// concretizza la scelta per la profondità di colore
set_color_depth(bpp);
// concretizza la modalità video scelta e la risoluzione
set_gfx_mode(c, w, h, 0, 0);



// Il comando 'alert' visualizza una finestra con fino a tre messaggi visualizzabili e crea fino a due bottoni
// alert(messaggio1 è il contenuto della stringa, "Prendi una decisione" e "Premendo uno dei tasti seguenti:" sono visualizzatti direttamente come si presentano tra virgolette,
// "tasto Ok" e "tasto Cancel" rappresentano il messaggio che comparirà sui tasti stessi,
// 0 indica che la condizione assegnerà '1' come risultalo alla pressione del primo tasto,
```

```
// 1 darà '0' al risultato della condizione (se fosse digitato NULL indicherebbe che il secondo tasto è inattivo),
// il tutto è inserito in una condizione 'if' dove, se il risultato è '1' la condizione viene eseguita.

if (alert(messaggio1,"Prendi una decisione","Premendo uno dei tasti seguenti:", "tasto Ok", "tasto Cancel",0,1)==1) {

// 'gui_textout_ex' è un comando di visualizzazione testo dove: screen è la destinazione di visualizzazione, "hai premuto 'Ok'" è la stringa che sarà visualizzata,
// SCREEN_W/2 e SCREEN_H/2 rappresentano il centro dello schermo di inizio messaggio,
// 128 è il colore del messaggio, 255 è il colore del bordo messaggio, 1 indica che il messaggio è centrato (0=non centrato)

gui_textout_ex(screen, "Hai premuto 'ok', premi un tasto da tastiera per uscire...", SCREEN_W/2, SCREEN_H/2, 128, 255, 1);

// attende la pressione di un tasto da tastiera
readkey();
}
}

END_OF_MAIN();
```

Analizziamo le parti salienti del programma MINIGUI.C:
Allegro.h possiamo considerarla una libreria tutto fare. Infatti le risorse non sono limitate solamente alla grafica ed al suono. Questa libreria è in grado di darci la possibilità di costruire ogni tipo di menù, pulsante, finestra, ecc... Rendendo le nostre applicazioni e giochi vicini alle prerogative che i sistemi operativi con interfaccia grafica di oggi ci hanno abituato.
Lo stile delle finestre adottato in questo caso forse non molti se lo ricorderanno. In allegro.h l'interfaccia GUI riprende quelle usata dell'ormai sepolto sistema Atari Gem. Quello dell'Atari ST per i nostalgici e pure quello della poco diffusa serie di PC sempre targati Atari... Tutto questo potrà rivivere ancora oggi nelle nostre creazioni sui computer moderni!
Questo primo listato sulle possibilità GUI di allegro.h ci apre le porte per: un menù sulla scelta della modalità grafica, scelta dei colori delle finestre e utilizzo di pulsanti legati hai relativi eventi.

18 - Secondo passo verso le frontiere del GUI di allegro.h

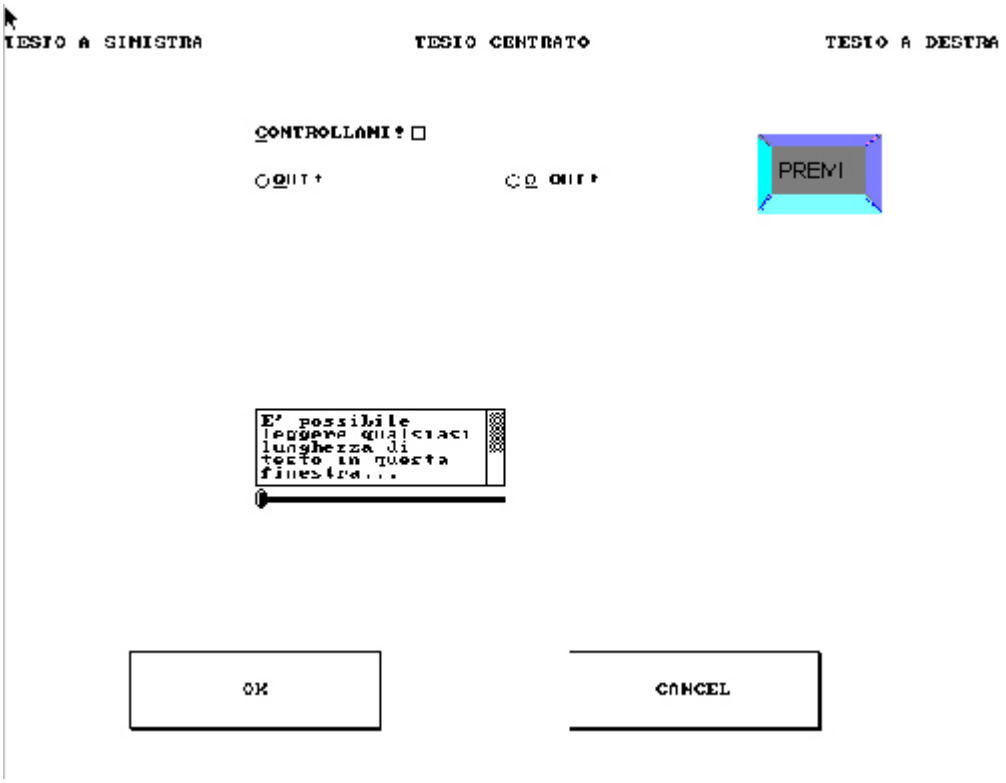


```
// Programma miniguidue.c
// Realizzato da Bartolomeo Davide Bertinetto
// www.bertinettobartolomeodavide
// contatto@bertinettobartolomeodavide.it

// Carico le librerie
#include <stdio.h>
#include "allegro.h"

// inizializzo una stringa lunga al massimo 32 caratteri
char stringa[32] = "Testo da sostituire!";

// Creo la procedura per l'array di selezione della lista numerica da uno a dieci
char *finestra_selezione(int indice, int *lunghezza_lista)
{
// Stringhe del corrispondente numerico da visualizzare in finestra
static char *stringhe[] =
{
"Zero", "Uno", "Due", "Tre", "Quattro", "Cinque",
"Sei", "Sette", "Otto", "Nove", "Dieci"
};

// Determino la lunghezza dell'array che in questo caso arriva ad 11 valori (da 0 a 10)
if (indice < 0) {
*lunghezza_lista = 11;
return NULL;
}
else
return stringhe[indice];
}
// Creo la finestra di dialogo su schermo in 'finestra_dialogo'
DIALOG finestra_dialogo[] =
{
// COMANDO POS. X,Y |ALTEZ.,LARG.|COL. TESTO|COL. SFON.|TASTO|RISULTATO|
SELEZIONE| -- |TES./STRING.|-- |--
/* (dialog proc) (x) (y) (w) (h) (fg) (bg) (key) (flags) (d1) (d2) (dp) (dp2)(dp3) */
{ d_edit_proc, 80, 32, 512, 48, 255, 0, 0, 0, sizeof(stringa)-1,0,stringa, NULL, NULL },
{ d_button_proc, 80, 132, 161, 49, 255, 0, 'c', 0, 0, 0, "&Cambia", NULL, NULL },
{ d_list_proc, 360,100, 207, 207, 255, 0, 0, 0, 0, 0, finestra_selezione,NULL, NULL },
{ d_button_proc, 80, 400, 161, 49, 255, 0, 0, D_EXIT, 0, 0, "OK", NULL, NULL },
{ d_button_proc, 360,400, 161, 49, 255, 0, 0, D_EXIT, 0, 0, "CANCEL", NULL, NULL },
{ NULL, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, NULL, NULL, NULL }
};

// Definizione del nome oggetto e del numero di comando(linea da 0...) da richiamare in 'finestra_dialogo'
inserita sopra
#define OGGETTO_LISTA 2
#define TASTO_PREMUTO 1

// corpo principale main
int main()
{

// Dimensione in numero caratteri delle tre stringhe di conversione in 'sprintf' (buf1,buf2,buf39
```

```
char buf1[80], buf2[80], buf3[80];

// Variabile destinazione del risultato numerico di 'finestr_dialogo' per i tasti 'OK' e 'CANCEL',
rispettivamente in questo caso '3' e '4'
int ret;

// inizializzazione generali
allegro_init();
install_keyboard();
install_mouse();
install_timer();

// Modalità grafica
set_gfx_mode(GFX_GDI, 640, 480, 0, 0);

// Scelta palette standard del desktop allegro.h in GUI
set_palette(desktop_palette);

// Trasferisce il risultato numerico di 'finestra_dialogo' in 'ret'
ret = do_dialog(finestra_dialogo, -1);

// comandi 'sprintf' di trasferimento ed inserimento variabili nelle stringhe di caratteri 'buf...' da utilizzare
nel comando 'alert'
sprintf(buf1, "Se premo 'Ok' ho '3', mentre se premo 'CANCEL' ho '4', quindi ho premuto: %d", ret);
sprintf(buf2, "Stringa inserita: '%s' e selezione tasto premuto: '%d'", stringa,
finestra_dialogo[TASTO_PREMUTO].flags);
sprintf(buf3, "Ho scelto il numero: %d", finestra_dialogo[OGGETTO_LISTA].d1);

// Visualizza una finestra al centro dello schermo contenente il risultato finale con testo e la pressione di
un tasto conclusivo 'OK'
alert(buf1, buf2, buf3, "OK", NULL, 0, 0);
}

END_OF_MAIN();
```

Analizziamo le parti salienti del programma MINIGUIDUE.C:
Il listato di 'miniguidue.c' prende in esame un menù di selezione completamente personalizzabile con i relativi eventi che ne derivano, un comando di 'edit' del testo con traferimento ad un stringa, un tasto di selezione 'acceso o spento' e due tasti 'OK e CANCEL' con relativa risposta numerica a seconda della scelta per determinare un possibile evento. Il risultato finale è inserito in una finestra al centro dello schermo data dal comando 'alert' con relativo tasto di conferma.
Questo programma C da la possibilità all'utente di realizzare concretamente delle vere interfacce 'punta e clicca' nei software sviluppati in modo semplice e chiaro.

19 - Terzo passo verso le frontiere del GUI di allegro.h




```
// Programma miniguitre.c
// Realizzato da Bartolomeo Davide Bertinetto
// www.bertinettobartolomeodavide
// contatto@bertinettobartolomeodavide.it

// Carico le librerie
#include <stdio.h>
#include "allegro.h"

// Stringa di caratteri lunga a piacimento che inserita nella casella di testo
char testo[] = "E' possibile leggere qualsiasi lunghezza di testo in questa finestra... \
Quindi se volete inserire una frase modificate questa stringa!";

// Creo la finestra di dialogo su schermo in 'finestra_dialogo'
DIALOG finestra_dialogo[] =
{
// COMANDO POS. X,Y |ALT.,LARG.|COL. |COL. SFON.|TASTO|RISUL.|SELEZ.|SELEZ.|
TEST./STRI. |-- |--
/* (dialog proc) (x) (y) (w) (h) (fg) (bg) (key) (flags) (d1) (d2) (dp) (dp2) (dp3) */

{ d_button_proc, 80, 400, 161, 49, 255, 0, 0, D_EXIT, 0, 0, "OK", NULL, NULL },
{ d_button_proc, 360, 400,161, 49, 255, 0, 0, D_EXIT, 0, 0, "CANCEL", NULL, NULL },
{ d_slider_proc, 160, 300, 160, 12, 255, 0, 0, 0, 100, 0, NULL, NULL, NULL },
{ d_textbox_proc, 160, 250, 160, 48, 255, 0, 0, 0, 0, 0, (void *)testo, NULL, NULL },
{ d_icon_proc, 480, 80, 80, 50, 0, 255, 0, 0, 0, 0, NULL, NULL, NULL },
{ d_text_proc, 0, 20, 0, 0, 255, 0, 0, 0, 0, 0, "TESTO A SINISTRA",NULL, NULL },
{ d_ctext_proc, 318, 20, 0, 0, 255, 0, 0, 0, 0, 0, "TESTO CENTRATO",NULL, NULL },
{ d_rtext_proc, 636, 20, 0, 0, 255, 0, 0, 0, 0, 0, "TESTO A DESTRA", NULL, NULL },
{ d_radio_proc, 160, 100, 160, 19, 255, 0, 'q', 0, 0, 0, "&QUI!", NULL, NULL },
{ d_radio_proc, 320, 100, 160, 19, 255, 0, 'o', 0, 0, 0, "&O QUI!", NULL, NULL },
{ d_check_proc, 160, 70, 160, 20, 255, 0, 'c', 0, 0, 0, "&CONTROLLAMI!", NULL, NULL },
{ NULL, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, NULL, NULL, NULL }
};
```

```
// Definizione del nome oggetto e del numero di comando(linea da 0...) da richiamare in 'finestra_dialogo' inserita sopra
#define SLIDER_SELETTORE 2
#define ICONA 4
#define RADIO_QUI 8
#define RADIO_OQUI 9
#define CONTROLLO 10

// corpo principale main
int main()
{

// Dimensione in numero caratteri delle tre stringhe di conversione in 'sprintf' (buf1,buf2,buf3)
char buf1[80], buf2[80], buf3[80];
// Variabile destinazione del risultato numerico di 'finestr_dialogo' per i tasti 'OK' e 'CANCEL', rispettivamente in questo caso '3' e '4'
int ret;

// Determino la palette delle immagini con la variabile 'palette'
PALETTE colori;

// inizializzazione generali
allegro_init();
install_keyboard();
install_mouse();
install_timer();

// Modalità grafica
set_gfx_mode(GFX_GDI, 640, 480, 0, 0);

// Scelta palette standard del desktop allegro.h in GUI
set_palette(desktop_palette);

// Assegno l'immagine all'icona caricando il file '.bmp' da disco
finestra_dialogo[ICONA].dp = load_bitmap("icona.bmp", colori);

// Trasferisce il risultato numerico di 'finestra_dialogo' in 'ret'
ret = do_dialog(finestra_dialogo, -1);

// comandi 'sprintf' di trasferimento ed inserimento variabili nelle stringhe di caratteri 'buf...' da utilizzare nel comando 'alert'
sprintf(buf3, "Se icona premuta visualizza '2' altrimenti '0', quindi: %i!", finestra_dialogo[ICONA].flags);
sprintf(buf2, "Posizione selettore slider: %i!- QUI: %i - O QUI: %i",
finestra_dialogo[SLIDER_SELETTORE].d2, finestra_dialogo[RADIO_QUI].flags,
finestra_dialogo[RADIO_OQUI].flags);
sprintf(buf1, "Se premo 'Ok' ho '0', mentre se premo 'CANCEL' ho '1', quindi ho premuto: %d", ret);

// Se è spuntata (flags maggiore di 0) la casella di controllo compare il messaggio di testo
if (finestra_dialogo[CONTROLLO].flags>0) gui_textout_ex(screen, "OKAY!!! CONTROLLO SELEZIONATO" ,SCREEN_W/2, 470, 128, 255, 1);

// Visualizza una finestra al centro dello schermo contenente il risultato finale con e la pressione di un tasto conclusivo 'OK'
alert(buf1, buf2, buf3, "OK", NULL, 0, 0);
}
```

END_OF_MAIN();

Analizziamo le parti salienti del programma MINIGUITRE.C:
Nel sorgente appena esposto vengono presi in considerazioni molti aspetti delle funzionalità GUI di allegro.h., come: posizione del testo su schermo proprio come se si trattasse di un programma di video scrittura, pressione di tasti-icona in bitmap, oggetti di selezione e spunta, finestre di testo con barre di scorrimento, selettori slider, ecc... Se nei listati di sviluppo GUI precedenti di allegro.h avevamo già analizzato molti componenti fondamentali, quelli ora contemplati conferiscono un sicuro tocco di classe alle nostre creazioni!

20 - Ultimo passo verso le frontiere del GUI di allegro.h

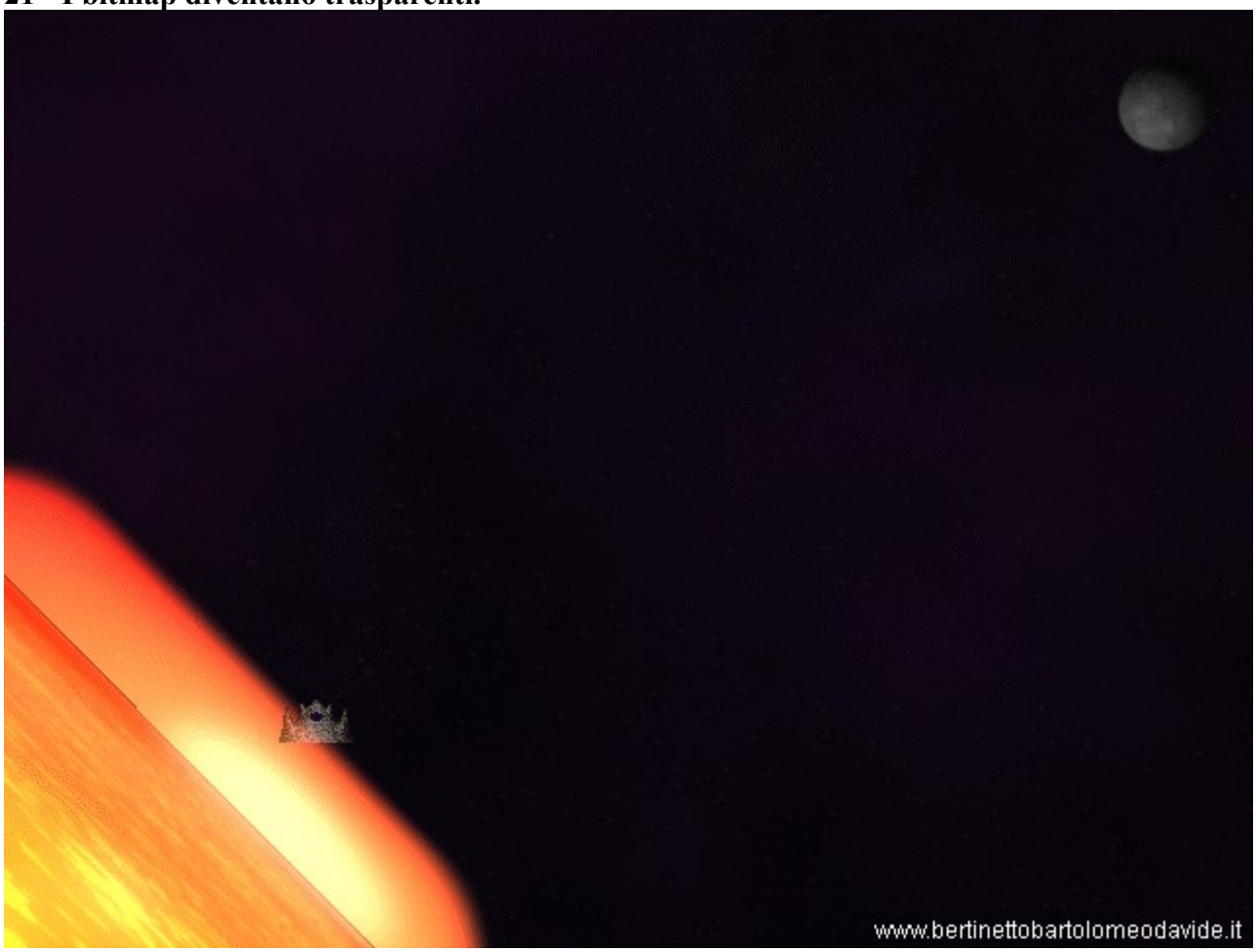
```
// Programma miniguifinal.c
// Realizzato da Bartolomeo Davide Bertinetto
// www.bertinettobartolomeodavide
// contatto@bertinettobartolomeodavide.it

#include <stdio.h> // LIBRERIA DI SISTEMA
#include "allegro.h" /* INCLUSA LA LIBRERIA ALLEGRO.H */

/* Menù di chiama per finestra di uscita */
int Esci(void)
{
if (alert("Vuoi veramente uscire?", NULL, NULL, "&Si", "&No", 's', 'n') == 1)
return D_CLOSE;
else
return D_O_K;
}

/* finestra di messaggio. */
int Informazioni(void)
{
alert("* Menu' a tendina *",
"",
"Esempio Allegro GUI",
"Ok", 0, 0, 0);
```

```
return D_O_K;
}

/* Finestra di chiamata per messaggio risposta. */
int risposta(void)
{
char str[256];
ustrzcpy (str, sizeof str, active_menu->text);
alert("Scelta Numero: ", "", ustrtok (str, "\t"), "Ok", NULL, 0, 0);
return D_O_K;
}

/* Procedura che assegna un risultato a seconda che la voce "controllo" sia selezionata o meno */
int chimata_controllo(void)
{
// 'active_menu' controlla so stato del flag
active_menu->flags ^= D_SELECTED;
if (active_menu->flags & D_SELECTED)

// 'active_menu' restituisce nel menù stesso la dicitura "controllo' o 'non-controllo'
active_menu->text = "Controllo";
else
active_menu->text = "Non-controllo";

// Segue il messaggio 'alert' indicante che qualcosa è cambiato
alert("Il menu' e' stato cambiato!", NULL, NULL, "Ok", NULL, 0, 0);

// Con di 'D_O_K' il programma continua e con 'D_CLOSE' termina
return D_O_K;
}

// sotto menù con funzione di controllo
MENU sottomenu[] =
{
// Nome comando |Chiama procedura |Chiama sotto-menù |flags |--
{ "Sotto-menu'", NULL, NULL, D_DISABLED, NULL },
{ "", NULL, NULL, 0, NULL },
{ "Controllo", chimata_controllo, NULL, D_SELECTED, NULL },
{ "Disabilitato", NULL, NULL, D_DISABLED, NULL },
{ NULL, NULL, NULL, 0, NULL }
};

// Primo sotto-menù
MENU menu1[] =
{
// Nome comando |Chiama procedura |Chiama sotto-menù |flags |--
{ "Risposta &1", risposta, NULL, 0, NULL },
{ "Risposta &2", risposta, NULL, 0, NULL },
{ "&Esci", Esci, NULL, 0, NULL },
{ NULL, NULL, NULL, 0, NULL }
};

// Secondo sotto-menù
MENU menu2[] =
{
// Nome comando |Chiama procedura |Chiama sotto-menù |flags |--
```

```
{ "&Test", risposta, NULL, 0, NULL },
{ "&Sotto menu'", NULL, sottomenu, 0, NULL },
{ NULL, NULL, NULL, 0, NULL }
};

// Sotto-menù aiuto
MENU menuaiuto[] =
{

// Nome comando |Chiama procedura |Chiama sotto-menù |flags |--
{ "&Informazioni ", Informazioni, NULL, 0, NULL },
{ NULL, NULL, NULL, 0, NULL }
};

// Menù principale
MENU il_menu[] =
{

// Nome comando |Chiama procedura |Chiama sotto-menù |flags |--
{ "&Uno", NULL, menu1, 0, NULL },
{ "&Due", NULL, menu2, 0, NULL },
{ "&Aiuto", NULL, menuaiuto, 0, NULL },
{ NULL, NULL, NULL, 0, NULL }
};

// Creo la finestra di dialogo su schermo in 'finestra_dialogo'
DIALOG finestra_dialogo[] =
{
// COMANDO POS. X,Y |ALT.,LARG.|COL.|COL. SFO.|TASTO|RISUL.|SEL.|SEL.|TEST./STRING.
|-- |--
/* (dialog proc) (x) (y) (w) (h) (fg) (bg) (key) (flags) (d1) (d2) (dp) (dp2) (dp3) */
{ d_menu_proc, 160, 0, 0, 0, 255, 0, 0, 0, 0, 0, il_menu, NULL, NULL },
{ NULL, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, NULL, NULL, NULL }
};

// INIZIO PROCEDURA DI ALLEGRO //
int main()
{

// INIZIALIZZA ALLEGRO //
allegro_init();
install_keyboard();
install_mouse();

// SELEZIONA LA PROFONDITA' DI COLORE A 32 BIT //
set_color_depth(32);

// SETTA LA MODALITA' GRAFICA COME AUTOMATICA SU UNA FINESTRA DI 640x480
PIXEL //
set_gfx_mode(GFX_GDI, 640, 480, 0, 0);

// VISUALIZZA LA FINESTRA DI DIALOGO CREATA
do_dialog(finestra_dialogo, -1);
}
END_OF_MAIN();
```

Analizziamo le parti salienti del programma MINIGUIFINAL.C:
Questo listato completa l'argomento GUI con allegro.h dando all'utente il controllo più che ottimale delle interfacce. Costruire un menù a tendina non è mai stato così facile... In questa parte finale sul GUI si analizza come personalizzare menù e infiniti sotto menù generando qualsiasi tipo di evento sia con il mouse che con la pressione di tasti.

21 - I bitmap diventano trasparenti.




```
/* LISTATO C/C++ DI TRASPARENZA.C
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
*/

#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA
#include "allegro.h" /* INCLUSA LA LIBRERIA ALLEGRO.H */
BITMAP *sprite, *background, *buffer; // DICHIARAZIONE VARIABILI //
PALLETE colori; // DICHIARAZIONE VARIABILI //
int x; // DICHIARAZIONE VARIABILI //
int y; // DICHIARAZIONE VARIABILI //

void doppiobuffering() // INIZIO PROCEDURA DI DOPPIO BUFFERING //
{
vsync(); // SINCRONIZZAZIONE AGGIORNAMENTO SCHERMO //
blit(buffer, screen, 0, 0, 0, 0, 640, 480); // VISUALIZZA IL CONTENUTO DI BUFFER SU SCHERMO
```

```
//
clear(buffer); // PULISCE IL BUFFER //
}

int main() // INIZIO PROCEDURA DI ALLEGRO //
{

allegro_init(); // INIZIALIZZA ALLEGRO //
install_keyboard(); // INSTALLA LA TASTIERA //

set_color_depth(32); // SELEZIONA LA PROFONDITA' DI COLORE A 32 BIT //
set_gfx_mode(GFX_GDI, 640, 480, 0, 0); // SETTA LA MODALITA' GRAFICA COME GDI SU UNA FINESTRA DI 640x480 PIXEL //
sprite = load_bitmap("sprite.bmp",colori); // CARICA IL FILE SPRITE.BMP DOVE IL VIOLETTO IN MODALITA' TRUE COLOR E TRAPARENTE (ROSSO=255, VERDE=0, BLU=255) MENTRE PER LA MODALITA A 256 COLORI IL TRASPARENTE E IL NERO (R=0, V=0, B=0) //
background = load_bitmap("background.bmp",colori); // CARICA IL FILE BACKGROUND.BMP //
buffer = create_bitmap(640, 480); // CREA UN BUFFER BITMAP DI 640x480 PIXEL //
clear(buffer); // CANCELLA IL BUFFER //

x=140; // CENTRA LO STRITE //
y=350; // CENTRA LO SPRITE //

while (!key[KEY_SPACE]) { // CREA UN CICLO INFINO FINO ALLA PRESSIONE DEL TASTO DI SPAZIO //
blit(background, buffer, 0, 0, 0, 0, 640, 480); // VISUALIZZA L'IMMAGINE DI BACKGROUND SUL BUFFER //

set_trans_blender(0, 0, 0, 128); // DETERMINA: (ININFLUENTE, ININFLUENTE, ININFLUENTE, TRASPARENZA)
draw_trans_sprite(buffer, sprite, x, y); // VISUALIZZA LO SPRITE TRASPARENTE SUL BUFFER //
doppiobuffering(); // CARICA LA PROCEDURA DI DOPPIO BUFFERING //

if (key[KEY_UP]) y=y-3; // MOVIMENTO SPRITE IN ALTO ALLA PRESSIONE DEL TASTO CURSORE UP //
if (key[KEY_DOWN]) y=y+3; // MOVIMENTO SPRITE IN BASSO ALLA PRESSIONE DEL TASTO CURSORE DOWN //
if (key[KEY_LEFT]) x=x-3; // MOVIMENTO SPRITE A SINISTRA ALLA PRESSIONE DEL TASTO CURSORE LEFT //
if (key[KEY_RIGHT]) x=x+3; // MOVIMENTO SPRITE A DESTRA ALLA PRESSIONE DEL TASTO CURSORE RIGHT //
}

destroy_bitmap(sprite); // DISTRUGGE IL BITMAP //
destroy_bitmap(background); // DISTRUGGE IL BITMAP //
destroy_bitmap(buffer); // DISTRUGGE IL BITMAP //

allegro_exit(); // TERMINA LA LIBRERIA ALLEGRO //
}
END_OF_MAIN (); // FINE DEL PROGRAMMA //
```

Analizziamo le parti salienti del programma TRASPARENZA.C:
Il codice appena riportato è del tutto identico a quello di 'SPRITE.C', tranne per la sostituzione del comando 'draw_sprite(buffer, sprite, x, y);' con 'draw_trans_sprite(buffer, sprite, x, y);' e l'aggiunta di 'set_trans_blender(0, 0, 0, 128);' che insieme danno la possibilità di ottenere uno sprite trasparente. Per

variare la trasparenza è sufficiente modificare il valore 128 con un altro (0=trasparenza totale – 256 trasparenza nulla), ce da notare che oltre il valore 256 l'immagine subisce particolari variazioni di palette...

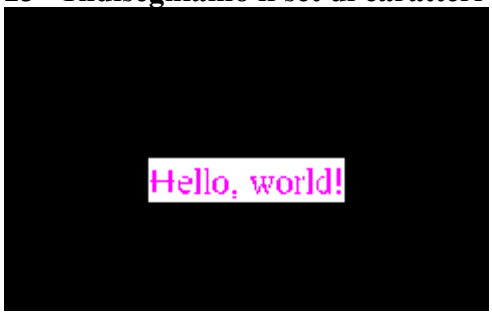
22 - Variazioni del canale ALPHA.


```
/* LISTATO C/C++ DI ALPHA.C
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
*/

#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA
#include "allegro.h" /* INCLUSA LA LIBRERIA ALLEGRO.H */
BITMAP *sprite, *background, *buffer; // DICHIARAZIONE VARIABILI //
PALLETE colori; // DICHIARAZIONE VARIABILI //
int x; // DICHIARAZIONE VARIABILI //
int y; // DICHIARAZIONE VARIABILI //

void doppiobuffering() // INIZIO PROCEDURA DI DOPPIO BUFFERING //
{
vsync(); // SINCRONIZZAZIONE AGGIORNAMENTO SCHERMO //
blit(buffer, screen, 0, 0, 0, 0, 640, 480); // VISUALIZZA IL CONTENUTO DI BUFFER SU SCHERMO
//
clear(buffer); // PULISCE IL BUFFER //
```

```
}

int main() // INIZIO PROCEDURA DI ALLEGRO //
{

allegro_init(); // INIZIALIZZA ALLEGRO //
install_keyboard(); // INSTALLA LA TASTIERA //

set_color_depth(32); // SELEZIONA LA PROFONDITA' DI COLORE A 32 BIT //
set_gfx_mode(GFX_GDI, 640, 480, 0, 0); // SETTA LA MODALITA' GRAFICA COME GDI SU UNA FINESTRA DI 640x480 PIXEL //
sprite = load_bitmap("sprite.bmp",colori); // CARICA IL FILE SPRITE.BMP DOVE IL VIOLETTO IN MODALITA' TRUE COLOR E TRAPARENTE (ROSSO=255, VERDE=0, BLU=255) MENTRE PER LA MODALITA A 256 COLORI IL TRASPARENTE E IL NERO (R=0, V=0, B=0) //
background = load_bitmap("background.bmp",colori); // CARICA IL FILE BACKGROUND.BMP //
buffer = create_bitmap(640, 480); // CREA UN BUFFER BITMAP DI 640x480 PIXEL //
clear(buffer); // CANCELLA IL BUFFER //

x=140; // CENTRA LO STRITE //
y=350; // CENTRA LO SPRITE //

while (!key[KEY_SPACE]) { // CREA UN CICLO INFINO FINO ALLA PRESSIONE DEL TASTO DI SPAZIO //
blit(background, buffer, 0, 0, 0, 0, 640, 480); // VISUALIZZA L'IMMAGINE DI BACKGROUND SUL BUFFER //

set_trans_blender(0, 0, 0, 0); // DETERMINA: (ROSSO, VERDE, BLU, ININFLUENTE)
draw_lit_sprite(buffer, sprite, x, y, 192); // VISUALIZZA LO SPRITE CON VARIAZIONE ALPHA SUL BUFFER //
doppiobuffering(); // CARICA LA PROCEDURA DI DOPPIO BUFFERING //

if (key[KEY_UP]) y=y-3; // MOVIMENTO SPRITE IN ALTO ALLA PRESSIONE DEL TASTO CURSORE UP //
if (key[KEY_DOWN]) y=y+3; // MOVIMENTO SPRITE IN BASSO ALLA PRESSIONE DEL TASTO CURSORE DOWN //
if (key[KEY_LEFT]) x=x-3; // MOVIMENTO SPRITE A SINISTRA ALLA PRESSIONE DEL TASTO CURSORE LEFT //
if (key[KEY_RIGHT]) x=x+3; // MOVIMENTO SPRITE A DESTRA ALLA PRESSIONE DEL TASTO CURSORE RIGHT //
}

destroy_bitmap(sprite); // DISTRUGGE IL BITMAP //
destroy_bitmap(background); // DISTRUGGE IL BITMAP //
destroy_bitmap(buffer); // DISTRUGGE IL BITMAP //

allegro_exit(); // TERMINA LA LIBRERIA ALLEGRO //
}
END_OF_MAIN (); // FINE DEL PROGRAMMA //
```

Analizziamo le parti salienti del programma ALPHA.C:
Il listato di alpha.c è identico a trasparenza.c con l'unica variante del comando 'draw_lit_sprite(buffer, sprite, x, y, 192);' che determina appunto la variazione alpha della palette colore. Gli indici da 0 a 255 creano un spostamento del colore verso il nero, mentre da 256 a 511 creano varie tonalità sull'immagine negativa e poi continuano i valori... In questo contesto il comando 'set_trans_blender(0, 0, 0, 0);' può agire sui valori RGB ma non sulla trasparenza.

23 - Ridisegniamo il set di caratteri del BIOS.

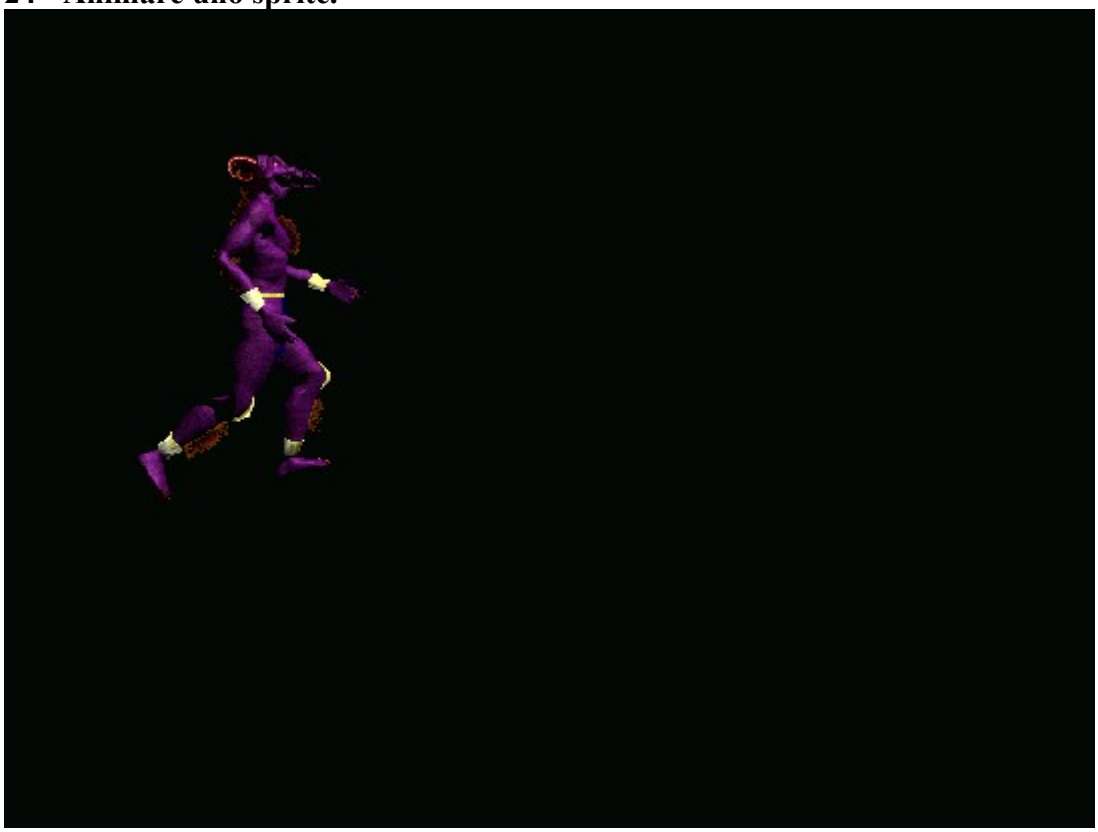



```
/* LISTATO C/C++ DI FONT.C
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
*/

// Carica la libreria allegro.h
#include <allegro.h>

// Assegno la variabile puntatore per i caratteri
FONT *font;

// Procedura principale
main()
{

// Inizializzo la libreria
allegro_init();
install_keyboard();

// Determino la modalità grafica
set_gfx_mode(GFX_GDI, 320, 200, 0, 0);

// Carico il file bitmap contenente il set di caratteri ridisegnato
font = load_font("font.pcx", NULL, NULL);

// Scrivo qualcosa su schermo
textout_centre_ex(screen, font, "Hello, world!", SCREEN_W/2, SCREEN_H/2,
makecol(255,0,255), makecol(255,255,255));

// Attende la pressione di un tasto
readkey();

// Elimino dalla memoria il set di caratteri
destroy_font(font);
}

END_OF_MAIN();
```

Analizziamo le parti salienti del programma FONT.C:
Molto sovente il set di caratteri fornito dal bios è un po' scarso e delude le aspettative se viene inserito in un videogames... Per questo motivo allegro.h ci da la possibilità di variare a nostro piacimento il set di

caratteri. L'operazione è estremamente semplice, visto che è sufficiente ridisegnare il set di caratteri contenuto in un bitmap a 256 colori (es. PCX). Attenzione però a seguire lo schema dell'immagine contenuto in questo esempio! Infatti il carattere modificato deve seguire un ordine preciso e inoltre deve essere contenuto in un rettangolo nero. Nell'esempio sopra esposto non è indicata ne la palette ne la profondità di colore perché il set di caratteri manipolato è monocromatico per semplicità didattica. Ora non vi resta che divertirvi a ridisegnare i caratteri del PC come più vi piace.

24 - Animare uno sprite.



```
/* LISTATO C/C++ DI REMAKE.C
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
HOME: HTTP://WWW.BERTINETTOBARTOLOMEODAVIDE.IT
E-MAIL: contatto@bertinettobartolomeodavide.it
*/

#include "allegro.h" /* CARICA LA LIBRERIA ALLEGRO.H */
#include "bestia.h" /* CARICA LA LIBRERIA CHE ABBIAMO CREATO CON IL FILE *.DAT */

DATAFILE *dat; // DICHIARAZIONE VARIABILE DEL FILE DAT //
BITMAP *buffer;
int beastcorsaani, beastflagcorsa; // INIZIALIZZO LE VARIABILI DEL GIOCATORE //
void doppiobuffering() // INIZIO PROCEDURA DI DOPPIO BUFFERING //
{
vsync(); // SINCRONIZZAZIONE AGGIORNAMENTO SCHERMO //
blit(buffer, screen, 0, 0, 0, 0, 640, 480); // VISUALIZZA IL CONTENUTO DI BUFFER SU SCHERMO
//
clear(buffer); // PULISCE IL BUFFER //
}
```

```
void personaggio() { // CREO LA PROCEDURA DEL GIOCATORE
if (key[KEY_RIGHT]) { // SE VIENE PREMUTO IL TASTO 'CORSORE DESTRO ALLORA...
if (beastflagcorsa==0) { // CONTROLLA LA VARIABILE 'BEASTFLAGCORSA' SIA UGUALE A '0' ALLORA...
draw_sprite(buffer, dat[beastcorsaani].dat,50,50); // DISEGNA LO SPRITE SUL BUFFER AD UN DETEMINATO FOTOGRAMMA
beastcorsaani++; // CONTATORE DEI FOTOGRAMMI PER LA CORSA DEL GIOCATORE
if (beastcorsaani>=19) { // SE 'BEASTCORSAANI' E' UGUALE O SUPERIORE A 19 ALLORA...
beastflagcorsa=1; // 'BEASTFLAGCORSA' DIVENTA UGUALE AD 1
}
}
if (beastflagcorsa==1) { // SE 'BEASTFLAGCORSA' E' UGUALE AD 1 ALLORA...
draw_sprite(buffer, dat[beastcorsaani].dat,50,50); // DISEGNA LO SPRITE SUL BUFFER AD UN DETEMINATO FOTOGRAMMA
beastcorsaani--; // CONTATORE DEI FOTOGRAMMI PER LA CORSA DEL GIOCATORE
if (beastcorsaani<=0) { // SE 'BEASTCORSAANI' E' UGUALE O MIINORE A 0 ALLORA...
beastflagcorsa=0; // 'BEASTFLAGCORSA' DIVENTA UGUALE A 0 E L'ANIMAZIONE RIPARTE DA CAPO
}
}
}
}

int main() /* INIZIO PROCEDURA */
{

PALETTE colori; /* DICHIARAZIONE VARIABILE COLORI */

allegro_init(); /* INIZIALIZZA LA LIBRERIA ALLEGRO */
install_keyboard(); /* INSTALLA LA TASTIERA */
set_color_depth(32); /* IMPOSTA LA PALETTE DEI COLORI A 32 BIT */
set_palette(colori);
set_gfx_mode(GFX_GDI, 640, 480, 0, 0); /* IMPOSTA LA MODALITA' GRAFICA 640X480 */
dat = load_datafile("bestia.dat"); // CARICA IL FILE DAT DEL GIOCATORE//
buffer = create_bitmap(640, 480); // DETERMINA LA GRANDEZZA DEL BUFFER IL PIXEL CHE SARA' POI VISUALIZZATO SU SCHERMO
beastcorsaani=0; // INIZIA A CONTARE I FOTOGRAMMI PARTENDO DA 0
beastflagcorsa=0; // FA PARTIRE CON 0 LA VARIABILE 'BEASTCORSAANI'
while (!key[KEY_ESC]) { // CREA UN CICLO INFINITO FINO ALLA PRESSIONE DEL TASTO 'ESC'
doppiobuffering(); // RICHIAMA LA PROCEDURA DEL DOPPIO BUFFERING
personaggio(); // RICHIAMA LA PROCEDURA DEL PERSONAGGIO PRINCIPALE
}
allegro_exit(); /* TERMINA LA LIBRERIA ALLEGRO */
}
END_OF_MAIN (); /* FINE DEL PROGRAMMA */
```

Analizziamo le parti salienti del programma REMAKE.C:
Animare uno sprite non è mai stato così facile se si utilizza un file .DAT. Infatti è sufficiente inserire tutti i fotogrammi di un personaggio all'interno di questo file e poi farli scorre uno alla volta nei vostri eseguibili, per incanto i protagonisti dei vostri videogiochi prenderanno vita. E' possibile creare un file .dat per ogni azione se i fotogrammi sono molti. Altrimenti può bastare un solo file .dat per ogni singolo personaggio contenente tutti i frames per ogni azione... Questa scelta è solo vostra!

25 - Creare un archivio compresso LZSS ed estrarlo.

- Programma che comprime:

```
// Programma comprime.c
// Realizzato da Bartolomeo Davide Bertinetto
// www.bertinettobartolomeodavide
// contatto@bertinettobartolomeodavide.it


#include "allegro.h"

int main()
{
// Puntatori che indicano il tipo di operazione da effettuare
char *m1, *m2;

// Variabili che determinano la dimensione del file
long s1, s2;

// Puntatori che contengono il file in entrata ed in uscita
PACKFILE *in, *out;

// Variabile alla quale è assegnato il file sotto forma di numeri interi
int c;

// Ordina al computer di leggere il file decompresso
m1 = F_READ;

// Ordina al computer di scrivere un file compresso
m2 = F_WRITE_PACKED;

// Determina la dimensione del file da leggere
s1 = file_size("testo.txt");

// Decide che il file "testo.txt" è quello da leggere
in = pack_fopen("testo.txt", m1);

// Si indica che "testo.lzs" è il file di destinazione compresso
out = pack_fopen("testo.lzs", m2);

// ciclo nel quale viene trasferito il contenuto del file letto a quello creato e compresso
// Il ciclo va avanti fin tanto che non viene incontro il carattere 'EOF' cioè terminatore del file
while ((c = pack_getc(in)) != EOF) {
if (pack_putc(c, out) != c)
break;
}

// Chiude il file letto
pack_fclose(in);

// chiude il file scritto
pack_fclose(out);
```

```
// Determina la fimensione dell'archivio creato
s2 = file_size("testo.lzs");

// Visualizza il messaggio su schermo indicante i risultati
printf("\nDimensione file letto: %ld\nDimensione file scritto: %ld\n%ld%%\n", s1, s2, (s2*100+(s1>>1))/
s1);
}
END_OF_MAIN();
```

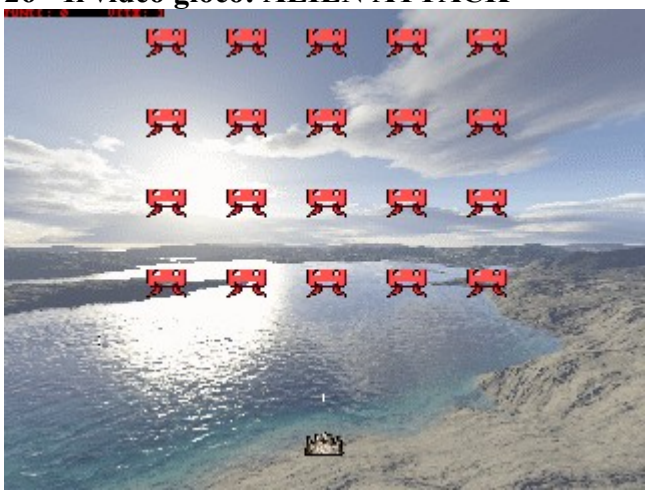
- Programma che decomprime:

```
// Programma decomprime.c
// Realizzato da Bartolomeo Davide Bertinetto
// www.bertinettobartolomeodavide
// contatto@bertinettobartolomeodavide.it

#include "allegro.h"

int main()
{
// Puntatori che indicano il tipo di operazione da effettuare
char *m1, *m2;

// Variabili che determinano la dimensione del file
long s1, s2;

// Puntatori che contengono il file in entrata ed in uscita
PACKFILE *in, *out;

// Variabile alla quale è assegnato il file sotto forma di numeri interi
int c;

// Ordina al computer di leggere il file compresso
m1 = F_READ_PACKED;

// Ordina al computer di scrivere un file decompresso
m2 = F_WRITE;

// Determina la dimensione del file da leggere
s1 = file_size("testo.lzs");

// Decide che il file "testo.lzs" è quello da leggere compresso
in = pack_fopen("testo.lzs", m1);
// Si indica che "testo2.txt" è il file di destinazione decompresso
out = pack_fopen("testo2.txt", m2);

// ciclo nel quale viene trasferito il contenuto del file letto compresso a quello creato
// Il ciclo va avanti fin tanto che non viene incontro il carattere 'EOF' cioè terminatore del file
while ((c = pack_getc(in)) != EOF) {
if (pack_putc(c, out) != c)
break;
}
```

```
// Chiude il file letto
pack_fclose(in);

// chiude il file scritto
pack_fclose(out);

// Determina la fimensione del file estratto
s2 = file_size("testo2.txt");

// Visualizza il messaggio su schermo indicante i risultati
printf("\nDimensione file letto: %ld\nDimensione file scritto: %ld\n%ld%%\n", s1, s2, (s2*100+(s1>>1))/
s1);
}
END_OF_MAIN();
```

Analizziamo le parti salienti del programma COMPRIME.C e DECOMPRIME.C:
Questo sistema di compressione non è di certo efficace come lo ZIP od altri... In ogni caso è molto utile ed estremamente veloce. Vi tornerà sicuramente utile nell'archiviare i contenuti più diversi. Consiglio di esaminare il formato LZSS su GOOGLE.

26 - Il video gioco: ALIEN ATTACK



```
// ALIEN ATTACK - FREEWARE 2006
// DI BARTOLOMEO DAVIDE BERTINETTO
// www.bertinettobartolomeodavide.it
// ALLEGATO AL LIBRO
// REALIZZATO GRAZIE ALL'UTILIZZO DELLA LIBRERIA ALLEGRO.H


#include "allegro.h" // carica la libreria allegro.h

BITMAP *sprite, *buf, *fuoco, *nemico[25], *fnemico, *esplosione, *logo; // dichiarazione variabili
bitmap
BITMAP *sfondo1, *sfondo2, *sfondo3, *sfondo4, *sfondo5; // dichiarazione variabili bitmap
PALLETE co; // dichiarazione variabile palette
int x, y; // coordinate giocatore //
int fx, fy, fok; // coordinate e variabili fuoco giocatore //
int ex, ey, ritespl, esplode; // ex e ey=pos. esplosione player, ritespl=tempo di esplosione, explode=start
per l'esplosione //
int nx[25], ny[25], na[25], cattivovince, ritardogameover, vel; // coordinate e variabili nemico in array
```

```
fino a 25, incremento velocita' //
int fnr[25], fn[25], fnx[25], fny[25], fnstop; // fnr[a]= numero casuale, fn[a]= permette lo sparo del proiettile, fnstop[a]= fa si che non vengano sparati altri proiettili, fny[a] e fny[a]= posizione proiettile //
int espx[25], espy[25]; // coordinate in array per esplosione //
int a, b, c, e[25], f[25], g[25]; // variabili di array e posizione di partenza nemici //
int d=25, contapunti, meno, vite=3, ini; // numero di nemici - dimensione max di array - variabile conta punti e contatore di esaurimento nemici, vite giocatore settato a 3, variabile di inizio //
char score[80]; // crea la dimensione max in cifre per il punteggio //
int ritardo; // variabile temporizzatrice //
int fondale; // variabile per il conteggio di fondale
SAMPLE *suono, *gameov, *vitameno; // variabili per il sonoro //
MIDI *music; // dichiarazione variabile per la musica

// inizio con la pressione ri 'RETURN'
void inizio(void) {
if (ini==0) { // variabile di inizio o di gioco: 0=inizio - 1=inizio //
fondale=0; // se è 0 non carica il fondale

// logo di presentazione //
blit (logo, buf, 0, 0, 0, 0, 640,480); // visualizza l'immagine //

// alcuni testi che sono visualizzati in sovrapposizione con la schermata di presentazione
textout(buf, font, "www.bertinettobartolomeodavide.it", 200, 20, 255);
textout(buf, font, "SVILUPPATO DA BARTOLOMEO DAVIDE BERTINETTO", 150, 400, 255);
textout(buf, font, "ALLEGRO.H 2006 FREEWARE", 110, 470, 255);
textout(buf, font, "PREMI ENTER PER INIZIARE", 220, 250, 215); // testo che indica la pressione del tasto, la posiozne x e y, il colore
if (key[KEY_ENTER]) {
vite=3; // assegna il numero di vite al giocatore
ini=1; // se è uguale a 1 dice al programma di iniziare il gioco
meno=0; // se è 0 significa che in questa parte del gioco sono presenti zero nemici
}
}
}

// aggiornamento dello schermo in doppio buffering //
void vai(void)
{
vsync(); // sicronizza la velocità di aggiornamento dello schermo
blit(buf, screen, 0, 0, 0, 0, 640, 480); // Trasferisce il contenuto del buffer su schermo
clear(buf); // Pulisce il buffer
}

//Il gioco inizia dal principio
void partenzadazero(void) {
// valori delle variabili di partenza //
if (ini==1) { // variabile di inizio o di gioco: 0=inizio - 1=gioco //
if (meno==0) { // vengono assegnati i seguenti valori solo se meno e uguale a zero: cioe' all'inizio e quando vengono distrutti tutti i nemici, cosi si ricomincia a giocare //
meno=20; // inizierà a contare da 20
d=20; // determina il numero di nemici su schermo
fx=x; // posizione del fuoco, che all'inizio coincide con quella del giocatore
fy=y; // posizione del fuoco, che all'inizio coincide con quella del giocatore
x=300; // posizione iniziale del giocatore
y=420; // posizione iniziale del giocatore
fok=0; // se 0 il proiettile non parte se 1 parte
```

```
a=0, b=0, c=20; // variabili dell'array

for (a=0; a<=d; a++) { // ciclo che crea un array di nuove variabili per la posizione di partenza di ogni nemico //
if (a>1) { // posizione x nell'array
b=b+80; // sposta la posizione di ogni nuovo nemico per ogni arry del ciclo
}
if (b>=400) { // posizione y dell'array
b=0; // mette b a 0
c=c+80; // sposta la posizione di ogni nuovo nemico per ogni array del ciclo
}

na[a]=1; // array del fuoco nemico
ny[a]=c; // posizione di un proiettile nemico nell'array
nx[a]=b; // posizione di un proiettile nemico nell'array
e[a]=0; f[a]=0; g[a]=0; // posizioni di partenza
espx[a]=0; espy[a]=0; // posizioni di partenza delle esplosioni
vel=1; // velocità di spostamento dei nemici
}
}
}
}

// tasti di spostamento del giocatore sinistra destra e visualizzazione giocatore //
void tasti(void)
{
if (ini==1) { // variabile di inizio gioco: 0=inizio - 1=gioco //
draw_sprite(buf, sprite, x, y); // disegna il giocatore //
if (key[KEY_LEFT]) x=x-2; if (x<=0) x=0; // sposta a sx //
if (key[KEY_RIGHT]) x=x+2; if (x>=600) x=600; // sposta a dx //
}
}

// Fuoco giocatore //
void fire(void)
{
if (ini==1) { // variabile di inizio o di gioco: 0=inizio - 1=gioco //
if (fok==0) { // se fok e' uguale a 0 allora permette di premere il tasto di fuoco //
if (key[KEY_SPACE]) { // se viene premuto lo spazio //
fok=1; // fok diventa 1 non permettendo di ripetere questa routine fino a quando non si sarà compiuto l'evento //
play_sample(suono, 255,128,1000, FALSE); // emette un suono //
}
}
if (fok==1) { // se fok e' uguale a 1 //
fy=fy-3; // fy decrementa di 3 facendo avanzare verso l'alto il proiettile //
draw_sprite(buf, fuoco, 18+fx, fy); // disegna il proiettile, il +18 serve per centrare il proiettile sul giocatore //
if (fy==0) { // se fx(il proiettile) va a 0(a bordo schermo in alto) //
fok=0; // allora fok ritorna a 0 permettendo di sparare un altro proiettile
fy=y; fx=x; fok=0; } } // tutte le variabili di posizione ritornano uguali a quelle del giocatore
if (fok==0) { // se fok e' uguale a zero //
fx=x; fy=y; // le variabili di partenza del proiettile sono uguali a quelle del giocatore //
}
}
}
```

```
// visualizzazione e movimento nemici //
void cattivo(void)
{
if (ini==1) { // variabile di inizio o di gioco: 0=inizio - 1=gioco //
for (a=0; a<=d; a++) { // ciclo che crea un array di nuove variabili //
if (e[a]==0) { // se e[a] e' diverso da 0 significa che il nemico e' stato colpito e quindi non viene
visualizzato //
draw_sprite(buf, nemico[a], nx[a], ny[a]); // disegna il nemico //

if (na[a]==1) { // se na[a] e' ugale a 1 //
nx[a]=nx[a]+vel; // nx[a] incrementa di 1 e il nemico avanza a dx //
if (nx[a]>=600) { // se nx supera 600 ny[a] incrementa di 40 e il nemico scende di una riga.
ny[a]=ny[a]+40; na[a]=2; } // na[a] diventa uguale a 2 //
}
if (na[a]==2) { // se na[a] e' uguale a 2 //
nx[a]=nx[a]-vel; // allora nx[a] decrementa di 1 e il nemico avanza verso sinistra //
if (nx[a]<=0) { // se nx[a] e minore o uguale a 0 //
ny[a]=ny[a]+40; na[a]=1; } // allora ny[a] incrementa di 40 e quindi il nemico scende di una riga. Cosi'
na[a] diventa uguale a 1 //
}
if (ny[a]>=400) { // se ny[a] e' uguale maggiore o uguale a 400 //
cattivovince=1; // IL NEMICO VINCE 'GAME OVER' PERCHE TOCCA IL TERRENO QUINDI
DIVENTA UGUALE A 1 //
play_sample(gameov, 255,128,1000, FALSE); // emette un suono //
}
} } } }

// collisione con alieno //
void collisione(void)
{
if (ini==1) { // variabile di inizio o di gioco: 0=inizio - 1=gioco //
for (a=0; a<=d; a++) { // ciclo che crea un array di nuove variabili //
if (e[a]==0) { // se e[a] e' uguale a 0 allora esegue //
if ((fy<=(ny[a]+14)) && (fy>=(ny[a]-14))) { // se fy(fuoco giocatore) e' minore o uguale a ny[a]
(posizione nemico) piu' 14(altezza nemico) e fy e' maggiore o uguale a ny[a] meno 14 allora //
if ((fx>=(nx[a]-20)) && (fx<=(nx[a]+20))) { // se fx(fuoco giocatore) e' maggiore o uguale a nx[a]
(posizione nemico) meno 20(larghezza nemico) e fx e' minore o uguale a nx[a] piu' 20 allora //
fok=0; e[a]=1; fx=x; fy=y; // variabili fuoco giocatore a zero ed e[a] diventa ugale a 1 cosi' non ci sara'
un'altra collsione con il nemico gia' esploso //
contapunti++; // incrementa il punteggio di 1 //
meno--; // conta quanti nemici vengono distrutti //
play_sample(vitameno, 255,128,1000, FALSE); // emette un suono //
}
}
} } } }

// esplosione alieno //
void explode(void)
{
if (ini==1) { // variabile di inizio o di gioco: 0=inizio - 1=gioco //
for (a=0; a<=d; a++) { // ciclo array esplosione //
if (e[a]==1) { // se e[a] e' a 1 allora significa che il nemico e stato colpito e deve esplodere //
if (f[a]==0) { // se f[a] e' uguale a 0 vuol dire che che il nemico non e' ancora esploso //
espx[a]=nx[a]; espy[a]=ny[a]; // l'esplosione avviene nella stessa posizione dell'alieno colpito cioe'
espx[a](posiozione esplosione) e' uguale a nx[a] (posizione alieno) e espy[a](posiozione esplosione) e'
```

```
uguale a ny[a](posizione alieno) //
f[a]=1; // se f[a] e' uguale a 1 l'esplosione di questo alieno non averra' piu' //
nx[a]=0; ny[a]=0; // mette a zero la posiozione dell'alieno esploso //
}
if (f[a]==1) { // se f[a] e' uguale a 1 parte la sequenza dell' esplosione //
g[a]=g[a]+1; // contatore di ritardo per il tempo di permanenza dell' esplosione //
draw_sprite(buf, esplosione, espx[a], espy[a]); // visualizza l'esplosione //
if (g[a]>=50) { // se g[a] e' maggiore o uguale a 50 significa che e' stato superato il tempo limite di
visualizzazione dell' esplosione //
f[a]=2; e[a]=2; // f[a] e e[a] sono impostati a 2 non verra colpito un alieno gia' colpito e non esplodera un
alieno gia' esploso //
}
}
} }
} }

// indicatore punteggio //
void puntivite(void) {
if (ini==1) { // variabile di inizio o di gioco: 0=inizio - 1=gioco //
sprintf(score, "PUNTI: %ld VITE: %ld", (long)contapunti, (long)vite); // da il valore a score e a vite da
visualizzare //
textout(buf, font, score, 1, 1, 215); // inserisce nel doppio buffer dello schermo il valore di score e vite che
verra poi visualizzato //
}
}

// routine di incremento velocita' del nemico quando diminuisce il numero //
void incrementovelocitanemico(void) {
if (ini==1) {
if ((meno<=8) && (meno>=4)) { // se ci sono tra 4 e 8 nemici allora //
vel=2; // la velocita' del nemico aumenta di 1 //
}
if ((meno<=3) && (meno>=2)) { // se ci sono tra 3 e 2 nemici allora //
vel=3; // la velocita' del nemico aumenta di 2 //
}
if (meno==1) { // se c'è un solo nemico allora //
vel=4; // la velocita' del nemico aumenta di 3 //
}
}
}

// game over quando il cattivo vince o il player viene colpito //
void gameover(void) {
if (ini==1) { // variabile di inizio o di gioco: 0=inizio - 1=gioco //
if (cattivovince==1) { // se il cattivo tocca terra cattivi vince diventa = a 1 allora //
textout (buf, font, "GAME OVER", 290, 200, 215); // VISUALLIZZA GAME OVER //
ritardogameover++; // INCREMENTA DI 1 IL CONTATORE PER LA VISUALIZZAZIONE DI
GAME OVER //
}
if (ritardogameover>=200) { // DETERMINA IL TEMPO DI VISUALIZZAZIONE DELLA SCRITT
GAME OVER SE E' UGUALE A 200 ALLORA //
cattivovince=0; // CATTIVO VINCE RITORNA AL VALORE DI PARTENZA //
meno=0; // MENO RITORNA AL VALORE DI PARTENZA E FA RICOMINCIARE IL GIOCO DA
CAPO //
vite=0; // LE VITE VENGONO PORTATE A ZERO //
contapunti=0; // I PUNTI VENGONO AZZERA IL PUNTEGGIO //
```

```
ritardogameover=0; // AZZERA IL CONTATORE DI PERMANENZA DELLA SCRITTA //
ini=0; // azzera ini per ricominciare da capo //
esplode=0; // azzera lo start dell'esplosione per eliminare lo scoppio //
ritespl=0; // azzera il contatore di permanenza dell'esplosione //
}
}
if (esplode==1) { // se e' uguale a 1 parte la routine dell'esplosione del player //
draw_sprite(buf, esplosione, ex, ey); // disegna l'esplosione //
ritespl++; // contatore di permanenza dell'esplosione //
if (ritespl>=100) { // il contatore e' uguale o maggiore di 100 allora //
esplode=0; // lo start dell esplosione ritorna a zero //
x=300; y=420; // le coordinate del player ritornano centrali //
ritespl=0; // azzera il contatore dell'esplosione //
}
}
}

// routine di sparo del nemico //
void sparonemico(void) {
if (ini==1) {
for (a=0; a<=d; a++) { // se e' a 1 esegue la routine di sparo nemico //
if (e[a]==0) { // se e' a 0 significa che il nemico non e' stato colpito //
if (fnstop==0) { // Controlla che i nemici non sparino tutti contemporaneamente //
if ((fnr[a]>=20000) && (fnr[a]<=21000)) { // Fa si che avvenga lo sparo solo se fnr e' tra 20000 e
21000 //
play_sample(suono, 255,128,1000, FALSE); // emette un suono //
fn[a]=1; fnx[a]=nx[a]; fny[a]=ny[a]; fnstop=1; // Setta tutte le variabili per la partenza del proiettile //
}
fnr[a]=rand(); // fnr assume un numero casuale //
}
}
if (fn[a]==1) { // se fn[a] e' uguale a 1 il proiettile parte //
draw_sprite(buf, fnemico, fnx[a]+10, fny[a]+7); // disegna il proiettile centrato sul nemico //
fny[a]++; // incrementa fny di 1 facendo scorrere il proiettile verso il basso //
if (fny[a]>=480) { // se fny(il proiettile) e' uguale o maggiore di 480 allora //
fny[a]=ny[a]; // fny[a] riprende il valore di ny[a] //
fnx[a]=nx[a]; // fnx[a] riprende il valore di nx[a] //
fnr[a]=0; // il numero casuale viene riportato a 0 //
fn[a]=0; // fn[a] viene azzerato cosi che il proiettile non si muova e non venga disegnato //
fnstop=0; // fnstop ritorna a 0 cosiche' possano partire altri proiettili //
}
}
}
}
}

// collisione del proiettile nemico con il player //
void collisioneconproiettilenemico(void) {
if (ini==1) { // se e' uguale a 1 allora esegue la routine //
for (a=0; a<=d; a++) { // crea l'array //
if (fn[a]==1) { // se e' uguale a 1 significa che il proiettile e' stato sparato //
if ((fny[a]>=y-28) && (fny[a]<=y)) { // punto di collisione con il player y //
if ((fnx[a]>=x-20) && (fnx[a]<=x+20)) { // punto di collisione con il player x //
fnx[a]=nx[a]; fny[a]=ny[a]; // se il player viene colpito il proiettile viene trasferito nella posizione di
partenza //
vite--; // decremento di una vita //
```

```
esplode=1; // se e uguale a 1 fa partire l'esplosione del player //
ex=x; ey=y; // le variabili dell'esplosione diventano uguali a quelle del player //
x=-1000; y=-1000; // il player scompare fuori dallo schermo //
play_sample(vitameno, 255,128,1000, FALSE); // emette un suono //
if (vite==0) { // se le vite sono uguali a 0 allora //
cattivovince=1; // parte la routine di game over //
play_sample(gameov, 255,128,1000, FALSE); // emette un suono //
}
}}}}}}

// routine dei fondali
void sfondo(void) {
if (ini==1) { // se 1 da il via alla routine
if (meno==0) fondale++; // se ci sono 0 nemici su schermo cambia fondale facendo avanzare il contatore
if (fondale>5) fondale=1; // se il contatore è superiore a 5 riparte da 1
if (fondale==1) draw_sprite (buf, sfondo1, 0, 0); // visualizza l'immagine sfondo //
if (fondale==2) draw_sprite (buf, sfondo2, 0, 0); // visualizza l'immagine sfondo //
if (fondale==3) draw_sprite (buf, sfondo3, 0, 0); // visualizza l'immagine sfondo //
if (fondale==4) draw_sprite (buf, sfondo4, 0, 0); // visualizza l'immagine sfondo //
if (fondale==5) draw_sprite (buf, sfondo5, 0, 0); // visualizza l'immagine sfondo //
}
}

// inizializzazione della libreria allegro //
int main()
{

allegro_init();
install_keyboard();

suono = load_sample("fuoco.wav"); // carica il file wav //
gameov = load_sample("gameover.wav"); // carica il file wav //
vitameno = load_sample("vitameno.wav"); // carica il file wav //
music = load_midi("albmsont.mid"); // carica il file mid
install_sound(DIGI_AUTODETECT, MIDI_AUTODETECT, 0); // inizializza la scheda audio //
set_volume(255,255); // setta i parametri di volume //
play_midi(music, TRUE); // da il via ad un ciclo musicale infinito

set_color_depth(32); // setta la palette a 32 bit
set_palette(co); // setta la palette per la variabile co
set_gfx_mode(GFX_GDI, 640, 480, 0, 0); // inizializza il formato grafico //

//carica le immagini del gioco
logo = load_bmp("presentazione0060.bmp",co); // carica l'immagine logo //
sfondo1 = load_bmp("background1.bmp",co); // carica l'immagine sfondo //
sfondo2 = load_bmp("background2.bmp",co); // carica l'immagine sfondo //
sfondo3 = load_bmp("background3.bmp",co); // carica l'immagine sfondo //
sfondo4 = load_bmp("background4.bmp",co); // carica l'immagine sfondo //
sfondo5 = load_bmp("background5.bmp",co); // carica l'immagine sfondo //

sprite = load_bmp("playerhi0001.bmp",co); // carica il giocatore //
fuoco = load_bmp("fuoco.bmp",co); // carica il fuoco //
fnemico = load_bmp("fuoco.bmp",co); // carica il fuoco nemico //
for (a=0; a<=d; a++) { // ciclo che crea un array di nuove variabili //
nemico[a] = load_bmp("mostroinvader0001.bmp",co); // carica l'immagine dei nemici //
}
```

```
esplosione = load_bmp("esplosione.bmp",co); // carica l'immagine dell'esplosione //
buf = create_bitmap(640, 480); // crea un una finestra di buffer 320 x 200 pixel //
clear(buf); // pulisce il buffer dello schermo //

// ciclo while che finisce con la pressione di ESC //
// all' interno di questo cicle sono racchiuse tutti gli oggetti costruiti in precedenza //
while (!key[KEY_ESC]) {
// successione delle routine create
inizio();
vai();
sfondo();
partenzadazero();
tasti();
fire();
cattivo();
collisione();
explode();
puntivite();
gameover();
incrementovelocitanemico();
sparonemico();
collisioneconproiettilenemico();
}

// vengono distrutti tutti i bitmap caricati prima della chiusura del programma //
destroy_bitmap(sprite);
destroy_bitmap(fuoco);
destroy_bitmap(fnemico);
// ciclo che crea un array di nuove variabili //
for (a=0; a<=d; a++) {
destroy_bitmap(nemico[a]);
}
destroy_bitmap(buf);
destroy_bitmap(logo);
destroy_bitmap(sfondo1);
destroy_bitmap(sfondo2);
destroy_bitmap(sfondo3);
destroy_bitmap(sfondo4);
destroy_bitmap(sfondo5);

// vengono distrutti tutti i sample e midi presenti in memoria
destroy_sample(vitameno);
destroy_sample(gameov);
destroy_sample(suono);
destroy_midi(music);
}
END_OF_MAIN (); // termina il programma
```

COMMENTO ALLE PRIME LINEE DI CODICE:
ALIEN ATTACK

Spiegherò il funzionamento di un video gioco classico vecchio stile, chiamato ALIEN ATTACK. Il listato che segue sarà commentato a dovere in ogni sua parte. Spiegando la funzione di ogni singola procedura.

```
#include "allegro.h"
```

```
BITMAP *sprite, *buf, *fuoco, *nemico[25], *fnemico, *esplosione, *logo;
BITMAP *sfondo1, *sfondo2, *sfondo3, *sfondo4, *sfondo5;
PALLETE co;
int x, y;
int ex, ey, ritespl, esplode;
int nx[25], ny[25], na[25], cattivovince, ritardogameover, vel;
int fnr[25], fn[25], fnx[25], fny[25], fnstop;
int espx[25], espy[25];
int a, b, c, e[25], f[25], g[25];
int d=25, contapunti, meno, vite=3, ini;
char score[80];
int ritardo;
int fondale;
SAMPLE *suono, *gameov, *vitameno;
MIDI *music;
```

Inizializzazione libreria 'allegro.h' necessaria alla compilazione del videogioco. Se non fosse inserita questa libreria, l'intero listato per il programma di compilazione non avrebbe senso!
Nella seconda parte vengono dichiarate le variabili necessarie al corretto funzionamento del programma, come: BITMAP, SUONI E MUSICHE, CREAZIONE DI ARRAY E VARIABILI CONDIZIONALI, ECC...

```
void inizio(void) {
if (ini==0) {
fondale=0;

blit (logo, buf, 0, 0, 0, 0, 640,480);

textout(buf, font, "www.bertinettobartolomeodavide.it", 200, 20, 255);
textout(buf, font, "SVILUPPATO DA BARTOLOMEO DAVIDE BERTINETTO", 150, 400, 255);
textout(buf, font, "ALLEGRO.H 2006 FREEWARE", 110, 470, 255);
textout(buf, font, "PREMI ENTER PER INIZIARE", 220, 250, 215);

if (key[KEY_ENTER]) {
vite=3;
ini=1;
meno=0;
}
}
}
```

La prima procedura è chiamata 'inizio' proprio perché si tratta del primo passo che deve compiere il video gioco: “visualizzare una schermata di presentazione”.
Il primo comando che segue 'void inizio(void) {' è l'istruzione 'if (ini==0) {', se 'ini' è uguale a '0' significa che il gioco non è ancora in funzione è deve essere visualizzata la schermata di presentazione.
Mentre 'fondale=0' ordina al computer di visualizzare come presentazione l'immagine numero '0'.
Il comando 'blit' visualizza l'immagine contenuta nel BITMAP 'logo' all'interno del buffer 'buf' dentro una finestra di 640X480 pixel.
Per visualizzare del testo in una schermata grafica si utilizza il comando 'textout'. Questa istruzione dice al computer di visualizzare una stringa di caratteri con un determinato colore ad una determinata posizione (x,y).
Dopo che la schermata è visualizzata con immagini e scritte, è il caso di porre una condizione di avanzamento, determinata in questo caso da: 'if (key[KEY_ENTER]) {'. Vale a dire che deve essere premuto il tasto 'enter' per fare proseguire il gioco. Quindi le variabili di vite saranno poste a 3 con

'vite=3', sarà determinato l'avvio di tutte le routine relative al gioco vero e proprio con il comando 'ini=1' e poniamo i nemici distrutti a zero con 'meno=0'.

```
void vai(void)
{
vsync();
blit(buf, screen, 0, 0, 0, 0, 640, 480);
clear(buf);
}
```

Con questa procedura 'void vai(void) {' costruviamo un algoritmo di doppio buffering. Ovvero prima di disegnare effettivamente il videogioco su schermo lo inseriamo su un buffer: nella variabile bitmap 'buf' per la precisione. Questo sistema evita il cosidetto sfarfallamento dello schermo, rendendo il video gioco molto fluido.
La voce 'vsync();' sicronizza la velocità di aggiornamento dello schermo rendendola costante. Mentre 'blit(buf, screen, 0, 0, 0, 0, 640, 480);' sposta il contenuto del buffer su schermo, visualizzando quello che è il video gioco vero e proprio. Infine 'clear(buf);' cancella dopo ogni ciclo il contenuto del buffer permettendo di disegnare elementi nuovi.

```
void partenzadazero(void) {

if (ini==1) {
if (meno==0) {
meno=20;
d=20;
fx=x;
fy=y;
x=300;
y=420;
fok=0;
a=0, b=0, c=20;

for (a=0; a<=d; a++) {
if (a>1) {
b=b+80;
}
if (b>=400) {
b=0;
c=c+80;
}

na[a]=1;
ny[a]=c;
nx[a]=b;
e[a]=0; f[a]=0; g[a]=0;
espx[a]=0; espy[a]=0;
vel=1;
}
}
}
}
```

Come possiamo capire dal nome della procedura 'void partenzadazero(void) {' in questa parte del programma sono inizializzati tutti i principali valori di partenza. La procedura partenzadazero si rende

attiva solo quando serve, cioè al primo avvio del video gioco oppure quando si ricomincia una nuova partita. L'attivazione è resa possibile portando 'ini=1' e il controllo delle vite del giocatore che devono essere uguali a zero 'vite=0'.
Qui viene creato un array di variabili che determinano la posizione di ogni nemico sull'asse x e y, oltre a quelli del fuoco ed esplosione.

```
void tasti(void)
{
if (ini==1) {
draw_sprite(buf, sprite, x, y);
if (key[KEY_LEFT]) x=x-2; if (x<=0) x=0;
if (key[KEY_RIGHT]) x=x+2; if (x>=600) x=600;
}
}
```

Per poter interagire in un video gioco è decessario predisporre un sistema di controllo del giocatore. In questo caso è stata scelta la tastiera.
Come al solito si supera il comando di controllo 'if' che deve essere uguale a '1' e si arriva così ai comandi veri e propri di questa routine.
Il primo della serie è 'draw_sprite(buf, sprite, x, y);' che serve per disegnare uno sprite su schermo(nel buffer in questo caso, dato che è utilizzata la tecnica del doppio buffering). Uno sprite indica un personaggio o una figura staccata dal fondale per la quale sono previste azioni interattive automatiche o impartite dall'utente. Il contenuto del bitmap, in questo caso la variabile 'sprite' è immagazzinata nel buffer, in questo caso 'buf', a delle determinate coordinate 'x,y'.
Seguono i comandi condizionali 'if' che controllano l'azione scelta dall'utente: pressione del tasto cursore di sinistra'[KEY_LEFT]' o di destra'[KEY_RIGHT]'. Per lo spostamento a sinistra viene diminuita 'x' di due unità(pixel) alla volta e per lo spostamento a destra è incrementata dello spesso valore.
Esitono due controlli di posizione per il giocatore principale, ovvero se si è raggiunto il bordo dello schermo(0 o 600). Nel caso si verifichi questa condizione il giocatore rimane fermo contro il bordo. Se non esistesse il controllo di bordo schermo lo sprite del giocatore finerebbe per 'sparire dietro lo schermo'.

```
void fire(void)
{
if (ini==1) {
if (fok==0) {
if (key[KEY_SPACE]) {
fok=1;
play_sample(suono, 255,128,1000, FALSE);
}
}
if (fok==1) {
fy=fy-3;
draw_sprite(buf, fuoco, 18+fx, fy);
if (fy==0) {
fok=0;
fy=y; fx=x; fok=0; } }
if (fok==0) {
fx=x; fy=y;
}

}
}
```

Lo spostamento del player principale a destra o a sinistra non determina una grossa capacità di interazione in un video game, è necessario qualcosa di più...

Inserie la possibilità di fare fuoco e quindi di struggere dei nemici rappresenta una capacità interattiva in un enorme quantità di generi di video giochi. Perciò la procedura che segue prevede proprio di inserire una serie di comandi che permettano di lanciare un proiettile alla volta con l'obbiettivo di colpire e ditruggere un personaggio nemico.
Dopo la solita variabile di controllo di inzio gioco 'ini=1' segue immediatamente un seccessivo controllo per la verifica se il tasto di fuoco(lo spazio) è già stato premuto oppure no 'fok=0' permettendo di premere il tasto che da il via alla partenza del proiettile.
Dopo il comando 'if (key[KEY_SPACE]) {' che controlla la pressione del tasto di spazio ponendo 'fok=1', dicendo al computer di non 'ascoltare' nuove pressioni di questo tasto fino a quando il proiettile sarà presente sullo schermo.
'play_sample(suono, 255,128,1000, FALSE);' permette l'emissione di un suono al manifestarsi dell'evento di fuoco presente nella varibile 'fuoco'.
Appena dopo che 'fok=1' inizia il conteggio dei pixel per lo spostamento del proiettile sull'asse 'y' decrementadolo di 3 pixel a ciclo. Quindi si disegna lo sprite contenuto nel bitmap 'fuoco' sul buffer 'buf' a delle precise coordinate. La coordinata 'x' è incrementata di 18 pixel per centrare il punto di partenza del proiettile con il centro del player. Non appena il missile raggiunge il bordo alto dello schermo lo sprite scompare 'if (fy==0) {' e 'fok=0'. Di conseguenza le coordinate del proiettile vengono nuovamente fatte coincidere con quello del giocatore per un a successiva partenza 'fy=y; fx=x; fok=0'.
A questo punto il computer 'obbedirà' ad una nuova pressione del tasto di spazio sparando nuovamente.

```
void cattivo(void)
{
if (ini==1) {
for (a=0; a<=d; a++) {
if (e[a]==0) {

draw_sprite(buf, nemico[a], nx[a], ny[a]);

if (na[a]==1) {
nx[a]=nx[a]+vel;
if (nx[a]>=600) {
ny[a]=ny[a]+40; na[a]=2; }
}
if (na[a]==2) {
nx[a]=nx[a]-vel;
if (nx[a]<=0) {
ny[a]=ny[a]+40; na[a]=1; }
}
if (ny[a]>=400) {
cattivovince=1;
play_sample(gameov, 255,128,1000, FALSE);
}
} } } }
```

Per assurdo se ci fosse un solo personaggio in un video gioco non ci sarebbero certamente molti stimoli nel giocarci! E' crea quindi la necessità di ovviare a questo problema visualizzando una serie di avversari, che magari, abbiamo la missione di ostacolarci...
All'attivazione del gioco viene creato un'array delle variabili(un espediente che moltiplica le variabili creando una griglia che può essere anche multidimensionale), con questa tecnica è possibile creare molti nemici con pochissimi comandi 'for (a=0; a<=d; a++) {'. Esiste un controllo che determina l'abbattimento di un nemico 'if (e[a]==0) {'. Inoltre i personaggi nemici devono essere visualizzati nel buffer con 'draw_sprite(buf, nemico[a], nx[a], ny[a]);', questa linea, come molte altre, è riciclata tante volte quanto è grande l'array da creare, esempio 'nemico[a]'.
Seguono due sotto routine che determinano lo spostamento automatico dei nemici 'if (na[a]==1) {'... e 'if

(na[a]==2) {'... Per ultima la condizione in cui i nemici arrivano a toccare il bordo inferiore dello schermo, vincendo la partita 'if (ny[a]>=400) {'...

```
void collisione(void)
{
if (ini==1) {
for (a=0; a<=d; a++) {
if (e[a]==0) {
if ((fy<=(ny[a]+14)) && (fy>=(ny[a]-14))) {
if ((fx>=(nx[a]-20)) && (fx<=(nx[a]+20))) {
fok=0; e[a]=1; fx=x; fy=y;
contapunti++;
meno--;
play_sample(vitameno, 255,128,1000, FALSE);
}
}

} } } }
```

Non è sufficiente disegnare personaggi su schermo e farli muovere! E' necessario dotare gli sprite di una sorta di componente fisica nello spazio. Bisogna perciò dotare i nemici, i proiettili e il giocatore di un area che se viene oltrepassata in qualche modo il computer lo interpreta come una collisione che genererà una serie di eventi...
Dopo aver avviato il gioco 'ini=1' sarà necessario creare un array per ogni nemico 'for (a=0; a<=d; a++) {'
Ora con una serie di comandi condizionali viene determinata la collisione tra il proiettile del giocatore e il nemico. Per fare questo è confrontata la dimensione e la posizione del nemico (dimensione nemico dal -14 a +14 sull'asse y e da -20 e +20 sull'asse x) con quella del missile. Quando la collisione si verifica 'fok=0' permette al giocatore di sparare un nuovo proiettile, mentre 'e[a]=1' disattiva il nemico distrutto evitando che sia colpito nuovamente. A questo punto le coordinate del missile saranno uguagliate a quelle del giocatore.
Ora il punteggio sarà incrementato di una unità e ridotto di una unità il numero di nemici. Per ultimo il computer emette un suono di distruzione 'play_sample(vitameno, 255,128,1000, FALSE)'.

Da questo punto in poi credo che abbiate raggiunto le basi per comprendere la costruzione di qualsiasi videogioco 2D... Ora è il momento di cambiare argomento e passare ai video games a tre dimensioni, grazie alle librerie OPENGL!!!

Grafica 3d con ALLEGROGL

- Guida all'uso attraverso listati C/C++ commentati -

Con questa pagina cerco di aprire le porte alla gestione della grafica OPENGL attraverso la libreria allegro(allegro.h) e allegrogl(alleggl.h).
Con questa parte attualizzo l'uso della libreria allegro.h... Infatti il mondo dei videogames è ormai passato alla grafica 3D e ben poche persone si sognerebbero di giocare per giornate intere a video giochi bidimensionali, come capitava ormai molti anni fa quando ero ragazzino io! Tutta l'argomentazione sulla creazione di videogiochi 3d possiede soltanto più finalità puramente didattiche che tornano utili per lo

sviluppo di ambienti tridimensionali. Questo è il futuro...

Esaminiamo così il primo listato in grado di fornire un risultato grafico tridimensionale senza oggetti.

Struttura di base vuota.

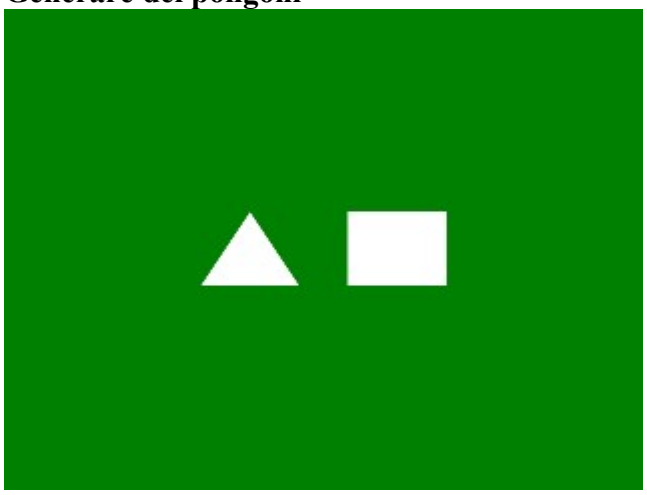
```
/* CODICE ESEMPLIFICATIVO SULLA GESTIONE DELLE OPENGL CON ALLEGRO
SCRITTO INTERAMENTE DA BARTOLOMEO DAVIDE BERTINETTO

WWW.BERTINETTOBARTOLOMEODAVIDE.IT

Listato: 3dn01vuoto.c

*/

#include <allegro.h>


#include "alleggl.h"
void camera (void);
void oggetto (void);


// procedura di creazione e posizionamento della camera nella scena e prospettiva
void camera (void)
{

// aggiunge la prospettiva alla scena
glMatrixMode (GL_PROJECTION);

// comando simile ad un reset di conferma
glLoadIdentity ();

// indica le proporzioni della finestra e dell'angolo prospettico
// inoltre l'ultimo valore indica la profondità massima di visione della camera.
// Tutti gli oggetti oltre questo valore non saranno visti nella scena.

glFrustum (-1.0, 1.0, -1.0, 1.0, 1.0, 40.0);

// indica alla scena la matrice di visualizzazione sulle caratteristiche
// del modello
glMatrixMode (GL_MODELVIEW);

// Cancella il buffer del colore RGB ed il depth buffer(pulisce la scena)
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// comando simile ad un reset di conferma
glLoadIdentity();

//posiziona la camera (x,y,z)
glTranslatef (0, 0, -0.5);
```

```
// orienta la camera nei rispettivi assi...
glRotatef (0, 1, 0, 0); //angolo x
glRotatef (0, 0, 1, 0); //angolo y
glRotatef (0, 0, 0, 1); //angolo z

// salva la posizione e rotazione della camera nella matrice scenica
glPushMatrix();
}

// posizione, orientamento, colore e creazione dell'oggetto

void oggetto (void)
{

/* Il comando 'glTranslatef(x,y,z)' sposta/posiziona un oggetto all'interno
della scena. Il centro della scena è il valore 0.0f in tutti gli assi.
Quindi per l'asse X si va a sinistra con i numeri negativi e a destra con quelli
positivi. Per l'asse Y si va in alto con i numeri positivi ed in basso con quelli
negativi. L'asse Z va all'interno dello schermo con i numeri negativi e
all'esterno con quelli positivi. */

glTranslatef(-1.5f,0.0f,-6.0f);

// PUNTO NEL QUALE SI INSERISCE IL CODICE DEI POLIGONI DA VISUALIZZARE
glFlush();
allegro_gl_flip();
}

int main ()
{
allegro_init();
install_allegro_gl();
allegro_gl_clear_settings();
allegro_gl_set (AGL_COLOR_DEPTH, 32);
allegro_gl_set (AGL_Z_DEPTH, 16);
allegro_gl_set (AGL_FULLSCREEN, TRUE);
allegro_gl_set (AGL_DOUBLEBUFFER, 1);
allegro_gl_set (AGL_SUGGEST, AGL_COLOR_DEPTH | AGL_Z_DEPTH | AGL_DOUBLEBUFFER
| AGL_FULLSCREEN);

// Attiva l'antialiasing (...,1). Disattiva l'antialiasing (...,0)
allegro_gl_set (AGL_RENDERMETHOD, 1);
// Profondità colore Palette
set_color_depth (32);

// Determina la modalità OPENGL e la risoluzione
set_gfx_mode(GFX_OPENGL, 640, 480, 0, 0);

install_keyboard();
install_timer();
allegro_gl_begin();

glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);
```

```
//Colore del fondale: (rosso,verde,blu,alfa) - max 1.0 min 0.0
glClearColor (0.0f, 0.5f, 0.0f, 0.5f);

//(GL_SMOOTH): Crea una migliore qualità di rendering(smussa i colori creando tonalità)
//es: (GL_FLAT): Il colore è il medesimo nella stessa faccia...
glShadeModel (GL_SMOOTH);

//INDICA AL COMPUTER CHE GLI OGGETTI SIANO DISEGNATI NEL GIUSTO ORDINE
glEnable (GL_DEPTH_TEST);

//attiva il miglior livello di correzione prospettica(calo velocità)
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

allegro_gl_end();

// Creo il ciclo
do {
camera();
oggetto();

} while (!key[KEY_ESC]);

}
END_OF_MAIN();
```

Commento al codice:
Il codice esposto getta le basi per creare un'infinità di programmi in grafica 3D aggiungendo semplicemente il codice necessario alle linee sopra esposte. Perciò consiglio di analizzarle attentamente!

Generare dei poligoni



```
/* CODICE ESEMPLIFICATIVO SULLA GESTIONE DELLE OPENGL CON ALLEGRO
SCRITTO INTERAMENTE DA BARTOLOMEO DAVIDE BERTINETTO

WWW.BERTINETTOBARTOLOMEODAVIDE.IT
Listato: 3dn02poligoni.c
*/
```

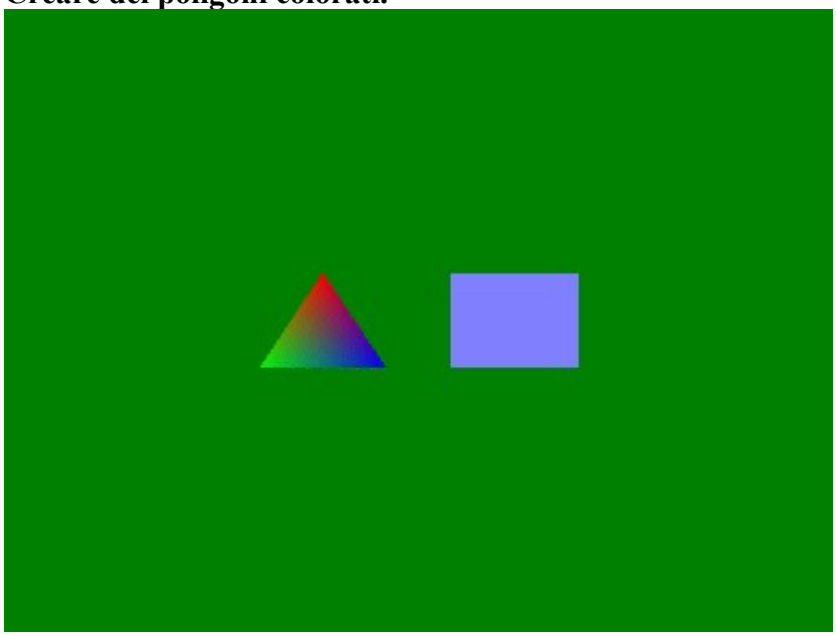
```
#include <allegro.h>
#include "alleggl.h"

void camera (void);
void oggetto (void);

// procedura di creazione e posizionamento della camera nella scena e prospettiva
void camera (void)
{
// aggiunge la prospettiva alla scena
glMatrixMode (GL_PROJECTION);

// comando simile ad un reset di conferma
glLoadIdentity ();

// indica le proporzioni della finestra e dell'angolo prospettico
// inoltre l'ultimo valore indica la profondità massima di visione della camera.
// Tutti gli oggetti oltre questo valore non saranno visti nella scena.
glFrustum (-1.0, 1.0, -1.0, 1.0, 1.0, 40.0);

// indica alla scena la matrice di visualizzazione sulle caratteristiche
// del modello
glMatrixMode (GL_MODELVIEW);

// Cancella il buffer del colore RGB ed il depth buffer(pulisce la scena)
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// comando simile ad un reset di conferma
glLoadIdentity();

//posiziona la camera
glTranslatef (0, 0, -0.5);

// orienta la camera nei rispettivi assi...
glRotatef (0, 1, 0, 0); //angolo x
glRotatef (0, 0, 1, 0); //angolo y
glRotatef (0, 0, 0, 1); //angolo z

// salva la posizione e rotazione della camera nella matrice scenica
glPushMatrix();
}

// posizione, orientamento, colore e creazione dell'oggetto
void oggetto (void)
{
/* Il comando 'glTranslatef(x,y,z)' sposta/posiziona un oggetto all'interno
della scena. Il centro della scena è il valore 0.0f in tutti gli assi.
Quindi per l'asse X si va a sinstra con i numeri negativi e a destra con quelli
positivi. Per l'asse Y si va in alto con i numeri positivi ed in basso con quelli
negativi. L'asse Z va all'interno dello schermo con i numeri negativi e
all'esterno con quelli positivi. */
glTranslatef(-1.5f,0.0f,-6.0f);

/* 'glBegin(GL_TRIANGLES)' indica che vogliamo iniziare a creare un triangolo.
Notoriamente un triangolo è composto da 3 vertici. Se noi volessimo creare 2 o
più triangoli è sufficiente aggiungere altre 3 linee di codice senza aggiungere
```

```
un nuovo 'glBegin(GL_TRIANGLES)' */
glBegin(GL_TRIANGLES); // disegna un triangolo

/* 'glVertex3f(x,y,z)' disegna un vertice nella scena alle coordinate indicate.
I valori sono indicati come unità. Perciò vi consiglio di fare delle prove prima di
disegnare i vostri poligoni, magari utilizzando della carta millimetrata... */
glVertex3f( 0.0f, 1.0f, 0.0f); // vertice altro
glVertex3f(-1.0f,-1.0f, 0.0f); // vertice a sinistra
glVertex3f( 1.0f,-1.0f, 0.0f); // vertice a destra
glEnd(); // Triangolo terminato

// Posizione nuovo poligono
glTranslatef(3.0f,0.0f,0.0f);

/* 'glBegin(GL_QUADS)' indica che vogliamo iniziare a creare un quadrato.
Notoriamente un quadrato è composto da 4 vertici. Se noi volessimo creare 2 o
più quadrati è sufficiente aggiungere altre 4 linee di codice senza aggiungere
un nuovo 'glBegin(GL_QUADS)' */

glBegin(GL_QUADS); // disegna un quadrato
glVertex3f(-1.0f, 1.0f, 0.0f); // vertice alto sinistra
glVertex3f( 1.0f, 1.0f, 0.0f); // vertice alto destra
glVertex3f( 1.0f,-1.0f, 0.0f); // vertice basso sinistra
glVertex3f(-1.0f,-1.0f, 0.0f); // vertice basso destra
glEnd(); // Quadrato terminato

/* PS: se volessimo disegnare un poligono con un numero superiore a 4 punti
dovremmo scrivere 'glBegin(GL_POLYGON)'... tutti i comandi 'glVertex3f(x,y,z)' che
servono e chiudere con il solito 'glEnd()' */
glFlush();
allegro_gl_flip();
}
int main ()
{
allegro_init();
install_allegro_gl();
allegro_gl_clear_settings();
allegro_gl_set (AGL_COLOR_DEPTH, 32);
allegro_gl_set (AGL_Z_DEPTH, 16);
allegro_gl_set (AGL_FULLSCREEN, TRUE);
allegro_gl_set (AGL_DOUBLEBUFFER, 1);
allegro_gl_set (AGL_SUGGEST, AGL_COLOR_DEPTH | AGL_Z_DEPTH | AGL_DOUBLEBUFFER
| AGL_FULLSCREEN);

// Attiva l'antialiasing (...,1). Disattiva l'antialiasing (...,0)
allegro_gl_set (AGL_RENDERMETHOD, 1);

// Profondità colore Palette
set_color_depth (32);

//Determina la modalità OPENGL e la risoluzione
set_gfx_mode(GFX_OPENGL, 640, 480, 0, 0);

install_keyboard();
install_timer();
allegro_gl_begin();
```

```
glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);

//Colore del fondale: (rosso,verde,blu,alfa) - max 1.0 min 0.0
glClearColor (0.0f, 0.5f, 0.0f, 0.5f);

//(GL_SMOOTH): Crea una migliore qualità di rendering(smussa i colori creando tonalità)
//es: (GL_FLAT): Il colore è il medesimo nella stessa faccia...
glShadeModel (GL_SMOOTH);

//INDICA AL COMPUTER CHE GLI OGGETTI SIAMO DISEGNATI NEL GIUSTO ORDINE
glEnable (GL_DEPTH_TEST);

//attiva il miglior livello di correzione prospettica(calo velocità)
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

allegro_gl_end();
do {
camera();
oggetto();
} while (!key[KEY_ESC]);
}
END_OF_MAIN();
```

Commento al codice:
Ammetto che i codici c per generare ambienti 3D sono decisamente più lunghi rispetto alle controparti 2D. In ogni caso credo valga la pena sacrificarsi un pochino per ottenere un risulta di altissimo impatto. In questo listato vengono dei poligoni senza alcun colore. Certo si è ancora molto lontani da applicarvi sopra texture bitmap. Ci arriveremo, promesso!

Creare dei poligoni colorati.



```
/* CODICE ESEMPLIFICATIVO SULLA GESTIONE DELLE OPENGL CON ALLEGRO
SCRITTO INTERAMENTE DA BARTOLOMEO DAVIDE BERTINETTO
```

WWW.BERTINETTOBARTOLOMEODAVIDE.IT

Listato: 3dn03poligonicolorati.c
*/

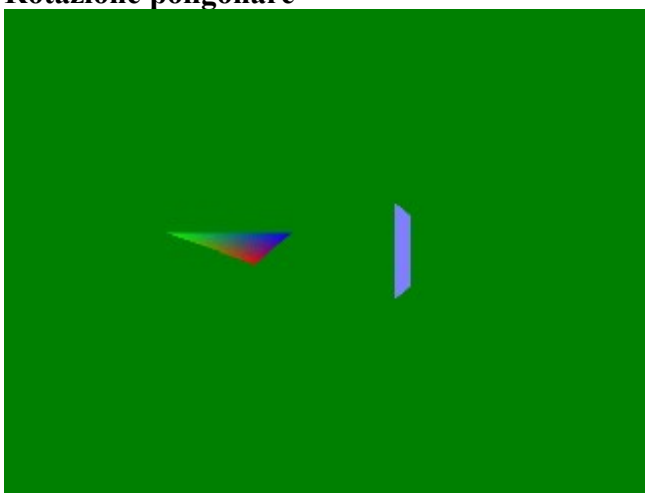
```
#include <allegro.h>
#include "alleggl.h"

void camera (void);
void oggetto (void);

// procedura di creazione e posizionamento della camera nella scena e prospettiva
void camera (void)
{

// aggiunge la prospettiva alla scena
glMatrixMode (GL_PROJECTION);

// comando simile ad un reset di conferma
glLoadIdentity ();

// indica le proporzioni della finestra e dell'angolo prospettico
// inoltre l'ultimo valore indica la profondità massima di visione della camera.
// Tutti gli oggetti oltre questo valore non saranno visti nella scena.

glFrustum (-1.0, 1.0, -1.0, 1.0, 1.0, 40.0);

// indica alla scena la matrice di visualizzazione sulle caratteristriche
// del modello
glMatrixMode (GL_MODELVIEW);

// Cancella il buffer del colore RGB ed il depth buffer(pulisce la scena)
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// comando simile ad un reset di conferma
glLoadIdentity();

// posiziona la camera
glTranslatef (0, 0, -0.5);

// orienta la camera nei rispettivi assi...
glRotatef (0, 1, 0, 0); //angolo x
glRotatef (0, 0, 1, 0); //angolo y
glRotatef (0, 0, 0, 1); //angolo z
// salva la posizione e rotazione della camera nella matrice scenica
glPushMatrix();

}

// posizione, orientamento, colore e creazione dell'oggetto

void oggetto (void)
{
```

```
/* Il comando 'glTranslatef(x,y,z)' sposta/posiziona un oggetto all'interno
della scena. Il centro della scena è il valore 0.0f in tutti gli assi.
Quindi per l'asse X si va a sinstra con i numeri negativi e a destra con quelli
positivi. Per l'asse Y si va in alto con i numeri positivi ed in basso con quelli
negativi. L'asse Z va all'interno dello schermo con i numeri negativi e
all'esterno con quelli positivi. */
glTranslatef(-1.5f,0.0f,-6.0f);

/* 'glBegin(GL_TRIANGLES)' indica che vogliamo iniziare a creare un triangolo.
Notoriamente un triangolo è composto da 3 vertici. Se noi volessimo creare 2 o
più triangoli è sufficiente aggiungere altre 3 linee di codice senza aggiungere
un nuovo 'glBegin(GL_TRIANGLES)' */
glBegin(GL_TRIANGLES); // disegna un triangolo

// Questo comando serve ad inserire un colore partendo da un vertice del poligono.
// E' importante sapere che tutto quello che è presente nel codice da questo punto
// sarà colorato del colore appena assegnato. Fino a quando non sarà incontrato
// un nuovo comando di colore.
glColor3f(1.0f,0.0f,0.0f); // assegna il colore rosso


/* 'glVertex3f(x,y,z)' disegna un vertice nella scena alle coordinate indicate.
I valori sono indicati come unità. Perciò vi consiglio di fare delle prove prima di
disegnare i vostri poligoni, magari utilizzando della carta millimetrata... */
glVertex3f( 0.0f, 1.0f, 0.0f); // vertice altro

glColor3f(0.0f,1.0f,0.0f); // assegna il colore verde

glVertex3f(-1.0f,-1.0f, 0.0f); // vertice a sinistra

glColor3f(0.0f,0.0f,1.0f); // assegna il colore blu

glVertex3f( 1.0f,-1.0f, 0.0f); // vertice a destra
glEnd(); // Triangolo terminato

// Nel poligono triangolare appena disegnano avremo i tre colori appena assegnati
// ai vertici che si fonderanno insieme creano un bell'effetto sfumato
// Posizione nuovo poligono
glTranslatef(3.0f,0.0f,0.0f);

/* 'glBegin(GL_QUADS)' indica che vogliamo iniziare a creare un quadrato.
Notoriamente un quadrato è composto da 4 vertici. Se noi volessimo creare 2 o
più quadrati è sufficiente aggiungere altre 4 linee di codice senza aggiungere
un nuovo 'glBegin(GL_QUADS)' */

/* Inserendo il comando di colore subito prima della procedura di creazione
del policono quadrilatero assegneremo a tutta la superfice un colore omogeneo
*/
glColor3f(0.5f,0.5f,1.0f); // Colore azzurrino

glBegin(GL_QUADS); // disegna un quadrato
glVertex3f(-1.0f, 1.0f, 0.0f); // vertice alto sinistra
glVertex3f( 1.0f, 1.0f, 0.0f); // vertice alto destra
glVertex3f( 1.0f,-1.0f, 0.0f); // vertice basso sinistra
glVertex3f(-1.0f,-1.0f, 0.0f); // vertice basso destra
glEnd(); // Quadrato terminato
```

```
/* PS: se volessimo disegnare un poligono con un numero superiore a 4 punti
dovremmo scrivere 'glBegin(GL_POLYGON)'... tutti i comandi 'glVertex3f(x,y,z)' che
servono e chiudere con il solito 'glEnd()' */
glFlush();
allegro_gl_flip();
}

int main ()
{
allegro_init();
install_allegro_gl();
allegro_gl_clear_settings();
allegro_gl_set (AGL_COLOR_DEPTH, 32);
allegro_gl_set (AGL_Z_DEPTH, 16);
allegro_gl_set (AGL_FULLSCREEN, TRUE);
allegro_gl_set (AGL_DOUBLEBUFFER, 1);
allegro_gl_set (AGL_SUGGEST, AGL_COLOR_DEPTH | AGL_Z_DEPTH | AGL_DOUBLEBUFFER
| AGL_FULLSCREEN);

// Attiva l'antialiasing (...,1). Disattiva l'antialiasing (...,0)
allegro_gl_set (AGL_RENDERMETHOD, 1);

// Profondità colore Palette
set_color_depth (32);

//Determina la modalità OPENGL e la risoluzione
set_gfx_mode(GFX_OPENGL, 640, 480, 0, 0);

install_keyboard();
install_timer();
allegro_gl_begin();
glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);

//Colore del fondale: (rosso,verde,blu,alfa) - max 1.0 min 0.0
glClearColor (0.0f, 0.5f, 0.0f, 0.5f);

//(GL_SMOOTH): Crea una migliore qualità di rendering(smussa i colori creando tonalità)
//es: (GL_FLAT): Il colore è il medesimo nella stessa faccia...
glShadeModel (GL_SMOOTH);

//INDICA AL COMPUTER CHE GLI OGGETTI SIANO DISEGNATI NEL GIUSTO ORDINE
glEnable (GL_DEPTH_TEST);
//attiva il miglior livello di correzione prospettica(calo velocità)
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
allegro_gl_end();

do {
camera();
oggetto();
} while (!key[KEY_ESC]);
}
END_OF_MAIN();
```

Commento al codice:
I primi passi verso il miglioramento grafico si stanno compiendo. In questo esempio sono stati applicati dei semplici colori ai poligoni che abbiamo creato. Noterete che l'aspetto generale del risultato cambia moltissimo rispetto all'esempio precedente, pur restando i soliti due poligoni.

Rotazione poligonare



```
/* CODICE ESEMPLIFICATIVO SULLA GESTIONE DELLE OPENGL CON ALLEGRO
SCRITTO INTERAMENTE DA BARTOLOMEO DAVIDE BERTINETTO

WWW.BERTINETTOBARTOLOMEODAVIDE.IT

Listato: 3dn04rotazionepoligoni.c
*/

#include <allegro.h>
#include "alleggl.h"

void camera (void);
void oggetto (void);
void rotazione (void);

int rtri=0; // variabile azzerata che ruoterà di n gradi il triangolo
int rquad=0; // variabile azzerata che ruoterà di n gradi il quadrato

// procedura di creazione e posizionamento della camera nella scena e prospettiva
void camera (void)
{

// aggiunge la prospettiva alla scena
glMatrixMode (GL_PROJECTION);

// comando simile ad un reset di conferma
glLoadIdentity ();

// indica le proporzioni della finestra e dell'angolo prospettico
// inoltre l'ultimo valore indica la profondità massima di visione della camera.
// Tutti gli oggetti oltre questo valore non saranno visti nella scena.
```

```
glFrustum (-1.0, 1.0, -1.0, 1.0, 1.0, 40.0);

// indica alla scena la matrice di visualizzazione sulle caratteristiche
// del modello
glMatrixMode (GL_MODELVIEW);

// Cancella il buffer del colore RGB ed il depth buffer(pulisce la scena)
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// comando simile ad un reset di conferma
glLoadIdentity();

//posiziona la camera
glTranslatef (0, 0, -0.5);

// orienta la camera nei rispettivi assi...
glRotatef (0, 1, 0, 0); //angolo x
glRotatef (0, 0, 1, 0); //angolo y
glRotatef (0, 0, 0, 1); //angolo z

// salva la posizione e rotazione della camera nella matrice scenica
glPushMatrix();
}

// posizione, orientamento, colore e creazione dell'oggetto

void oggetto (void)
{

/* Il comando 'glTranslatef(x,y,z)' sposta/posiziona un oggetto all'interno
della scena. Il centro della scena è il valore 0.0f in tutti gli assi.
Quindi per l'asse X si va a sinistra con i numeri negativi e a destra con quelli
positivi. Per l'asse Y si va in alto con i numeri positivi ed in basso con quelli
negativi. L'asse Z va all'interno dello schermo con i numeri negativi e
all'esterno con quelli positivi. */

/* tutto quello che segue il comando 'glLoadIdenty();' viene separato da quello che
è successo prima. */
glLoadIdentity(); // aggiunto per azzerare la scenda ad ogni ciclo. Separatore
glTranslatef(-1.5f,0.0f,-6.0f);

/* Il comando 'glRotatef(...);' serve per ruotare uno o più poligoni che lo seguono,
per questo viene usato 'glLoadIdentity();' come separatore.
Il primo valore, in questo caso la variabile 'rtri' determina i gradi di rotazione
del poligono (da 0 a 359). I tre valori che seguono indicano l'asse sul quale deve
essere effettuata la rotazione. Quindi ad esempio per l'asse X = 1.0f per si e 0.0f
per no. Stesso discorso per l'asse Y e Z. */
glRotatef(rtri,1.0f,0.0f,0.0f); // Ruota di n gradi nell'asse X

/* 'glBegin(GL_TRIANGLES)' indica che vogliamo iniziare a creare un triangolo.
Notoriamente un triangolo è composto da 3 vertici. Se noi volessimo creare 2 o
più triangoli è sufficiente aggiungere altre 3 linee di codice senza aggiungere
un nuovo 'glBegin(GL_TRIANGLES)' */
glBegin(GL_TRIANGLES); // disegna un triangolo
```

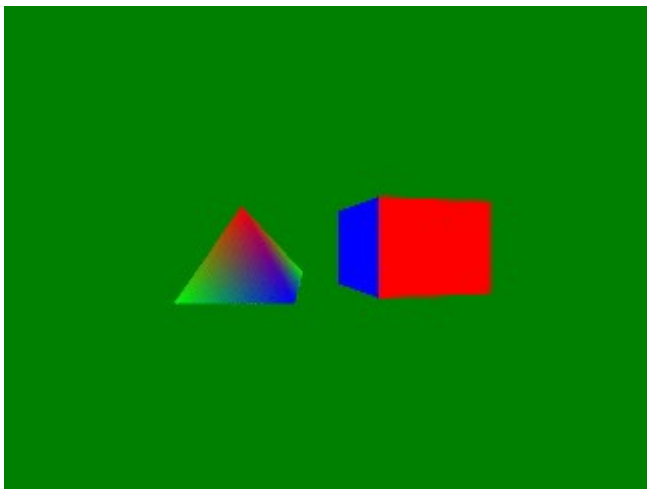
```
// Questo comando serve ad inserire un colore partendo da un vertice del poligono.
// E' importante sapere che tutto quello che è presente nel codice da questo punto
// sarà colorato del colore appena assegnato. Fino a quando non sarà incontrato
// un nuovo comando di colore.
glColor3f(1.0f,0.0f,0.0f); // assegna il colore rosso


/* 'glVertex3f(x,y,z)' disegna un vertice nella scena alle coordinate indicate.
I valori sono indicati come unità. Perciò vi consiglio di fare delle prove prima di
disegnare i vostri poligoni, magari utilizzando della carta millimetrata... */
glVertex3f( 0.0f, 1.0f, 0.0f); // vertice altro
glColor3f(0.0f,1.0f,0.0f); // assegna il colore verde
glVertex3f(-1.0f,-1.0f, 0.0f); // vertice a sinistra
glColor3f(0.0f,0.0f,1.0f); // assegna il colore blu
glVertex3f( 1.0f,-1.0f, 0.0f); // vertice a destra
glEnd(); // Triangolo terminato

// Nel poligono triangolare appena disegnano avremo i tre colori appena assegnati
// ai vertici che si fonderanno insieme creano un bell'effetto sfumato
glLoadIdentity(); // aggiunto per azzerare la scenda ad ogni ciclo. Separatore

// Posizione nuovo poligono
glTranslatef(1.5f,0.0f,-6.0f);
glRotatef(rquad,0.0f,1.0f,0.0f); // Ruota di n gradi nell'asse Y

/* Inserendo il comando di colore subito prima della procedura di creazione
del poligono quadrilatero assegneremo a tutta la superficie un colore omogeneo
*/
glColor3f(0.5f,0.5f,1.0f); // Colore azzurrino

/* 'glBegin(GL_QUADS)' indica che vogliamo iniziare a creare un quadrato.
Notoriamente un quadrato è composto da 4 vertici. Se noi volessimo creare 2 o
più quadrati è sufficiente aggiungere altre 4 linee di codice senza aggiungere
un nuovo 'glBegin(GL_QUADS)' */
glBegin(GL_QUADS); // disegna un quadrato
glVertex3f(-1.0f, 1.0f, 0.0f); // vertice alto sinistra
glVertex3f( 1.0f, 1.0f, 0.0f); // vertice alto destra
glVertex3f( 1.0f,-1.0f, 0.0f); // vertice basso sinistra
glVertex3f(-1.0f,-1.0f, 0.0f); // vertice basso destra
glEnd(); // Quadrato terminato

/* PS: se volessimo disegnare un poligono con un numero superiore a 4 punti
dovremmo scrivere 'glBegin(GL_POLYGON)'... tutti i comandi 'glVertex3f(x,y,z)' che
servono e chiudere con il solito 'glEnd()' */
glFlush();
allegro_gl_flip();
}

// Procedura di rotazione
void rotazione (void) {
rtri++; // Incrementa la variabile di un unità di grado ogni ciclo.
rquad++; // Incrementa la variabile di un unità di grado ogni ciclo.
if (rtri>359) rtri=0; // se è maggiore di 369 ritorna a 0. Giro completo
if (rquad>359) rquad=0; // se è maggiore di 369 ritorna a 0. Giro completo
}
int main ()
```

```
{
allegro_init();
install_allegro_gl();
allegro_gl_clear_settings();
allegro_gl_set (AGL_COLOR_DEPTH, 32);
allegro_gl_set (AGL_Z_DEPTH, 16);
allegro_gl_set (AGL_FULLSCREEN, TRUE);
allegro_gl_set (AGL_DOUBLEBUFFER, 1);
allegro_gl_set (AGL_SUGGEST, AGL_COLOR_DEPTH | AGL_Z_DEPTH | AGL_DOUBLEBUFFER
| AGL_FULLSCREEN);

// Attiva l'antialiasing (...,1). Disattiva l'antialiasing (...,0)
allegro_gl_set (AGL_RENDERMETHOD, 1);

// Profondità colore Palette
set_color_depth (32);

//Determina la modalità OPENGL e la risoluzione
set_gfx_mode(GFX_OPENGL, 640, 480, 0, 0);

install_keyboard();
install_timer();
allegro_gl_begin();
glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);

//Colore del fondale: (rosso,verde,blu,alfa) - max 1.0 min 0.0
glClearColor (0.0f, 0.5f, 0.0f, 0.5f);

//(GL_SMOOTH): Crea una migliore qualità di rendering(smussa i colori creando tonalità)
//es: (GL_FLAT): Il colore è il medesimo nella stessa faccia...
glShadeModel (GL_SMOOTH);

//INDICA AL COMPUTER CHE GLI OGGETTI SIANO DISEGNATI NEL GIUSTO ORDINE
glEnable (GL_DEPTH_TEST);

//attiva il miglior livello di correzione prospettica(calo velocità)
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

allegro_gl_end();

do {
camera();
oggetto();
rotazione(); // richiama la procedura di rotazione

} while (!key[KEY_ESC]);

}
END_OF_MAIN();
```

Commento al codice:
La creazione di un'ambiente tridimensionale è di per se estremamente affascinante ma l'impatto ottenibile può essere ulteriormente migliorato se sono aggiunte delle animazioni. Su questo frangente il codice

sopra sviluppato contempla il movimento rotatorio dei poligoni. L'azione del movimento, con semplici modifiche, può essere estesa al posizionamento sui tre assi (x, y, z)...

Generare oggetti solidi



```
/* CODICE ESEMPLIFICATIVO SULLA GESTIONE DELLE OPENGL CON ALLEGRO
SCRITTO INTERAMENTE DA BARTOLOMEO DAVIDE BERTINETTO

WWW.BERTINETTOBARTOLOMEODAVIDE.IT
Listato: 3dn05oggettisolidi.c
*/

#include <allegro.h>
#include "alleggl.h"

void camera (void);
void oggetto (void);
void rotazione (void); // nuova procedura

int rtri=0; // variabile azzerata che ruoterà di n gradi il triangolo
int rquad=0; // variabile azzerata che ruoterà di n gradi il quadrato

// procedura di creazione e posizionamento della camera nella scena e prospettiva
void camera (void)
{

// aggiunge la prospettiva alla scena
glMatrixMode (GL_PROJECTION);

// comando simile ad un reset di conferma
glLoadIdentity ();

// indica le proporzioni della finestra e dell'angolo prospettico
// inoltre l'ultimo valore indica la profondità massima di visione della camera.
// Tutti gli oggetti oltre questo valore non saranno visti nella scena.
glFrustum (-1.0, 1.0, -1.0, 1.0, 1.0, 40.0);

// indica alla scena la matrice di visualizzazione sulle caratteristiche
// del modello
```

```
glMatrixMode (GL_MODELVIEW);

// Cancella il buffer del colore RGB ed il depth buffer(pulisce la scena)
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// comando simile ad un reset di conferma
glLoadIdentity();

//posiziona la camera
glTranslatef (0, 0, -0.5);

// orienta la camera nei rispettivi assi...
glRotatef (0, 1, 0, 0); //angolo x
glRotatef (0, 0, 1, 0); //angolo y
glRotatef (0, 0, 0, 1); //angolo z

// salva la posizione e rotazione della camera nella matrice scenica
glPushMatrix();

}

// posizione, orientamento, colore e creazione dell'oggetto
void oggetto (void)
{

/* Il comando 'glTranslatef(x,y,z)' sposta/posiziona un oggetto all'interno
della scena. Il centro della scena è il valore 0.0f in tutti gli assi.
Quindi per l'asse X si va a sinistra con i numeri negativi e a destra con quelli
positivi. Per l'asse Y si va in alto con i numeri positivi ed in basso con quelli
negativi. L'asse Z va all'interno dello schermo con i numeri negativi e
all'esterno con quelli positivi. */

/* tutto quello che segue il comando 'glLoadIdenty();' viene separato da quello che
è successo prima. */
glLoadIdentity(); // aggiunto per azzerare la scena ad ogni ciclo. Separatore
glTranslatef(-1.5f,0.0f,-6.0f);

/* Il comando 'glRotatef(...);' serve per ruotare uno o più poligoni che lo seguono,
per questo viene usato 'glLoadIdentity();' come separatore.
Il primo valore, in questo caso la variabile 'rtri' determina i gradi di rotazione
del poligono (da 0 a 359). I tre valori che seguono indicano l'asse sul quale deve
essere effettuata la rotazione. Quindi ad esempio per l'asse X = 1.0f per si e 0.0f
per no. Stesso discorso per l'asse Y e Z. */
glRotatef(rtri,1.0f,0.0f,0.0f); // Ruota di n gradi nell'asse X
glBegin(GL_TRIANGLES); // Disegna una piramide

// Prima faccia
glColor3f(1.0f,0.0f,0.0f);
glVertex3f( 0.0f, 1.0f, 0.0f);
glColor3f(0.0f,1.0f,0.0f);
glVertex3f(-1.0f,-1.0f, 1.0f);
glColor3f(0.0f,0.0f,1.0f);
glVertex3f( 1.0f,-1.0f, 1.0f);

// Seconda faccia
glColor3f(1.0f,0.0f,0.0f);
```

```
glVertex3f( 0.0f, 1.0f, 0.0f);
glColor3f(0.0f,0.0f,1.0f);
glVertex3f( 1.0f,-1.0f, 1.0f);
glColor3f(0.0f,1.0f,0.0f);
glVertex3f( 1.0f,-1.0f, -1.0f);

// Terza faccia
glColor3f(1.0f,0.0f,0.0f);
glVertex3f( 0.0f, 1.0f, 0.0f);
glColor3f(0.0f,1.0f,0.0f);
glVertex3f( 1.0f,-1.0f, -1.0f);
glColor3f(0.0f,0.0f,1.0f);
glVertex3f(-1.0f,-1.0f, -1.0f);

// Quarta faccia
glColor3f(1.0f,0.0f,0.0f);
glVertex3f( 0.0f, 1.0f, 0.0f);
glColor3f(0.0f,0.0f,1.0f);
glVertex3f(-1.0f,-1.0f,-1.0f);
glColor3f(0.0f,1.0f,0.0f);
glVertex3f(-1.0f,-1.0f, 1.0f);
glEnd(); // Piramide terminata

// Nella piramide appena disegnata avremo i tre colori appena assegnati
// ai vertici che si fonderanno insieme creano un bell'effetto sfumato
glLoadIdentity(); // aggiunto per azzerare la scena ad ogni ciclo. Separatore

// Posizione nuovo poligono
glTranslatef(1.5f,0.0f,-6.0f);
glRotatef(rquad,0.0f,1.0f,0.0f); // Ruota di n gradi nell'asse Y
glBegin(GL_QUADS); // disegna un cubo

// Prima faccia
glColor3f(0.5f,0.5f,1.0f);
glVertex3f( 1.0f, 1.0f,-1.0f);
glVertex3f(-1.0f, 1.0f,-1.0f);
glVertex3f(-1.0f, 1.0f, 1.0f);
glVertex3f( 1.0f, 1.0f, 1.0f);

// Seconda faccia
glColor3f(1.0f,0.5f,0.0f);
glVertex3f( 1.0f,-1.0f, 1.0f);
glVertex3f(-1.0f,-1.0f, 1.0f);
glVertex3f(-1.0f,-1.0f,-1.0f);
glVertex3f( 1.0f,-1.0f,-1.0f);

// Terza faccia
glColor3f(1.0f,0.0f,0.0f);
glVertex3f( 1.0f, 1.0f, 1.0f);
glVertex3f(-1.0f, 1.0f, 1.0f);
glVertex3f(-1.0f,-1.0f, 1.0f);
glVertex3f( 1.0f,-1.0f, 1.0f);

// Quarta faccia
glColor3f(1.0f,1.0f,0.0f);
glVertex3f( 1.0f,-1.0f,-1.0f);
```

```
glVertex3f(-1.0f,-1.0f,-1.0f);
glVertex3f(-1.0f, 1.0f,-1.0f);
glVertex3f( 1.0f, 1.0f,-1.0f);

// Quinta faccia
glColor3f(0.0f,0.0f,1.0f);
glVertex3f(-1.0f, 1.0f, 1.0f);
glVertex3f(-1.0f, 1.0f,-1.0f);
glVertex3f(-1.0f,-1.0f,-1.0f);
glVertex3f(-1.0f,-1.0f, 1.0f);

// Sesta faccia
glColor3f(1.0f,0.0f,1.0f);
glVertex3f( 1.0f, 1.0f,-1.0f);
glVertex3f( 1.0f, 1.0f, 1.0f);
glVertex3f( 1.0f,-1.0f, 1.0f);
glVertex3f( 1.0f,-1.0f,-1.0f);
glEnd(); // Cubo terminato

glFlush();
allegro_gl_flip();
}

// Procedura di rotazione
void rotazione (void) {

rtri=rtri+5; // Incrementa la variabile di 5 unità di grado ogni ciclo.
rquad=rquad+5; // Incrementa la variabile di 5 unità di grado ogni ciclo.

if (rtri>359) rtri=0; // se è maggiore di 369 ritorna a 0. Giro completo
if (rquad>359) rquad=0; // se è maggiore di 369 ritorna a 0. Giro completo
}

int main ()
{
allegro_init();
install_allegro_gl();
allegro_gl_clear_settings();
allegro_gl_set (AGL_COLOR_DEPTH, 32);
allegro_gl_set (AGL_Z_DEPTH, 16);
allegro_gl_set (AGL_FULLSCREEN, TRUE);
allegro_gl_set (AGL_DOUBLEBUFFER, 1);
allegro_gl_set (AGL_SUGGEST, AGL_COLOR_DEPTH | AGL_Z_DEPTH | AGL_DOUBLEBUFFER
| AGL_FULLSCREEN);

// Attiva l'antialiasing (...,1). Disattiva l'antialiasing (...,0)
allegro_gl_set (AGL_RENDERMETHOD, 1);

// Profondità colore Palette
set_color_depth (32);

//Determina la modalità OPENGL e la risoluzione
set_gfx_mode(GFX_OPENGL, 640, 480, 0, 0);
install_keyboard();
install_timer();
allegro_gl_begin();
```

```
// glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);

//Colore del fondale: (rosso,verde,blu,alfa) - max 1.0 min 0.0
glClearColor (0.0f, 0.5f, 0.0f, 0.5f);

//(GL_SMOOTH): Crea una migliore qualità di rendering(smussa i colori creando tonalità)
//es: (GL_FLAT): Il colore è il medesimo nella stessa faccia...
glShadeModel (GL_SMOOTH);

//INDICA AL COMPUTER CHE GLI OGGETTI SIANO DISEGNATI NEL GIUSTO ORDINE
glEnable (GL_DEPTH_TEST);

//attiva il miglior livello di correzione prospettica(calo velocità)
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

allegro_gl_end();
do {
camera();
oggetto();
rotazione(); // richiama la procedura di rotazione

} while (!key[KEY_ESC]);
}
END_OF_MAIN();
```

Commento al codice:
Fino ad ora abbiamo esaminato 4 listati che avevano ben poco a che vedere con la grafica tridimensionale a parte il motore 3D delle openGL. Con questo quinto listato creiamo finalmente due oggetti solidi(un cubo ed una piramide). Queste due forme sono dette in gergo 3D, primitive, esistono molte altre forme di base: sfere, cilindri, coni, ecc... Mettendo insieme tanti oggetti di questo tipo potremo generare veri e propri mondi virtuali a tre dimensioni!

Oggetti solidi con texture ed illuminazione


```
/* CODICE ESEMPLIFICATIVO SULLA GESTIONE DELLE OPENGL CON ALLEGRO
```

SCRITTO INTERAMENTE DA BARTOLOMEO DAVIDE BERTINETTO

WWW.BERTINETTOBARTOLOMEODAVIDE.IT
Listato: 3dn06oggettisoliditexture.c
*/

```
#include <allegro.h>
#include "alleggl.h"
void camera (void);
void oggetto (void);
void rotazione (void);

// variabile azzerata che ruoterà di n gradi il quadrato
int rquad=0;

// variabile alla quale verra assegnata la texture
GLuint tex;

// procedura di creazione e posizionamento della camera nella scena e prospettiva
void camera (void)
{

// aggiunge la prospettiva alla scena
glMatrixMode (GL_PROJECTION);

// comando simile ad un reset di conferma
glLoadIdentity ();

// indica le proporzioni della finestra e dell'angolo prospettico
// inoltre l'ultimo valore indica la profondità massima di visione della camera.
// Tutti gli oggetti oltre questo valore non saranno visti nella scena.
glFrustum (-1.0, 1.0, -1.0, 1.0, 1.0, 40.0);

// indica alla scena la matrice di visualizzazione sulle caratteristiche
// del modello
glMatrixMode (GL_MODELVIEW);

// Cancella il buffer del colore RGB ed il depth buffer(pulisce la scena)
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// comando simile ad un reset di conferma
glLoadIdentity();


// posiziona la camera
glTranslatef (0, 0, -0.5);

// orienta la camera nei rispettivi assi...
glRotatef (0, 1, 0, 0); //angolo x
glRotatef (0, 0, 1, 0); //angolo y
glRotatef (0, 0, 0, 1); //angolo z

// salva la posizione e rotazione della camera nella matrice scenica
glPushMatrix();
}
```

```
// posizione, orientamento, colore e creazione dell'oggetto
void oggetto (void)
{

// aggiunto per azzerare la scena ad ogni ciclo. Separatore
glLoadIdentity();

// Posizione nuovo poligono

glTranslatef(0.0f,0.0f,-3.0f);
glRotatef(rquad,0.0f,1.0f,0.0f); // Ruota di n gradi nell'asse Y

/* questo comando assegna una determinata texture ( indicata dalla variabile 'tex')
a tutte le facce dell'aggetto racchiuse tra 'glBegin(GL_QUADS);' e 'glEnd();'.
Quindi se si vuole assegnare una nuova texture ad un'altra faccia del medesimo
oggetto si dovra aprire un altro 'glBegin(GL_QUADS);' e 'glEnd();'. */
glBindTexture (GL_TEXTURE_2D, tex);
glBegin(GL_QUADS);


/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);
// Faccia posteriore
/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
// Faccia in alto
/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, 1.0f, 1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, 1.0f, 1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);
// Faccia in basso
/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
// Faccia di destra
/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);
// Faccia di sinistra
/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
```

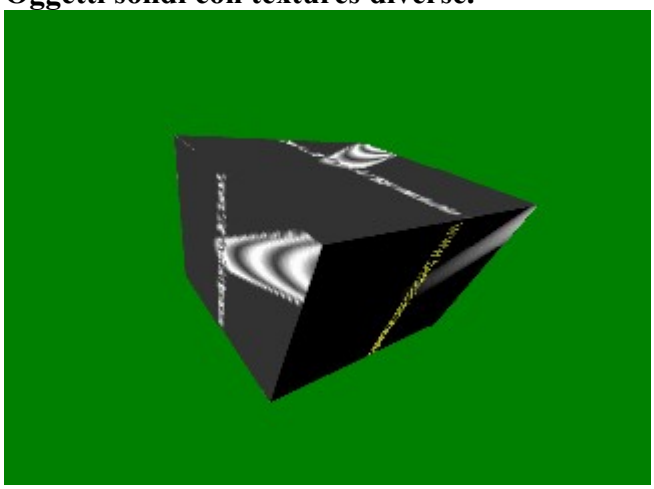
```
glEnd();
glFlush();
allegro_gl_flip();
}

// Procedura di rotazione
void rotazione (void) {
rquad=rquad+1; // Incrementa la variabile di 1 unità di grado ogni ciclo.
if (rquad>359) rquad=0; // se è maggiore di 359 ritorna a 0. Giro completo
}

// procedura che carica un texture
void setup_textures (void)
{
PALETTE pal; // variabile di palette
BITMAP *bmp, *bmp2; // carico le variabile bitmap
int w, h; // variabili di altezza e larghezza immagine
bmp = load_bitmap ("www.bertinettobartolomeodavide.it0001.bmp", pal); // Carico file immagine

// determina la qualità del dettaglio della texture
w = 256;
h = 256;

// creo il l'area bitmap, che sarà tanto più dettagliata quanto più saranno alti
// 'w' e 'h'. Pena perdita di velocità nell'esecuzione video.
bmp2 = create_bitmap (w, h);

// ridimensiono e centro l'immagine contenuta in 'bmp' e la traferisco in 'bmp2'
stretch_blit (bmp, bmp2, 0, 0, bmp->w, bmp->h, 0, 0, w, h);
destroy_bitmap (bmp); // pulisco la variabile dell'immagine texture

// parametri di visualizzazione gl della texture sul poligono
allegro_gl_begin();
glDisable (GL_DITHER); // disattivo il dithering delle texture (grana)
glEnable (GL_TEXTURE_2D); // attivo la texture 2d
glTexEnvi (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
glHint (GL_PERSPECTIVE_CORRECTION_HINT, GL_FASTEST);

// sposto il contenuto di 'bmp2' nella variabile 'tex'
tex = allegro_gl_make_texture (bmp2);
destroy_bitmap (bmp2); // elimino il contenuto di 'bmp2'
}

int main ()
{
allegro_init();
install_allegro_gl();
allegro_gl_clear_settings();
allegro_gl_set (AGL_COLOR_DEPTH, 32);
allegro_gl_set (AGL_Z_DEPTH, 16);
allegro_gl_set (AGL_FULLSCREEN, TRUE);
allegro_gl_set (AGL_DOUBLEBUFFER, 1);
allegro_gl_set (AGL_SUGGEST, AGL_COLOR_DEPTH | AGL_Z_DEPTH | AGL_DOUBLEBUFFER
| AGL_FULLSCREEN);

// Attiva l'antialiasing (...,1). Disattiva l'antialiasing (...,0)
```

```
allegro_gl_set (AGL_RENDERMETHOD, 1);

// Profondità colore Palette
set_color_depth (32);

//Determina la modalità OPENGL e la risoluzione
set_gfx_mode(GFX_OPENGL, 640, 480, 0, 0);
install_keyboard();
install_timer();
allegro_gl_begin();
glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);

//(GL_SMOOTH): Crea una migliore qualità di rendering(smussa i colori creando tonalità)
//es: (GL_FLAT): Il colore è il medesimo nella stessa faccia...
glShadeModel (GL_SMOOTH);

// INDICA AL COMPUTER CHE GLI OGGETTI SIANO DISEGNATI NEL GIUSTO ORDINE
glEnable (GL_DEPTH_TEST);

// attiva il miglior livello di correzione prospettica(calo velocità)
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
// luce bianca diffusa (red,green,blue,alpha)
GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};

// posizione infinita luce (x,y,z,infinita -0.0=si-1.0=no-)
GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};

// Assegna alla luce il colore stabilito
glLightfv (GL_LIGHT0, GL_DIFFUSE, light_diffuse);

// Assegna alla luce la posizione stabilita
glLightfv (GL_LIGHT0, GL_POSITION, light_position);

// attiva una luce specifica 'LIGHT0'
glEnable (GL_LIGHT0);

// Attiva l'illuminazione
glEnable (GL_LIGHTING);

allegro_gl_end();
do {
camera();
oggetto();
rotazione();
setup_textures();
} while (!key[KEY_ESC]);
}
END_OF_MAIN();
```

Commento al codice:
Come promesso in precedenza, ecco il primo esempio che permette ti utilizzare delle texture per completare in modo realistico i vostri oggetti solidi. Che cos'è però una texture? Le texture non sono niente altro che immagini, disegnate, fotografate o scannerizzate che vengono sovrapposte alle facce che compongono un oggetto poligonare, allo scopo di renderlo più realistico, facendolo assomigliare nei

materiali agli oggetti reali!

Oggetti solidi con textures diverse.



```
/* CODICE ESEMPLIFICATIVO SULLA GESTIONE DELLE OPENGL CON ALLEGRO
SCRITTO INTERAMENTE DA BARTOLOMEO DAVIDE BERTINETTO

WWW.BERTINETTOBARTOLOMEODAVIDE.IT
listato: 3dn07oggettisoliditexturediverse.c
*/

#include <allegro.h>
#include "alleggl.h"

void camera (void);
void oggetto (void);
void rotazione (void);
void setup_textures (void); // nuova procedura
void setup_textures2 (void); // nuova procedura

int rquad=0; // variabile azzerata che ruoterà di n gradi il quadrato
GLuint tex; // variabile alla quale verra assegnata la texture
GLuint tex2; // variabile alla quale verra assegnata la texture n°2

// procedura di creazione e posizionamento della camera nella scena e prospettiva
void camera (void)
{

// aggiunge la prospettiva alla scena
glMatrixMode (GL_PROJECTION);

// comando simile ad un reset di conferma
glLoadIdentity ();

// indica le proporzioni della finestra e dell'angolo prospettico
// inoltre l'ultimo valore indica la profondità massima di visione della camera.
// Tutti gli oggetti oltre questo valore non saranno visti nella scena.
glFrustum (-1.0, 1.0, -1.0, 1.0, 1.0, 40.0);
```

```
// indica alla scena la matrice di visualizzazione sulle caratteristriche
// del modello
glMatrixMode (GL_MODELVIEW);

// Cancella il buffer del colore RGB ed il depth buffer(pulisce la scena)
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// comando simile ad un reset di conferma
glLoadIdentity();

//posiziona la camera
glTranslatef (0, 0, -0.5);

// orienta la camera nei rispettivi assi...
glRotatef (0, 1, 0, 0); //angolo x
glRotatef (0, 0, 1, 0); //angolo y
glRotatef (0, 0, 0, 1); //angolo z

// salva la posizione e rotazione della camera nella matrice scenica
glPushMatrix();
}

// posiozione, orientamento, colore e creazione dell'oggetto
void oggetto (void)
{

// aggiunto per azzerare la scena ad ogni ciclo. Separatore
glLoadIdentity(); e

// Posizione nuovo poligono
glTranslatef(0.0f,0.0f,-3.0f);
glRotatef(rquad,1.0f,1.0f,0.0f); // Ruota di n gradi nell'asse X e Y

/* questo comano assegna una determinata texture ( indicata dalla variabile 'tex')
a tutte le faccie dell'aggetto racchiuse tra 'glBegin(GL_QUADS);' e 'glEnd();'.
Quindi se si vuole assegnare una nuova texture ad un'altra faccia del medesimo
oggetto si dovra aprire un altro 'glBegin(GL_QUADS);' e 'glEnd();'. */
glBindTexture (GL_TEXTURE_2D, tex);
glBegin(GL_QUADS);

// Faccia frontale
/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);

// Faccia posteriore
/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);

// Faccia in alto
/* Creo la faccia del cubo ed applico la texture su di essa */
```

```
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, 1.0f, 1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, 1.0f, 1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);
glEnd(); // concludo le prime tre facce contenenti la texture 1

// applico una nuova texture 'tex2' alle faccie che seguono
glBindTexture (GL_TEXTURE_2D, tex2);

// disegno le altre 3 faccie
glBegin(GL_QUADS);

// Faccia in basso
/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);

// Faccia di destra
/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);
// Faccia di sinistra
/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
glEnd(); // concludo le ultime tre facie con la nuoa texture 'tex2'
glFlush();
allegro_gl_flip();
}

// Procedura di rotazione
void rotazione (void) {
rquad=rquad+1; // Incrementa la variabile di 1 unità di grado ogni ciclo.
if (rquad>359) rquad=0; // se è maggiore di 359 ritorna a 0. Giro completo
}

// procedura che carica un texture
void setup_textures (void)
{
PALETTE pal; // variabile di palette
BITMAP *bmp, *bmp2; // carico le variabili bitmap
int w, h; // variabili di altezza e larghezza immagine
bmp = load_bitmap ("www.bertinettobartolomeodavide.it0001.bmp", pal); // Carico file immagine

// determina la qualità del dettaglio della texture in pixel
w = 256;
h = 256;

// creo il l'area bitmap, che sara tanto più dettagliata quanto più saranno alti
// 'w' e 'h'. Pena perdita di velocità nell'esecuzione video.
```

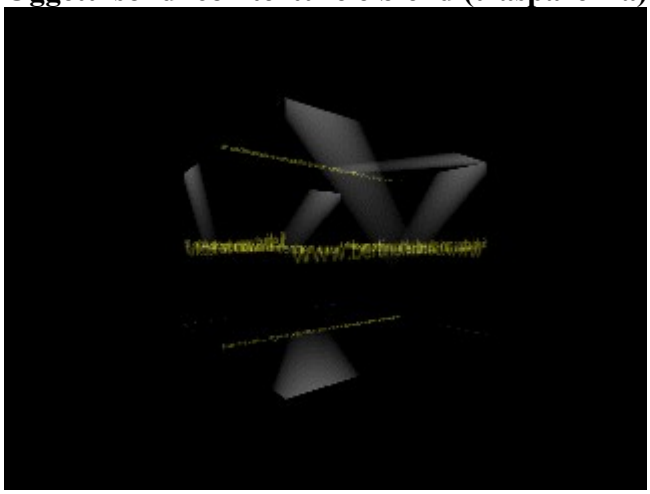
```
bmp2 = create_bitmap (w, h);

// ridimensiono e centro l'immagine contenuta in 'bmp' e la traferisco in 'bmp2'
stretch_blit (bmp, bmp2, 0, 0, bmp->w, bmp->h, 0, 0, w, h);
destroy_bitmap (bmp); // pulisco la variabile dell'immagine texture

// parametri di visualizzazione gl della texture sul poligono
allegro_gl_begin();
glDisable (GL_DITHER); // disattivo il dithering delle texture (grana)
glEnable (GL_TEXTURE_2D); // attivo la texture 2d
glTexEnvi (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
glHint (GL_PERSPECTIVE_CORRECTION_HINT, GL_FASTEST);

// sposto il contenuto di 'bmp2' nella variabile 'tex'
tex = allegro_gl_make_texture (bmp2);
destroy_bitmap (bmp2); // elimino il contenuto di 'bmp2'
}

// procedura per il caricamento e la creazione di una seconda texture
void setup_textures2 (void)
{
PALETTE pala; // variabile di palette 2
BITMAP *bmpa, *bmpb; // carico le variabile bitmap 2
int wa, hb; // variabili di altezza e larghezza immagine 2
bmpa = load_bitmap ("www.bertinettobartolomeodavide.it0002.bmp", pala);// Carico file immagine
diverso

// determina la qualità del dettaglio della texture 2 in pixel inferiore alla
//prima texture
wa = 64;
hb = 64;

// creo il l'area bitmap, che sara tanto più dettagliata quanto più saranno alti
// 'wa' e 'hb'. Pena perdita di velocità nell'esecuzione video.
bmpb = create_bitmap (wa, hb);

// ridimensiono e centro l'immagine contenuta in 'bmpa' e la traferisco in 'bmpb'
stretch_blit (bmpa, bmpb, 0, 0, bmpa->w, bmpa->h, 0, 0, wa, hb);
destroy_bitmap (bmpa); // pulisco la variabile dell'immagine texture

// parametri di visualizzazione gl della texture sul poligono
allegro_gl_begin();
glDisable (GL_DITHER); // disattivo il dithering delle texture (grana)
glEnable (GL_TEXTURE_2D); // attivo la texture 2d
glTexEnvi (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
glHint (GL_PERSPECTIVE_CORRECTION_HINT, GL_FASTEST);

// sposto il contenuto di 'bmpb' nella variabile 'tex2'
tex2 = allegro_gl_make_texture (bmpb);
destroy_bitmap (bmpb); // elimino il contenuto di 'bmpb'
}

int main ()
{
allegro_init();
install_allegro_gl();
```

```
allegro_gl_clear_settings();
allegro_gl_set (AGL_COLOR_DEPTH, 32);
allegro_gl_set (AGL_Z_DEPTH, 16);
allegro_gl_set (AGL_FULLSCREEN, TRUE);
allegro_gl_set (AGL_DOUBLEBUFFER, 1);
allegro_gl_set (AGL_SUGGEST, AGL_COLOR_DEPTH | AGL_Z_DEPTH | AGL_DOUBLEBUFFER
| AGL_FULLSCREEN);

// Attiva l'antialiasing (...,1). Disattiva l'antialiasing (...,0)
allegro_gl_set (AGL_RENDERMETHOD, 1);

// Profondità colore Palette
set_color_depth (32);

//Determina la modalità OPENGL e la risoluzione
set_gfx_mode(GFX_OPENGL, 640, 480, 0, 0);
install_keyboard();
install_timer();
allegro_gl_begin();
glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);

//Colore del fondale: (rosso,verde,blu,alfa) - max 1.0 min 0.0
glClearColor (0.0f, 0.5f, 0.0f, 0.5f);

//(GL_SMOOTH): Crea una migliore qualità di rendering(smussa i colori creando tonalità)
//es: (GL_FLAT): Il colore è il medesimo nella stessa faccia...
glShadeModel (GL_SMOOTH);

//INDICA AL COMPUTER CHE GLI OGGETTI SIANO DISEGNATI NEL GIUSTO ORDINE
glEnable (GL_DEPTH_TEST);

//attiva il miglior livello di correzione prospettica(calo velocità)
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

// luce bianca diffusa (red,green,blue,alpha)
GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};

// posizione infinita luce (x,y,z,infinita -0.0=si-1.0=no-)
GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
// Assegna alla luce il colore stabilito
glLightfv (GL_LIGHT0, GL_DIFFUSE, light_diffuse);

// Assegna alla luce la posizione stabilita
glLightfv (GL_LIGHT0, GL_POSITION, light_position);

// attiva una luce specifica 'LIGHT0'
glEnable (GL_LIGHT0);

// Attiva l'illuminazione
glEnable (GL_LIGHTING);
allegro_gl_end();

do {
camera();
oggetto();
```

```
rotazione();
setup_textures(); // procedura texture 1
setup_textures2(); // procedura texture 2
} while (!key[KEY_ESC]);
}
END_OF_MAIN();
```

Commento al codice:
Costruire un poligono con il medesimo aspetto per ogni faccia è sicuramente limitante per l'utente, oltre che poco realistico! Ecco perché nel listato appena esposto si possono applicare ben due superfici diverse alle facce del solito cubo. Naturalmente è possibile applicare un numero elevatissimo di superfici ad un qualsiasi poligono. Una caratterista a cui si deve prestare la massima attenzione utilizzando le textures è la loro risoluzione, più questa è elevata più rallenta l'esecuzione del programma da parte del computer.

Oggetti solidi con texture e blend (trasparenza).



```
/* CODICE ESEMPLIFICATIVO SULLA GESTIONE DELLE OPENGL CON ALLEGRO
SCRITTO INTERAMENTE DA BARTOLOMEO DAVIDE BERTINETTO

WWW.BERTINETTOBARTOLOMEODAVIDE.IT

Listato: 3dn08trasparenza.c
*/

#include <allegro.h>
#include "alleggl.h"

void camera (void);
void oggetto (void);
void rotazione (void);

// variabile azzerata che ruotera di n gradi il quadrato
int rquad=0;

// variabile alla quale verra assegnata la texture
GLuint tex;

// procedura di creazione e posizionamento della camera nella scena e prospettiva
```

```
void camera (void)
{

// aggiunge la prospettiva alla scena
glMatrixMode (GL_PROJECTION);

// comando simile ad un reset di conferma
glLoadIdentity ();

// indica le proporzioni della finestra e dell'angolo prospettico
// inoltre l'ultimo valore indica la profondità massima di visione della camera.
// Tutti gli oggetti oltre questo valore non saranno visti nella scena.
glFrustum (-1.0, 1.0, -1.0, 1.0, 1.0, 40.0);

// indica alla scena la matrice di visualizzazione sulle caratteristriche
// del modello
glMatrixMode (GL_MODELVIEW);

// Cancella il buffer del colore RGB ed il depth buffer(pulisce la scena)
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// comando simile ad un reset di conferma
glLoadIdentity();

//posiziona la camera
glTranslatef (0, 0, -0.5);

// orienta la camera nei rispettivi assi...
glRotatef (0, 1, 0, 0); //angolo x
glRotatef (0, 0, 1, 0); //angolo y
glRotatef (0, 0, 0, 1); //angolo z

// salva la posizione e rotazione della camera nella matrice scenica
glPushMatrix();

}

// posizione, orientamento, colore e creazione dell'oggetto
void oggetto (void)
{

// aggiunto per azzerare la scena ad ogni ciclo. Separatore
glLoadIdentity();

// Posizione nuovo poligono
glTranslatef(0.0f,0.0f,-3.0f);
glRotatef(rquad,0.0f,1.0f,0.0f); // Ruota di n gradi nell'asse Y

/* questo comano assegna una determinata texture ( indicata dalla variabile 'tex')
a tutte le faccie dell'aggetto racchiuse tra 'glBegin(GL_QUADS);' e 'glEnd();'.
Quindi se si vuole assegnare una nuova texture ad un'altra faccia del medesimo
oggetto si dovra aprire un altro 'glBegin(GL_QUADS);' e 'glEnd();'. */
glBindTexture (GL_TEXTURE_2D, tex);
glBegin(GL_QUADS);

// Faccia frontale
```

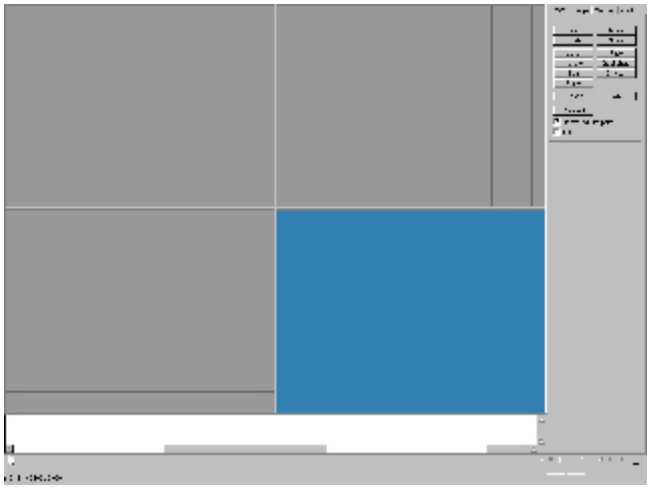
```
/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);

// Faccia posteriore
/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);

// Faccia in alto
/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, 1.0f, 1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, 1.0f, 1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);

// Faccia in basso
/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);

// Faccia di destra
/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);

// Faccia di sinistra
/* Creo la faccia del cubo ed applico la texture su di essa */
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
glEnd();
glFlush();
allegro_gl_flip();
}

// Procedura di rotazione
void rotazione (void) {
rquad=rquad+1; // Incrementa la variabile di 1 unità di grado ogni ciclo.
if (rquad>359) rquad=0; // se è maggiore di 359 ritorna a 0. Giro completo
}

// procedura che carica un texture
void setup_textures (void)
{
PALETTE pal; // variabile di palette
BITMAP *bmp, *bmp2; // carico le variabile bitmap
```

```
int w, h; // variabili di altezza e larghezza immagine
bmp = load_bitmap ("www.bertinettobartolomeodavide.it0001.bmp", pal); // Carico file immagine

// determina la qualità del dettaglio della texture
w = 128;
h = 128;

// creo il l'area bitmap, che sara tanto più dettagliata quanto più saranno alti
// 'w' e 'h'. Pena perdita di velocità nell'esecuzione video.
bmp2 = create_bitmap (w, h);

// ridimensiono e centro l'immagine contenuta in 'bmp' e la trasferisco in 'bmp2'
stretch_blit (bmp, bmp2, 0, 0, bmp->w, bmp->h, 0, 0, w, h);
destroy_bitmap (bmp); // pulisco la variabile dell'immagine texture

// parametri di visualizzazione gl della texture sul poligono
allegro_gl_begin();
glDisable (GL_DITHER); // disattivo il dithering delle texture (grana)
glEnable (GL_TEXTURE_2D); // attivo la texture 2d
glTexEnvi (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
glHint (GL_PERSPECTIVE_CORRECTION_HINT, GL_FASTEST);

// sposto il contenuto di 'bmp2' nella variabile 'tex'
tex = allegro_gl_make_texture (bmp2);
destroy_bitmap (bmp2); // elimino il contenuto di 'bmp2'
}

int main ()
{
allegro_init();
install_allegro_gl();
allegro_gl_clear_settings();
allegro_gl_set (AGL_COLOR_DEPTH, 32);
allegro_gl_set (AGL_Z_DEPTH, 16);
allegro_gl_set (AGL_FULLSCREEN, TRUE);
allegro_gl_set (AGL_DOUBLEBUFFER, 1);
allegro_gl_set (AGL_SUGGEST, AGL_COLOR_DEPTH | AGL_Z_DEPTH | AGL_DOUBLEBUFFER
| AGL_FULLSCREEN);

// Attiva l'antialiasing (...,1). Disattiva l'antialiasing (...,0)
allegro_gl_set (AGL_RENDERMETHOD, 1);

// Profondità colore Palette
set_color_depth (32);

//Determina la modalità OPENGL e la risoluzione
set_gfx_mode(GFX_OPENGL, 640, 480, 0, 0);
install_keyboard();
install_timer();
allegro_gl_begin();
glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);

// DETERMINA IL LIVELLO DI 'LUMINOSITA' E DI TRASPARENZA (ALPHA)
glColor4f(1.0f,1.0f,1.0f,0.5f);
```

```
// VISSUALIZZA CORRETTAMENTE L'OGGETO TRASPARENTE ANCHE SENZA IL COMANDO
// 'GL_DEPTH_TEST' ATTIVATO.
glBlendFunc(GL_SRC_ALPHA,GL_ONE);

//(GL_SMOOTH): Crea una migliore qualità di rendering(smussa i colori creando tonalità)
//es: (GL_FLAT): Il colore è il medesimo nella stessa faccia...
glShadeModel (GL_SMOOTH);

// ATTIVA L'EFFETTO DELLA TRSPARENZA
glEnable(GL_BLEND);

//INDICA AL COMPUTER CHE GLI OGGETTI SIANO DISEGNATI NEL GIUSTO ORDINE
// QUESTO COMANDO DEVE ESSERE DISABILITATO QUANDO SI USANO LE TRASPARENZE
glDisable (GL_DEPTH_TEST);

//attiva il miglior livello di correzione prospettica(calo velocità)
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

// luce bianca diffusa (red,green,blue,alpha)
GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};

// posizione infinita luce (x,y,z,infinita -0.0=si-1.0=no-)
GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};

// Assegna alla luce il colore stabilito
glLightfv (GL_LIGHT0, GL_DIFFUSE, light_diffuse);

// Assegna alla luce la posizione stabilita
glLightfv (GL_LIGHT0, GL_POSITION, light_position);

// attiva una luce specifica 'LIGHT0'
glEnable (GL_LIGHT0);

// Attiva l'illuminazione
glEnable (GL_LIGHTING);

allegro_gl_end();

do {
camera();
oggetto();
rotazione();
setup_textures();
} while (!key[KEY_ESC]);
}
END_OF_MAIN();
```

Commento al codice:
Mano a mano che proseguiamo con la programmazione 3D aumentano gli effetti di realismo, uno di questi è proprio la trasparenza degli oggetti.

IMPORTARE UN QUALUNQUE OGGETTO 3D PER MEZZO DEL PROGRAMMA DI MODELLAZIONE 3D MILKSHAPE 3D E LA PLUGIN msOpenGLCppExporter v.1.0

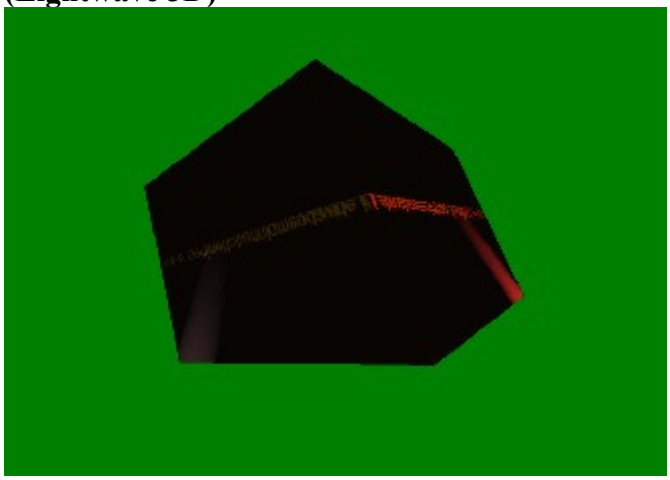


Grazie al software Milkshape 3d (ver.1.7.8) è possibile, con un semplicissimo procedimento, convertire un qualunque file 3d (lwo, 3ds o altro) in un file header del C/C++. Per ottenere questo risultato è sufficiente appoggiarsi ad una piccolissima plugin dedicata.
In un battibaleno avremo i nostri file 3d preferiti subito disponibili per i lavori OPENGL scritti in C/C++.

Seguendo questo breve procedimento, impareremo a “trasformare” un oggetto 3d in un formato compatibile con il linguaggio C/C++ grazie a MS3D:
-Installiamo il software Milkshape 3d (faccio notare che è permesso l'uso gratuito per 30 giorni).
-Scompattiamo il file contenente la plugin msOpenGLCppExporter v.1.0 in 'C:\Programmi\MilkShape 3D 1.7.8'
-Lanciamo Milkshape 3d.
-Clicchiamo con il puntatore sul menù 'File' poi su 'Import' e su 'lightwave LWO...' (compatibile con la vostra versione!)
-Selezionate il file di esempio presente in questo tutorial 'cubo.lwo'
-Dovrebbe comparirvi una finestra. Lasciate tutto com'è. Quindi confermate cliccando su 'OK'
-Ora è presente il cubo 3d in tutte e quattro le finestre di MS3D.
-Adesso dobbiamo mantenere le textures assegnate precedentemente con Lightwave, in questo caso. Perciò cliccate 'Group' in alto a destra.
-Selezioniamo il nome del materiale e quindi la voce 'cubo <Mat.:cubo>' e premiamo 'Select' subito sotto.
-A questo punto dovrebbero essersi colorate di rosso le facce di quel materiale presenti sull'oggetto 3d.
-Premiamo il menù 'tools' e selezioniamo 'Tile texture mapper'.
-E' comparsa una finestra con la voce 'Planar' o 'Cubic' (dipende dalla disposizione che vogliamo dare alla nostra texture), noi scegliamo 'Cubic'. Confermiamo con 'OK'.
-Siamo pronti per salvare il nostro oggetto per usarlo con le openGL. Ritorniamo sul menù 'File' e poi su 'Export', quindi scegliamo la voce 'OpenGL C++ (.h)...'
-Salviamolo nella cartella dove è presente il listato del programma che stiamo creando.

Adesso non vi resta che scaricare i sorgenti commentati, per capire al meglio come gestire la programmazione utilizzando un oggetto 3D importato contenente una sola texture.

Esempio di importazione oggetto cubico da un programma di modellazione tridimensionale (Lightwave 3D)



```
/* listato tutorial - cubo.c - e - cubo.h -
Creato da Bertinetto Bartolomeo Davide il 28/04/2006
infocalimero@libero.it

Lo scopo è di spiegare la gestione delle openGL con
ALLEGRO.H e ALLEGGL.H.
http://allegrogl.sf.net/
http://alleg.sf.net/

L'importazione e la conversione del file 3d sono rese
possibili grazie alla libreria di:
MILKSHAPE 3D - msOpenGLCppExporter.dll.
www.milkshape3d.com
bob_nemeth@hotmail.com

Listato: 3dn09oggettocuboideimportato.c
*/


#include <allegro.h>

#include "alleggl.h"
#include "cubo.h" // file dell'oggetto 3d
// "Object" data structure

int i; // variabile intera per l'array di caricamento dell'oggetto
GLuint tex; // variabile di assegnazione texture all'oggetto

// procedura di definizione variabili contenute nel file.h dell'oggetto 3d
typedef struct _obj {
Point3 *vertices;
long *v_idx;
Point3 *normals;
long *n_idx;
Point2 *uvs;
```

```
int num_faces;
} Obj;


void camera (void);
void oggetto (void);

Obj cuboide; // nome oggetto

// procedura di creazione e posizionamento della camera nella scena e prospettiva
void camera (void)
{
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-1.0, 1.0, -1.0, 1.0, 1.0, 10000);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}

// posizione, orientamento, colore e creazione dell'oggetto
void oggetto (void)
{
static GLfloat angle1 = 0.0;
angle1++; // contatore gradi di rotazione
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
glLoadIdentity();

// posizionamento oggetto (x,y,z)
glTranslatef( 0.0, 0.0, -7 );

// rotazione oggetto (angolo,x=1.0,y=1.0,z=1.0)
glRotatef( angle1, 0.0, 1.0, 0.0 );
glRotatef( angle1 / 3.0, 1.0, 0.0, 0.0 );

// Assegno la texture all'oggetto
glBindTexture (GL_TEXTURE_2D, tex);

// Indico che l'oggetto è formato da triangoli
glBegin( GL_TRIANGLES );

// Do inizio all'array di lettura delle coordinate dei vertici, facce e texture
for( i=0; i<cuboide.num_faces; i++)
{

// coordinate texture (x,y)
glTexCoord2f( cuboide.uvs[cuboide.v_idx[i]].x, cuboide.uvs[cuboide.v_idx[i]].y );

// coordinate poligono
glNormal3f( cuboide.normals[cuboide.n_idx[i]].x,
cuboide.normals[cuboide.n_idx[i]].y,
cuboide.normals[cuboide.n_idx[i]].z );
glVertex3f(cuboide.vertices[cuboide.v_idx[i]].x,
cuboide.vertices[cuboide.v_idx[i]].y,
cuboide.vertices[cuboide.v_idx[i]].z );
}
glEnd();
```

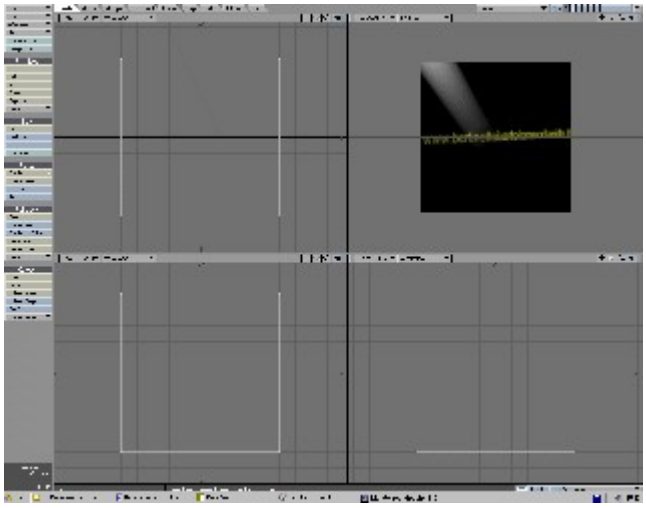
```
glFlush();
allegro_gl_flip();
}


void texture (void)
{
PALETTE pal; // variabile palette
BITMAP *bmp, *bmp2; // variabili dei bitmap
int w, h; // variabili di dimensione

// carico l'immagine bitmap e assegno la palette
bmp = load_bitmap ("www.bertinettobartolomeodavide.it0001.bmp", pal);

// larghezza ed altezza in pixel di qualità immagine
w = 128;
h = 128;

// matrice buffer bitmap dove sarà contenuta l'immagine ridimensionata
bmp2 = create_bitmap (w, h);

// ridimensiono il bitmap e lo trasferisco da 'bmp' a 'bmp2'
stretch_blit (bmp, bmp2, 0, 0, bmp->w, bmp->h, 0, 0, w, h);

// libero la ram da 'bmp'
destroy_bitmap (bmp);

// iniziano i comandi allegro gl per la texture
allegro_gl_begin();

// Disabilita la texture con una sorta di granatura
glDisable (GL_DITHER);

// Attiva la texture 2d
glEnable (GL_TEXTURE_2D);

/* Determina la visualizzazione della texture a contatto con la luce
- GL_BLEND Il Colore della texture viene moltiplicato per il colore del pixel e infine moltiplicato per
una costante.
- GL_DECAL Il Colore della texture sostituisce quello del colore
- GL_MODULATE Il colore della Texture viene moltiplicato per quello del pixel.
*/
glTexEnvi (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

/* Questa riga indica al computer che il dettaglio della texture deve essere alto
nei momenti in cui si trova vicino alla camera e basso quando la texture è ad una
distanza maggiore */
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR_MIPMAP_NEARE
ST);

// assegna il bitmap appena ridimensionato ' bmp2' alla variabile di texture 'tex'
tex = allegro_gl_make_texture (bmp2);

// libero la ram da 'bmp2'
destroy_bitmap (bmp2);
}
```

```
int main ()
{
allegro_init();
install_allegro_gl();
allegro_gl_clear_settings();
allegro_gl_set (AGL_COLOR_DEPTH, 32);
allegro_gl_set (AGL_Z_DEPTH, 16);
allegro_gl_set (AGL_FULLSCREEN, TRUE);
allegro_gl_set (AGL_DOUBLEBUFFER, 1);
allegro_gl_set (AGL_SUGGEST, AGL_COLOR_DEPTH | AGL_Z_DEPTH | AGL_DOUBLEBUFFER
| AGL_FULLSCREEN);

// Attiva l'antialiasing (...,1). Disattiva l'antialiasing (...,0)
allegro_gl_set (AGL_RENDERMETHOD, 1);

// Profondità colore Palette
set_color_depth (32);

//Determina la modalità OPENGL e la risoluzione
set_gfx_mode(GFX_OPENGL, 640, 480, 0, 0);
install_keyboard();
install_timer();
allegro_gl_begin();
allegro_gl_use_mipmapping(TRUE);
glEnable(GL_COLOR_MATERIAL);

//Colore del fondale: (rosso,verde,blu,alfa) - max 1.0 min 0.0
glClearColor (0.0f, 0.5f, 0.0f, 0.5f);

//(GL_SMOOTH): Crea una migliore qualità di rendering(smussa i colori creando tonalità)
//es: (GL_FLAT): Il colore è il medesimo nella stessa faccia...
glShadeModel (GL_SMOOTH);

//INDICA AL COMPUTER CHE GLI OGGETTI SIAMO DISEGNATI NEL GIUSTO ORDINE
glEnable (GL_DEPTH_TEST);

//attiva il miglior livello di correzione prospettica(calo velocità)
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

// luce rossa diffusa (red,green,blue,alpha)
GLfloat light_diffuse[] = {1.0, 0.0, 0.0, 1.0};

// posizione infinita luce (x,y,z,infinita -0.0=si-1.0=no-)
GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};

// Assegna alla luce il colore stabilito
glLightfv (GL_LIGHT0, GL_DIFFUSE, light_diffuse);
// Assegna alla luce la posizione stabilita
glLightfv (GL_LIGHT0, GL_POSITION, light_position);

// attiva una luce specifica 'LIGHT0'
glEnable (GL_LIGHT0);

// Attiva l'illuminazione
glEnable (GL_LIGHTING);
```

```
// conversioni di variabili per la creazione del oggetto 3d
cuboide.vertices = cubo_vertex;
cuboide.v_idx = cubo_vidx;
cuboide.normals = cubo_normal;
cuboide.n_idx = cubo_nidx;
cuboide.uvs = cubo_uv;
cuboide.num_faces = (sizeof(cubo_vidx)/sizeof(long));
allegro_gl_end();

// Ciclo in cui vengono caricate le varie procedure
do {
camera();
oggetto();
texture();
} while (!key[KEY_ESC]);
}
END_OF_MAIN();
```

Commento al listato:
Perché il tutto funzioni è importante analizzare e copiare nella stessa cartella del listato 'cubo.c' appena esposto, così come il codice che segue 'cubo.h'. Infatti l'header spiegato subito dopo rappresenta il file contenete l'oggetto 3D vero e proprio.

FILE DEL MODELLO 3D CONVERTITO IN 'CUBO.H':

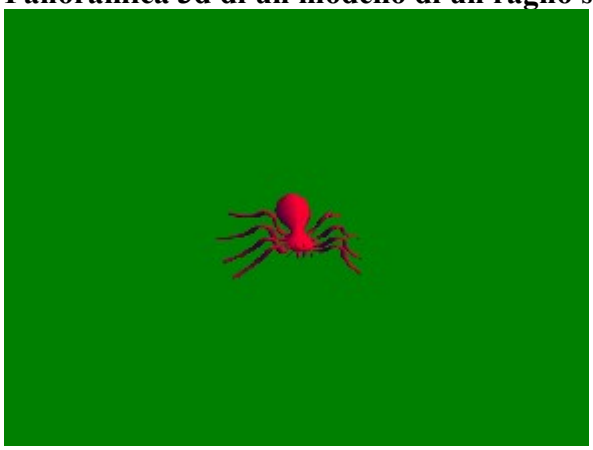


```
// OpenGL C++

typedef struct _point2 {
double x,y;
} Point2;

typedef struct _point3 {
double x,y,z;
} Point3;

// ********** MESH DATA **********

// --- Mesh: cubo ---
```

```
Point3 cubo_vertex[] = {
{-2.500000, 2.500000, 2.500000},
{-2.500000, -2.500000, -2.500000},
{-2.500000, -2.500000, 2.500000},
{-2.500000, 2.500000, -2.500000},
{2.500000, -2.500000, 2.500000},
{-2.500000, -2.500000, 2.500000},
{2.500000, -2.500000, -2.500000},
{2.500000, 2.500000, -2.500000},
{2.500000, 2.500000, 2.500000},
{-2.500000, 2.500000, -2.500000},
{-2.500000, 2.500000, 2.500000},
{2.500000, 2.500000, -2.500000},
{2.500000, -2.500000, 2.500000},
{-2.500000, -2.500000, 2.500000},
{2.500000, -2.500000, -2.500000},
{2.500000, 2.500000, -2.500000}
};

Point2 cubo_uv[] = {
{1.000000, 1.000000},
{0.000000, 0.000000},
{1.000000, 0.000000},
{0.000000, 1.000000},
{1.000000, 1.000000},
{0.000000, 1.000000},
{1.000000, 0.000000},
{1.000000, 1.000000},
{1.000000, 1.000000},
{0.000000, 0.000000},
{0.000000, 1.000000},
{1.000000, 0.000000},
{1.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 1.000000}
};

Point3 cubo_normal[] = {
{-1.000000, 0.000000, 0.000000},
{0.000000, -1.000000, 0.000000},
{0.000000, 0.000000, -1.000000},
{0.000000, 1.000000, 0.000000},
{0.000000, 0.000000, 1.000000},
{1.000000, 0.000000, 0.000000}
};

long cubo_vidx[] = {
0, 1, 2,
0, 3, 1,
1, 4, 5,
1, 6, 4,
3, 6, 1,
3, 7, 6,
8, 9, 10,
8, 11, 9,
```

```
12, 10, 13,
12, 8, 10,
14, 8, 12,
14, 15, 8
};

long cubo_nidx[] = {
0, 0, 0,
0, 0, 0,
1, 1, 1,
1, 1, 1,
2, 2, 2,
2, 2, 2,
3, 3, 3,
3, 3, 3,
4, 4, 4,
4, 4, 4,
5, 5, 5,
5, 5, 5
};
```

Panoramica 3d di un modello di un ragno senza textures



```
/* CODICE ESEMPLIFICATIVO SULLA GESTIONE DELLE OPENGL CON ALLEGRO
SCRITTO INTERAMENTE DA BARTOLOMEO DAVIDE BERTINETTO

WWW.BERTINETTOBARTOLOMEODAVIDE.IT

Listato: 3dn10ragno3dnotextures.c

*/

#include <allegro.h>
#include "alleggl.h"

#include "ragno.h"

// "Object" data structure
int i;
```

```
typedef struct _obj {
Point3 *vertices;
long *v_idx;
Point3 *normals;
long *n_idx;
Point2 *uvs;
int num_faces;
} Obj;

void reshapeFunc(int w, int h);
void displayFunc();
void camera (void);
void oggetto (void);

Obj ragno;

// procedura di creazione e posizionamento della camera nella scena e prospettiva
void camera (void)
{
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-1.0, 1.0, -1.0, 1.0, 1.0, 10000);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}

// posizione, orientamento, colore e creazione dell'oggetto
void oggetto (void)
{
glColor3f(1.0, 0.0, 1.0);
static GLfloat angle1 = 0.0;
angle1++;
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
glLoadIdentity();
glTranslatef( 0.0, 0.0, -100 );
glRotatef( angle1, 0.0, 1.0, 0.0 );
glRotatef( angle1 / 3.0, 1.0, 0.0, 0.0 );
glBegin( GL_TRIANGLES );
for( i=0; i<ragno.num_faces; i++)
{
glNormal3f( ragno.normals[ragno.n_idx[i]].x,
ragno.normals[ragno.n_idx[i]].y,
ragno.normals[ragno.n_idx[i]].z );
glVertex3f(ragno.vertices[ragno.v_idx[i]].x, ragno.vertices[ragno.v_idx[i]].y,
ragno.vertices[ragno.v_idx[i]].z );
}
glEnd();
glFlush();
allegro_gl_flip();
}

int main ()
{
allegro_init();
install_allegro_gl();
allegro_gl_clear_settings();
```

```
allegro_gl_set (AGL_COLOR_DEPTH, 32);
allegro_gl_set (AGL_Z_DEPTH, 16);
allegro_gl_set (AGL_FULLSCREEN, TRUE);
allegro_gl_set (AGL_DOUBLEBUFFER, 1);
allegro_gl_set (AGL_SUGGEST, AGL_COLOR_DEPTH | AGL_Z_DEPTH | AGL_DOUBLEBUFFER
| AGL_FULLSCREEN);

// Attiva l'antialiasing (...,1). Disattiva l'antialiasing (...,0)
allegro_gl_set (AGL_RENDERMETHOD, 1);

// Profondità colore Palette
set_color_depth (32);

//Determina la modalità OPENGL e la risoluzione
set_gfx_mode(GFX_OPENGL, 640, 480, 0, 0);
install_keyboard();
install_timer();
allegro_gl_begin();
glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);

//Colore del fondale: (rosso,verde,blu,alfa) - max 1.0 min 0.0
glClearColor (0.0f, 0.5f, 0.0f, 0.5f);
//(GL_SMOOTH): Crea una migliore qualità di rendering(smussa i colori creando tonalità)
//es: (GL_FLAT): Il colore è il medesimo nella stessa faccia...
glShadeModel (GL_SMOOTH);

//INDICA AL COMPUTER CHE GLI OGGETTI SIAMO DISEGNATI NEL GIUSTO ORDINE
glEnable (GL_DEPTH_TEST);

//attiva il miglior livello di correzione prospettica(calo velocità)
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
GLfloat light_diffuse[] = {1.0, 0.0, 0.0, 1.0}; // red diffuse light
GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0}; // infinite light position
glLightfv (GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv (GL_LIGHT0, GL_POSITION, light_position);
glEnable (GL_LIGHT0);
glEnable (GL_LIGHTING);

//Abilita la luce 'GL_LIGHT1'
glEnable(GL_LIGHT1);
ragno.vertices = Group01_vertex;
ragno.v_idx = Group01_vidx;
ragno.normals = Group01_normal;
ragno.n_idx = Group01_nidx;
ragno.uvs = Group01_uv;
ragno.num_faces = (sizeof(Group01_vidx)/sizeof(long));
allegro_gl_end();

do {
camera();
oggetto();
} while (!key[KEY_ESC]);
}
END_OF_MAIN();
```

Commento al listato:
Questo rappresenta l'esempio concreto sulla possibilità di importare un oggetto complesso gestendolo con le openGL e la programmazione C. Per ovvie ragioni di lunghezza non compare il contenuto dell'header.h contenente il modello 3d del ragno.

Panoramica 3d di un modello di un ragno con textures

```
/* email: contatto@bertinettobartolomeodavide.it - infocalimero@libero.it
www.bertinettobartolomeodavide.it
Listato: 3dn11ragno3dtextures.c

*/


#include <allegro.h>
#include "alleggl.h"

#include "ragnomultitexture.h"

int i;
GLuint tex1; // variabile di assegnazione texture 1 all'oggetto
GLuint tex2; // variabile di assegnazione texture 2 all'oggetto
GLuint tex3; // variabile di assegnazione texture 3 all'oggetto
GLuint tex4; // variabile di assegnazione texture 4 all'oggetto

// struttura dati dell' oggetto 'obj'

typedef struct _obj {
Point3 *vertices;
long *v_idx;
Point3 *normals;
long *n_idx;
Point2 *uvs;
int num_faces;
} Obj;

void reshapeFunc(int w, int h);
void displayFunc();
void camera (void);
void oggetto (void);

// oggetti di caricamento poligoni per nome faccia
```

```
Obj ragnomultitexture;
Obj ragnomultitexture2;
Obj ragnomultitexture3;
Obj ragnomultitexture4;

// procedura di creazione e posizionamento della camera nella scena e prospettiva

void camera (void)
{
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-1.0, 1.0, -1.0, 1.0, 1.0, 10000);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}

// posizione, orientamento, texture e creazione dell'oggetto

void oggetto (void)
{
static GLfloat angle1 = 0.0;
angle1++;
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
glLoadIdentity();
glTranslatef( 0.0, 0.0, -50 );
glRotatef( angle1, 0.0, 1.0, 0.0 );
glRotatef( angle1 / 3.0, 1.0, 0.0, 0.0 );

// Assegno la texture all'oggetto
glBindTexture (GL_TEXTURE_2D, tex1);

// Caricamento faccie chiamate Spider_Belly
glBegin( GL_TRIANGLES );
for( i=0; i<ragnomultitexture.num_faces; i++)
{
glTexCoord2f( ragnomultitexture.uvs[ragnomultitexture.v_idx[i]].x,
ragnomultitexture.uvs[ragnomultitexture.v_idx[i]].y );
glNormal3f( ragnomultitexture.normals[ragnomultitexture.n_idx[i]].x,
ragnomultitexture.normals[ragnomultitexture.n_idx[i]].y,
ragnomultitexture.normals[ragnomultitexture.n_idx[i]].z );
glVertex3f(ragnomultitexture.vertices[ragnomultitexture.v_idx[i]].x,
ragnomultitexture.vertices[ragnomultitexture.v_idx[i]].y,
ragnomultitexture.vertices[ragnomultitexture.v_idx[i]].z );
}
glEnd();

// Assegno la texture all'oggetto
glBindTexture (GL_TEXTURE_2D, tex2);

// Caricamento facce chiamate Spider_Body
glBegin( GL_TRIANGLES );
for( i=0; i<ragnomultitexture2.num_faces; i++)
{
glTexCoord2f( ragnomultitexture2.uvs[ragnomultitexture2.v_idx[i]].x,
ragnomultitexture2.uvs[ragnomultitexture2.v_idx[i]].y );
glNormal3f( ragnomultitexture2.normals[ragnomultitexture2.n_idx[i]].x,
```

```
ragnomultitexture2.normals[ragnomultitexture2.n_idx[i]].y,
ragnomultitexture2.normals[ragnomultitexture2.n_idx[i]].z );
glVertex3f(ragnomultitexture2.vertices[ragnomultitexture2.v_idx[i]].x,
ragnomultitexture2.vertices[ragnomultitexture2.v_idx[i]].y,
ragnomultitexture2.vertices[ragnomultitexture2.v_idx[i]].z );
}
glEnd();

// Assegno la texture all'oggetto
glBindTexture (GL_TEXTURE_2D, tex3);

// Caricamento faccie chiamate Spider_Eyes
glBegin( GL_TRIANGLES );
for( i=0; i<ragnomultitexture3.num_faces; i++)
{
glTexCoord2f( ragnomultitexture3.uvs[ragnomultitexture3.v_idx[i]].x,
ragnomultitexture3.uvs[ragnomultitexture3.v_idx[i]].y );
glNormal3f( ragnomultitexture3.normals[ragnomultitexture3.n_idx[i]].x,
ragnomultitexture3.normals[ragnomultitexture3.n_idx[i]].y,
ragnomultitexture3.normals[ragnomultitexture3.n_idx[i]].z );
glVertex3f(ragnomultitexture3.vertices[ragnomultitexture3.v_idx[i]].x,
ragnomultitexture3.vertices[ragnomultitexture3.v_idx[i]].y,
ragnomultitexture3.vertices[ragnomultitexture3.v_idx[i]].z );
}
glEnd();

// Assegno la texture all'oggetto
glBindTexture (GL_TEXTURE_2D, tex4);

// Caricamento facce chiamate Spider_Legs
glBegin( GL_TRIANGLES );
for( i=0; i<ragnomultitexture4.num_faces; i++)
{
glTexCoord2f( ragnomultitexture4.uvs[ragnomultitexture4.v_idx[i]].x,
ragnomultitexture4.uvs[ragnomultitexture4.v_idx[i]].y );
glNormal3f( ragnomultitexture4.normals[ragnomultitexture4.n_idx[i]].x,
ragnomultitexture4.normals[ragnomultitexture4.n_idx[i]].y,
ragnomultitexture4.normals[ragnomultitexture4.n_idx[i]].z );
glVertex3f(ragnomultitexture4.vertices[ragnomultitexture4.v_idx[i]].x,
ragnomultitexture4.vertices[ragnomultitexture4.v_idx[i]].y,
ragnomultitexture4.vertices[ragnomultitexture4.v_idx[i]].z );
}
glEnd();
glFlush();
allegro_gl_flip();
}

// procedura di caricamento texture 1 da disco
void texture (void)
{
PALETTE pal; // variabile palette
BITMAP *bmp, *bmp2; // variabili dei bitmap
int w, h; // variabili di dimensione

// carico l'immagine bitmap e assegno la palette
bmp = load_bitmap ("tex1.bmp", pal);
```

```
// larghezza ed altezza in pixel di qualità immagine
w = 128;
h = 128;
// matrice buffer bitmap dove sarà contenuta l'immagine ridimensionata
bmp2 = create_bitmap (w, h);

// ridimensiono il bitmap e lo trasferisco da 'bmp' a 'bmp2'
stretch_blit (bmp, bmp2, 0, 0, bmp->w, bmp->h, 0, 0, w, h);

// libero la ram da 'bmp'
destroy_bitmap (bmp);

// iniziano i comandi allegro gl per la texture
allegro_gl_begin();

// Disabilita la texture con una sorta di granatura
glDisable (GL_DITHER);

// Attiva la texture 2d
glEnable (GL_TEXTURE_2D);

/* Determina la visualizzazione della texture a contatto con la luce
- GL_BLEND Il Colore della texture viene moltiplicato per il colore del pixel e infine moltiplicato per una costante.
- GL_DECAL Il Colore della texture sostituisce quello del colore
- GL_MODULATE Il colore della Texture viene moltiplicato per quello del pixel.
*/
glTexEnvi (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

/* Questa riga indica al computer che il dettaglio della texture deve essere alto
nei momenti in cui si trova vicino alla camera e basso quando la texture è ad una
distanza maggiore */
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR_MIPMAP_NEAREST);

// assegna il bitmap appena ridimensionato ' bmp2' alla variabile di texture 'tex'
tex1 = allegro_gl_make_texture (bmp2);

// libero la ram da 'bmp2'
destroy_bitmap (bmp2);
}



// procedura di caricamento texture 2 da disco
void texture2 (void)
{
PALETTE palb; // variabile palette
BITMAP *bmpb, *bmpb2; // variabili dei bitmap
int wb, hb; // variabili di dimensione

// carico l'immagine bitmap e assegno la palette
bmpb = load_bitmap ("tex2.bmp", palb);
// larghezza ed altezza in pixel di qualità immagine
wb = 128;
```

```
hb = 128;

// matrice buffer bitmap dove sarà contenuta l'immagine ridimensionata
bmpb2 = create_bitmap (wb, hb);

// ridimensiono il bitmap e lo trasferisco da 'bmpb' a 'bmpb2'
stretch_blit (bmpb, bmpb2, 0, 0, bmpb->w, bmpb->h, 0, 0, wb, hb);

// libero la ram da 'bmpb'
destroy_bitmap (bmpb);

// iniziano i comandi allegro gl per la texture
allegro_gl_begin();

// Disabilita la texture con una sorta di granatura
glDisable (GL_DITHER);

// Attiva la texture 2d
glEnable (GL_TEXTURE_2D);

/* Determina la visualizzazione della texture a contatto con la luce
- GL_BLEND Il Colore della texture viene moltiplicato per il colore del pixel e infine moltiplicato per
una costante.
- GL_DECAL Il Colore della texture sostituisce quello del colore
- GL_MODULATE Il colore della Texture viene moltiplicato per quello del pixel.
*/
glTexEnvi (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

/* Questa riga indica al computer che il dettaglio della texture deve essere alto
nei momenti in cui si trova vicino alla camera e basso quando la texture è ad una
distanza maggiore */
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR_MIPMAP_NEARE
ST);

// assegna il bitmap appena ridimensionato ' bmpb2' alla variabile di texture 'tex2'
tex2 = allegro_gl_make_texture (bmpb2);

// libero la ram da 'bmpb2'
destroy_bitmap (bmpb2);
}

// procedura di caricamento texture 3 da disco

void texture3 (void)
{
PALETTE palc; // variabile palette
BITMAP *bmpc, *bmpc2; // variabili dei bitmap
int wc, hc; // variabili di dimensione
// carico l'immagine bitmap e assegno la palette
bmpc = load_bitmap ("tex3.bmp", palc);

// larghezza ed altezza in pixel di qualità immagine
wc = 128;
hc = 128;

// matrice buffer bitmap dove sarà contenuta l'immagine ridimensionata
```

```
bmpc2 = create_bitmap (wc, hc);

// ridimensiono il bitmap e lo trasferisco da 'bmpc' a 'bmpc2'
stretch_blit (bmpc, bmpc2, 0, 0, bmpc->w, bmpc->h, 0, 0, wc, hc);

// libero la ram da 'bmpc'
destroy_bitmap (bmpc);

// iniziano i comandi allegro gl per la texture
allegro_gl_begin();

// Disabilita la texture con una sorta di granatura
glDisable (GL_DITHER);

// Attiva la texture 2d
glEnable (GL_TEXTURE_2D);

/* Determina la visualizzazione della texture a contatto con la luce
- GL_BLEND Il Colore della texture viene moltiplicato per il colore del pixel e infine moltiplicato per
una costante.
- GL_DECAL Il Colore della texture sostituisce quello del colore
- GL_MODULATE Il colore della Texture viene moltiplicato per quello del pixel.
*/
glTexEnvi (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

/* Questa riga indica al computer che il dettaglio della texture deve essere alto
nei momenti in cui si trova vicino alla camera e basso quando la texture è ad una
distanza maggiore */
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR_MIPMAP_NEARE
ST);

// assegna il bitmap appena ridimensionato ' bmpc2' alla variabile di texture 'tex3'
tex3 = allegro_gl_make_texture (bmpc2);

// libero la ram da 'bmpc2'
destroy_bitmap (bmpc2);
}

// procedura di caricamento texture 4 da disco
void texture4 (void)
{
PALETTE pald; // variabile palette
BITMAP *bmpd, *bmpd2; // variabili dei bitmap
int wd, hd; // variabili di dimensione

// carico l'immagine bitmap e assegno la palette
bmpd = load_bitmap ("tex4.bmp", pald);

// larghezza ed altezza in pixel di qualità immagine
wd = 128;
hd = 128;

// matrice buffer bitmap dove sarà contenuta l'immagine ridimensionata
bmpd2 = create_bitmap (wd, hd);

// ridimensiono il bitmap e lo trasferisco da 'bmpd' a 'bmpd2'
```

```
stretch_blit (bmpd, bmpd2, 0, 0, bmpd->w, bmpd->h, 0, 0, wd, hd);

// libero la ram da 'bmpd'
destroy_bitmap (bmpd);

// iniziano i comandi allegro gl per la texture
allegro_gl_begin();

// Disabilita la texture con una sorta di granatura
glDisable (GL_DITHER);

// Attiva la texture 2d
glEnable (GL_TEXTURE_2D);

/* Determina la visualizzazione della texture a contatto con la luce
- GL_BLEND Il Colore della texture viene moltiplicato per il colore del pixel e infine moltiplicato per una costante.
- GL_DECAL Il Colore della texture sostituisce quello del colore
- GL_MODULATE Il colore della Texture viene moltiplicato per quello del pixel.
*/
glTexEnvi (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

/* Questa riga indica al computer che il dettaglio della texture deve essere alto
nei momenti in cui si trova vicino alla camera e basso quando la texture è ad una
distanza maggiore */
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR_MIPMAP_NEAREST);

// assegna il bitmap appena ridimensionato ' bmpd2' alla variabile di texture 'tex4'
tex4 = allegro_gl_make_texture (bmpd2);

// libero la ram da 'bmpd2'
destroy_bitmap (bmpd2);
}


int main ()
{
allegro_init();
install_allegro_gl();
allegro_gl_clear_settings();
allegro_gl_set (AGL_COLOR_DEPTH, 32);
allegro_gl_set (AGL_Z_DEPTH, 16);
allegro_gl_set (AGL_FULLSCREEN, TRUE);
allegro_gl_set (AGL_DOUBLEBUFFER, 1);
allegro_gl_set (AGL_SUGGEST, AGL_COLOR_DEPTH | AGL_Z_DEPTH | AGL_DOUBLEBUFFER | AGL_FULLSCREEN);

// Attiva l'antialiasing (...,1). Disattiva l'antialiasing (...,0)
allegro_gl_set (AGL_RENDERMETHOD, 1);

// Profondità colore Palette
set_color_depth (32);

// Determina la modalità OPENGL e la risoluzione
set_gfx_mode(GFX_OPENGL, 640, 480, 0, 0);
```

```
install_keyboard();
install_timer();
allegro_gl_begin();
glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);

//Colore del fondale: (rosso,verde,blu,alfa) - max 1.0 min 0.0
glClearColor (0.0f, 0.5f, 0.0f, 0.5f);

//(GL_SMOOTH): Crea una migliore qualità di rendering(smussa i colori creando tonalità)
//es: (GL_FLAT): Il colore è il medesimo nella stessa faccia...
glShadeModel (GL_SMOOTH);

//INDICA AL COMPUTER CHE GLI OGGETTI SIAMO DISEGNATI NEL GIUSTO ORDINE
glEnable (GL_DEPTH_TEST);

//attiva il miglior livello di correzione prospettica(calo velocità)
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

// Illuminazione
GLfloat light_diffuse[] = {0.5, 0.5, 0.5, 1.0}; // luce bianca diffusa
GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0}; // luce di posizione infinita
glLightfv (GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv (GL_LIGHT0, GL_POSITION, light_position);
glEnable (GL_LIGHT0);
glEnable (GL_LIGHTING);

// trasferisce il contenuto del file.h nelle variabili delle facce Spider_Belly
ragnomultitexture.vertices = Spider_Belly_vertex;
ragnomultitexture.v_idx = Spider_Belly_vidx;
ragnomultitexture.normals = Spider_Belly_normal;
ragnomultitexture.n_idx = Spider_Belly_nidx;
ragnomultitexture.uvs = Spider_Belly_uv;
ragnomultitexture.num_faces = (sizeof(Spider_Belly_vidx)/sizeof(long));

// trasferisce il contenuto del file.h nelle variabili delle facce Spider_Body
ragnomultitexture2.vertices = Spider_Body_vertex;
ragnomultitexture2.v_idx = Spider_Body_vidx;
ragnomultitexture2.normals = Spider_Body_normal;
ragnomultitexture2.n_idx = Spider_Body_nidx;
ragnomultitexture2.uvs = Spider_Body_uv;
ragnomultitexture2.num_faces = (sizeof(Spider_Body_vidx)/sizeof(long));

// trasferisce il contenuto del file.h nelle variabili delle facce Spider_Eyes
ragnomultitexture3.vertices = Spider_Eyes_vertex;
ragnomultitexture3.v_idx = Spider_Eyes_vidx;
ragnomultitexture3.normals = Spider_Eyes_normal;
ragnomultitexture3.n_idx = Spider_Eyes_nidx;
ragnomultitexture3.uvs = Spider_Eyes_uv;
ragnomultitexture3.num_faces = (sizeof(Spider_Eyes_vidx)/sizeof(long));

// trasferisce il contenuto del file.h nelle variabili delle facce Spider_Legs
ragnomultitexture4.vertices = Spider_Legs_vertex;
ragnomultitexture4.v_idx = Spider_Legs_vidx;
ragnomultitexture4.normals = Spider_Legs_normal;
ragnomultitexture4.n_idx = Spider_Legs_nidx;
```

```
ragnomultitexture4.uvs = Spider_Legs_uv;
ragnomultitexture4.num_faces = (sizeof(Spider_Legs_vidx)/sizeof(long));

allegro_gl_end();
do {
camera();
oggetto();
texture(); // richiamo la procedura di texture 1
texture2(); // richiamo la procedura di texture 2
texture3(); // richiamo la procedura di texture 3
texture4(); // richiamo la procedura di texture 4
} while (!key[KEY_ESC]);
}
END_OF_MAIN();
```

Commento al listato:
Le possibilità ora offerte della gamma di conoscenze che abbiamo acquisito nel corso dello sviluppo di tutti questi listati ci permette di costruire moderni videogames 3D... Vediamo ora, qualche altro piccolo ma necessario aspetto della programmazione.

MIGLIOR FUNZIONAMENTO SU WINDOWS 7
Windows è un ottimo sistema operativo che implementa molte nuove funzionalità. Allegro.h però necessita di alcuni aggiustamenti per funzionare al meglio su questa piattaforma. Per questo ho riscontrato alcune incompatibilità di gestione riguardanti il comando 'set_gfx_mode', tanto da richiedere alcune parole a riguardo.
Fino ad ora, in tutti i miei tentativi ho riscontrato un malfunzionamento su qualsiasi prova ad utilizzare la soluzione full_screen su Windows 7. Devo comunque sottolineare che i miei esperimenti sono stati tutti svolti su macchina virtuale dato che il mio PC è dotato di Windows XP a 64 bit e non presenta alcun malfunzionamento.
La soluzione è chiaramente quella di escludere lo svolgimento a tutto schermo delle nostre applicazioni create con allegro se pensiamo di doverle eseguire su W7. E' obbligatorio quindi passare ad un soluzione in finestra.
La modifica è davvero semplice, dato che è sufficiente sostituire la dicitura del comando 'set_gfx_mode' con 'AUTODETECT_WINDOWED' anziché solamente 'AUTODETECT' oppure 'GDI' ma con quest'ultimo l'esecuzione sarà davvero lenta. Tutto nei miei test ha funzionato bene in questo modo. Non conosco l'esistenza di problemi su Windows Vista, comunque credo che se dovessero presentarsi problemi di questo tipo la soluzione sarebbe la medesima.

Altro disguido di Windows 7 è stato nel caricamento di DevCpp. L'IDE in questione risulta abbastanza datato per W7, quindi è necessario selezionare l'icona del programma e premere il tasto destro del mouse. Quindi selezionare proprietà e poi compatibilità. Quindi impostate su Windows Xp e tutto dovrebbe risolversi al meglio. Selezionare l'opzione per tutti gli utenti è quindi premere 'OK'. Tutto sarà risolto.

Non escludo che in futuro possano rivelarsi altri problemi ma potete stare certi che se saranno gravi, cercherò di risolverli se sarà possibile. Allegro.h è un libreria troppo bella!

Per venire incontro a tutti gli utenti, viste le molte email che ricevo legate alla compilazione e link di allegro.h ho deciso di mettere online la mia versione preferita di allegro.h e che uso abitualmente. Già compilata e pronta all'uso con addirittura inclusi allegGL.h ed almp3.h. Questa vi permetterà di eseguire tutto il contenuto dei miei lavori.

Link:

http://www.bertinettobartolomeodavide.it/programmazione/ALLEGROH/compilarealegroh/indexcompiler.htm

Aggiungo che il software in questione(compilatore.exe) è stato trasformato in immagine grafica bitmap (compilatore.bmp) con una delle mie realizzazioni. Pertanto è indispensabile decriptarlo con il software 'bitmap2file' scaricabile a questo indirizzo: http://bertinettobartolomeodavide.it/file2bitmap. Per l'uso vedere videotutorial.

Codice di apertura degli archivi contenenti i sorgenti di tutto il libro e download.

Digitare sul vostro computer dal browser web che usate per la navigazione il seguente indirizzo senza sbagliare e potrete scaricare l'archivio ZIP con tutti i listati della presente guida:

http://www.bertinettobartolomeodavide.it/programmazione/ALLEGROH/LIBRO2/fileebook.zip

Grazie al codice di attivazione potrete utilizzare tutto il contenuto del libro. La scelta di inserire un codice per accedere ai sorgenti è legata al fatto di rendere inscindibili il libro ed i file che lo compongono:

imparoaprogrammare

Appendici aggiuntive...

Listato di movimento casule che potrà essere d'aiuto nella creazione di rudimentali schemi di intelligenza artificiale.

```
/* LISTATO C/C++ DI MOVIMENTO CASUALE
REALIZZATO DA BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
*/


#include <stdio.h> // LIBRERIA DI SISTEMA
#include <string.h> // LIBRERIA DI SISTEMA
#include <time.h> // LIBRERIA DI SISTEMA
#include "allegro.h" /* INCLUSA LA LIBRERIA ALLEGRO.H */

// dichiarazione variabili

BITMAP *buffer;
int x;
int y;
long casuale;
char numero[80];
int cicli;

void doppiobuffering() // INIZIO PROCEDURA DI DOPPIO BUFFERING //
```

```
{
vsync(); // SINCRONIZZAZIONE AGGIORNAMENTO SCHERMO //
blit(buffer, screen, 0, 0, 0, 0, 640, 480); // VISUALIZZA IL CONTENUTO DI BUFFER SU SCHERMO
//
clear(buffer); // PULISCE IL BUFFER //
}


int main() // INIZIO PROCEDURA DI ALLEGRO //
{


allegro_init(); // INIZIALIZZA ALLEGRO //
install_keyboard(); // INSTALLA LA TASTIERA //

set_color_depth(32); // SELEZIONA LA PROFONDITA' DI COLORE A 32 BIT //
set_gfx_mode(GFX_GDI, 640, 480, 0, 0); // SETTA LA MODALITA' GRAFICA COME GDI SU UNA
FINESTRA DI 640x480 PIXEL //



buffer = create_bitmap(640, 480); // CREA UN BUFFER BITMAP DI 640x480 PIXEL //
clear(buffer); // CANCELLA IL BUFFER //


// Posizione di partenza
x=220;
y=140;

while (!key[KEY_ESC]) { // CREA UN CICLO INFINO FINO ALLA PRESSIONE DEL TASTO
'ESC' //

// ogni 50 cicli viene generato un nuovo numero casuale

if (cicli==50) {casuale=(rand()/100)+50; cicli=0;

// controlla che il numero casuale sia compreso nel limite tra 0 e 401

if (casuale>=401) casuale=400; if (casuale<=0) casuale=0;
}

// disegna un cerchio pieno in una data posizione

circlefill(buffer, x, y, 10, makecol(255, 255, 255));

doppiobuffering(); // CARICA LA PROCEDURA DI DOPPIO BUFFERING //

// mette su schermo il numero casuale generato

sprintf(numero, "Numero casuale di direzione: %ld", (long)casuale);
textout(buffer, font, numero, 1, 400, makecol(255, 000, 255));

// Mini algoritmo di intelligenza artificiale per lo spostamento all'interno dello schermo visibile

if ((casuale>=0) && (casuale<=100)) {
y=y-3; if (y<=0) casuale=101; }
```

```
if ((casuale>=101) && (casuale<=200)) {
y=y+3; if (y>=480) casuale=0; }
if ((casuale>=201) && (casuale<=300)) {
x=x-3; if (x<=0) casuale=301; }
if ((casuale>=301) && (casuale<=400)) {
x=x+3; if (x>=640) casuale=201; }

// conta i cicli necessari alla generazione del numero casuale per la direzione di movimento

cicli++;
}


destroy_bitmap(buffer); // DISTRUGGE IL BITMAP //

allegro_exit(); // TERMINA LA LIBRERIA ALLEGRO //
}

END_OF_MAIN (); // FINE DEL PROGRAMMA //
```

ALLEGRO.H E ALLEGGL.H SU C# E PIATTAFORME FRAMEWORK.NET

La libreria allego e derivati stanno lentamente invecchiando, è un dato di fatto. Questa libreria pur mantenendo ancora oggi un altissimo grado di efficacia comincia ad essere via via non più utilizzata dai programmatori di tutto il mondo. Serve un nuovo spiraglio di vigore e forse questo è arrivato proprio con la conversione per C sharp e l'uso del framework.net.

Espongo qui in questa appendice due soluzioni per allegro.h in versione C#. Uno si accompagnerà maggiormente con la soluzione tipica della libreria per videogame 2d, l'altra per la grafica 3d OpenGL. Fermo restando che nelle presenti librerie sharp ho riscontrato diversi bug ricollegabili alle versioni beta delle presenti.

In materiale necessario è molto esiguo e tutto freeware:
framework.net 2.0(32 o 64 bit)
Visual C# express 2005 oppure Sharpdevelop 2.2
le librerie sharpallegro(ver. 0.0.4) e sharpalleggl (ver. 0.0.1)

Prenderò in esame il pacchetto di sviluppo Microsoft perché è quello che ho usato per svolgere i miei test.

Guida pratica per il primo listato con Sharpallegro per il 2d.

Vado su nuovo progetto e seleziono 'Applicazione Windows', con nome 'imm01'
Nello spazio del programma a destra, chiamato 'Esplora soluzioni' elimino tutto in modo da avere:
Soluzione 'imm01' (1progetto)
Proprietis ? Elimino tutto e lascio vuoto
Riferimenti ? Col tasto destro scelgo 'aggiungi riferimento...' e quindi mi sposto su 'sfoglia...'
sharpallegro
system
imm01.cs ? Qui rinomino il vecchio nome.cs con questo e cancello il codice in esso contenuto
Vado su 'Salva tutto'
Con doppio click sulla voce 'imm01.cs' vado ad inserire il seguente codice:

```
using System;
using sharpallegro;

namespace exhello
{
class imm01 : Allegro
{
static BITMAP immagine; /* DICHIARAZIONE VARIABILE IMMAGINE */

static void Main()
{
allegro_init(); /* INIZIALIZZA LA LIBRERIA ALLEGRO */
install_keyboard(); /* INSTALLA LA TASTIERA */
set_color_depth(32); /* IMPOSTA LA PALETTE DEI COLORI A 32 BIT */
set_gfx_mode(GFX_GDI, 640, 480, 0, 0); /* IMPOSTA LA MODALITA' GRAFICA 640x480 */
immagine = load_bitmap("immagine.bmp", NULL); /* CARICA IL FILE IMMAGINE.BMP */
blit(immagine, screen, 0, 0, 0, 0, 640, 480); /* VISUALIZZA L'IMMAGINE SULLO SCHERMO */
destroy_bitmap(immagine); /* ELIMINA L'IMMAGINE DAL BUFFER PRIMA DI USCIRE DAL
PROGRAMMA */
readkey(); /* LEGGE UN TASTO QUALSIASI DA TASTIERA */
allegro_exit(); /* TERMINA LA LIBRERIA ALLEGRO */
}
}
}
```

mi assicuro che nella cartella di lavoro(quella dell'eseguibile -Debug-) siano presenti i seguenti files:
immagine.bmp
sharpallegro.dll
alleg44.dll
Salvo tutto e premo il tasto 'F5' per svolgere la compilazione ed avviare il programma appena creato

Guida pratica per il primo listato con Sharpalleggl per il 3d.

Vado su nuovo progetto e seleziono 'Applicazione Windows', con nome '3dgl01'
Nello spazio del programma a destra, chiamato 'Esplora soluzioni' elimino tutto in modo da avere:
Soluzione '3dgl01' (1progetto)
Proprietis ? Elimino tutto e lascio vuoto
Riferimenti ? Col tasto destro scelgo 'aggiungi riferimento...' e quindi mi sposto su 'sfoglia...'
alleggl
allegro
system
gl01.cs ? Qui rinomino il vecchio nome.cs con questo e cancello il codice in esso contenuto
Vado su 'Salva tutto'
Con doppio click sulla voce 'gl01.cs' vado ad inserire il seguente codice:

```
using System;

using allegro;
using alleggl;

public class gl01 : AllegGL
```

```
{

// POSIZIONE, ROTAZIONE E AVVIO CAMERA PER PUNTO DI VISIONE
static void camera ()
{
// aggiunge la prospettiva alla scena
OpenGL.glMatrixMode(OpenGL.GL_PROJECTION);

// comando simile ad un reset di conferma
OpenGL.glLoadIdentity();

// indica le proporzioni della finestra e dell'angolo prospettico
// inoltre l'ultimo valore indica la profondità massima di visione della camera.
// Tutti gli oggetti oltre questo valore non saranno visti nella scena.
OpenGL.glFrustum(-1.0, 1.0, -1.0, 1.0, 1.0, 40.0);

// indica alla scena la matrice di visualizzazione sulle caratteristiche
// del modello
OpenGL.glMatrixMode(OpenGL.GL_MODELVIEW);

// Cancella il buffer del colore RGB ed il depth buffer(pulisce la scena)
OpenGL.glClear(OpenGL.GL_COLOR_BUFFER_BIT | OpenGL.GL_DEPTH_BUFFER_BIT);

// comando simile ad un reset di conferma
OpenGL.glLoadIdentity();

//posiziona la camera
OpenGL.glTranslatef(0.0f, 0.0f, -0.5f);

// orienta la camera nei rispettivi assi...
OpenGL.glRotatef(0, 1, 0, 0); //angolo x
OpenGL.glRotatef(0, 0, 1, 0); //angolo y
OpenGL.glRotatef(0, 0, 0, 1); //angolo z

// salva la posizione e rotazione della camera nella matrice scenica
OpenGL.glPushMatrix();
}

// POLIGONI NELLA SCENA
static void oggetto ()
{
//POSIZIONO E CREO UN OGGETTO POLIGONARE
OpenGL.glTranslatef(-1.5f, 0.0f, -6.0f);

OpenGL.glBegin(OpenGL.GL_POLYGON);

OpenGL.glVertex3f(0.0f, 1.0f, 0.0f); // vertice altro
OpenGL.glVertex3f(-1.0f, -1.0f, 0.0f); // vertice a sinistra
OpenGL.glVertex3f(1.0f, -1.0f, 0.0f); // vertice a destra
OpenGL.glEnd();

// NUOVO OGGETTO POLIGONARE

// Posizione nuovo poligono*/
OpenGL.glTranslatef(3.0f, 0.0f, 0.0f);
```

```
OpenGL.glBegin(OpenGL.GL_POLYGON);

OpenGL.glVertex3f(0.0f, 2.0f, 0.0f); // vertice altro
OpenGL.glVertex3f(-2.0f, -2.0f, 0.0f); // vertice a sinistra
OpenGL.glVertex3f(2.0f, -2.0f, 0.0f); // vertice a destra
OpenGL.glEnd();

OpenGL.glFlush();
allegro_gl_flip();
}

static void Main()
{
// INIZIALIZZO ALLEGROGL
allegro_init();
install_allegro_gl();
allegro_gl_clear_settings();
allegro_gl_set(AGL_COLOR_DEPTH, 32);
allegro_gl_set(AGL_Z_DEPTH, 16);
allegro_gl_set(AGL_FULLSCREEN, TRUE);
allegro_gl_set(AGL_DOUBLEBUFFER, 1);
allegro_gl_set(AGL_SUGGEST, AGL_COLOR_DEPTH | AGL_Z_DEPTH | AGL_DOUBLEBUFFER |
AGL_FULLSCREEN);

// Attiva l'antialiasing (...,1). Disattiva l'antialiasing (...,0)
allegro_gl_set(AGL_RENDERMETHOD, 1);
// Profondità colore Palette
set_color_depth(32);

// Determina la modalità OPENGL e la risoluzione
set_gfx_mode(GFX_OPENGL, 640, 480, 0, 0);

install_keyboard();
install_timer();
allegro_gl_begin();

// VARI COMANDI DI ATTIVAZIONE OPENGL
OpenGL.glEnable(OpenGL.GL_TEXTURE_2D);
OpenGL.glEnable(OpenGL.GL_BLEND);

OpenGL.glViewport(0, 0, SCREEN_W, SCREEN_H);
OpenGL.glMatrixMode(OpenGL.GL_PROJECTION);
OpenGL.glLoadIdentity();
OpenGL.glFrustum(-1.0, 1.0, -1.0, 1.0, 1, 60.0);

OpenGL.glEnable(OpenGL.GL_CULL_FACE);
OpenGL.glFrontFace(OpenGL.GL_CCW);

OpenGL.glMatrixMode(OpenGL.GL_MODELVIEW);

allegro_gl_end();

// INIZIO IL CICLO FINO ALLA PRESSIONE DEL TASTO 'ESC'
do
{
camera();
```

```
oggetto();
} while (!key[KEY_ESC]);
}
}
```

mi assicuro che nella cartella di lavoro(quella dell'eseguibile -Debug-) siano presenti i seguenti files:
alleggl.dll
allegro.dll
alleg42.dll
agl.dll
Salvo tutto e premo il tasto 'F5' per svolgere la compilazione ed avviare il programma appena creato

Trovate il materiale per la compilazione C# esposto negli esempi appena riportati nell'archivio .ZIP collegato a questo libro sotto la cartella 'csharp'. I file di avvio del progetto sono 'imm01.csproj' e '3dgl01.csproj'

Allegro.h diventa multi-piattaforma grazie a DosBox

Ormai sono passati molti anni da quando mi sono appassionato per la prima volta della programmazione di videogames con la libreria C/C++ allegro.h... I miei libri hanno fatto il giro del web. Di questo ne vado molto orgoglioso! Nulla però nel modo dell'informatica dura per sempre ed in questo settore tutto è amplificato rispetto ad altri ambiti. Qualche tempo fa ho aggiunto un'appendice su questo libro, rivolta al C# per rendere il mio lavoro longevo e multi piattaforma. Perseguire l'intento della multi piattaforma oggi come oggi è una priorità e la conversione del sorgente C# usando MonoDevelopment non è così semplice e priva di problemi come potrebbe sembrare.
Quando tutto pare giungere al termine restando segregati alla piattaforma windows(ricordo che allegro.h è parzialmente cross platform), tutto per magia assorbe un'aria di novità andando a ripescare da un lontanissimo passato: quello del mondo DOS! Si avete letto bene, la possibilità della multi piattaforma globale e persistente nel tempo si concretizza con la compilazione dei sorgenti creati con allegro.h per l'OS DOS. Nello specifico con il mitico emulatore DOSBOX. Questo virtualizzatore può essere usato senza riserve come strumento di lancio di un qualsiasi eseguibile DOS in modo completamente invisibile per l'utente, creando un semplice file script. DOSBOX esiste per tutti i computer, console e cellulari esistenti. Proprio per questo motivo potremo far 'girare' i nostri lavori fatti con allegro.h senza cambiare una sola 'virgola' al listato originale!
Nonostante l'estrema versatilità di DOSBOX la piattaforma che mia ha colpito di più per la sua ancora maggiore 'compatibilità universale' è JAVA DOSBOX. Attraverso la java virtual machine non dovrà essere usata una versione di DOSBOX per il sistema in uso, visto che java runtime è praticamente disponibile su tutte le macchine esistenti, compreso l'uso diretto dal web. Si anche questo punto è lampante: potrete far girare i vostri videogame basati su allegro.h come una qualsiasi applet web direttamente sul vostro sito internet, navigando dal vostro browser preferito!
Le richieste di Java DosBox sono esigue per far girare i vostri lavori scritti con in C con la libreria allegro.h. Ho potuto testarli su sistemi virtualizzati con VMware Player con Windows XP, Mac OS, Ubuntu installando le runtime java 2.0 o successive. Stop non è servito altro per vedere su Firefox, Safari e Explorer l'avvio degli eseguibili allegro.h contenuti in Java DosBox, senza che mi accorgessi di nulla! Davvero uno schianto travolgente veder partire dei programmi creati in assoluto stile retrò, su delle piattaforme ultramoderne senza il minimo sentore del passato.
Vediamo meglio quello che si può fare con allegro.h e DosBox prima con una soluzione comprensiva di compilatore DOS(DJGPP+RHIDE) e poi con la versione di lancio diretto del nostro programma appena creato...
TUTTO IL LAVORO DESCRITTO IN QUESTA APPENDICE SARA' IDENTICO PER TUTTE LE PIATTAFORME ESISTENTI.

Qui spiegherò l'uso di DosBox in versione generica:
-DOSBOX 0.74(relase che permette di avviare gli eseguibili Allegro in qualsiasi modalità video)

-JAVA RUNTIME 2.0 o superiore, altrimenti con dosbox nativo non serve nulla
-I FILE PRESENTI SUL MIO SITO WEB PER METTERE IN PRATICA IL TUTTO.

Il compilatore per Dos:
1.Scaricate il file 'djgpp-dosbox-xxx.zip' e decomprimetelo sul vostro disco fisso.
2.Troverete a questo punto la cartella 'djgpp-dosbox-xxx'
3.Entrate nella cartella appena creata ed avviate il programma 'dosbox'
4.Verrà lanciato il compilatore RHIDE per Dos
5.Premete i tasti 'ALT+ENTER' per mettere espandere la finestra a tutto schermo
6.Spostatevi su 'FILE', quindi su 'OPEN' e selezionate il sorgente 'hello.c'
7.Comparirà il listato in C comprensivo dei comandi allegro.h - PS. Visto che si tratta di una versione per DOS se il vostro software è stato creato per windows disognera eliminare il comando finale:
END_OF_MAIN ();
8.Ancora selezionate 'COMPILE' e poi 'BUILD ALL' !!! attendete qualche istante !!!
9.Eseguite il programma appena creato selezionando 'RUN' e ancora 'RUN' dalla tendina.
10.Chiudete il compilatore RHIDE da 'FILE' e poi 'EXIT'

Come avrete notato costruire un eseguibile DOS con questa procedura è molto semplice. Il bello è che non dovrete cambiare mai nulla per ogni macchina che sceglierete di usare!

PREPARARE UN ESEGUIBILE ALLEGRO.H CON DOSBOX IN QUALSIASI VERSIONE:
Dosbox offre la possibilità, grazie ad un file script(dosbox.conf), di preimpostare molti parametri che saranno eseguiti in successione al lancio del programma. Grazie alla manipolazione di questo semplice file di testo sarà possibile 'dire' a dosbox di eseguire direttamente i programmi di Allegro desiderati.
1.Il file 'hellodosbox-xxx.zip' nei file abbinati a questo libro, quindi decomprimiamolo
2.Otterremo una cartella 'hellodosbox-xxx'
3.apriamo con un editor di testo il file 'dosbox.conf'
4.Ci spostiamo al fondo dello script testuale
5.troviamo questo:
mount c .
c:
CWSDPMI.EXE
hello.exe
exit
6.Al posto di 'hello.exe' potremo aprire qualsiasi file Dos in automatico
7.Salviamo il tutto.
8.Quindi copiamo i file del nostro programma Allegro nella cartella 'hellodosbox-xxx'
9.Lanciamo Dosbox e quindi in automatico si avvierà l'eseguibile che abbiamo creato...
10.Se nello script, alla voce 'fullscreen=false' sostituiremo 'fullscreen=true', avvieremo il nostro videogioco a schermo intero!

ALLEGRO.H DAL BROWSER WEB
L'ultima fantastica novità, che amplifica più di tutte le strabilianti sorprese offerte da DosBox è la possibilità di creare pagine web con all'interno una finestra che visualizza gli eseguibili Allegro!!!
Poter mettere sul web un applicativo realizzato programmando in C insieme ad allegro.h è una cosa sorprendente, dato che con il nostro cellulare, palmare o computer sul quale è installato java runtime 2 o superiore sarà possibile vedere i vostri videogames raggiungere l'immortalità informatica.
Vediamo nel dettaglio come realizzare una pagina web che contenga una applet DosBox. Questo prodigio è possibile grazie alla variante JDosBox.

Codice HTML:
```
<html><head>
<title>allegro dosbox</title>
</head><body>
<APPLET CODE="jdos.gui.MainApplet" archive='exhello.jar,jdosbox.jar' WIDTH=640 HEIGHT=480>
```

```
<param value="true" name="separate_jvm">
<param name="param1" value="imgmount e jar://exhello.img -size 512,16,2,512">
<param name="param2" value="e:">
<param name="param3" value="cwsdpmi.exe">
<param name="param4" value="navetta.bat">
</APPLET>
</body></html>
```

Inserendo in un file il testo html appena indicato saranno caricati:
1.jdosbox.jar, dove è contenuto l'emulatore java di DosBox
2.exhello.jar, dove è contenuto il file all'interno delle cartella 'jdos' l'immagine 'exhello.img'
3.Quindi tutti gli script di configurazione per il lancio dell'applicazione finale. In questo caso 'navetta.bat' potrà essere sostituito con qualsiasi applicazione contenuta nell'immagine...
4.Aprendo il file html verranno caricati tutti i file ed archivi direttamente online dal browser web preferito(Firefox, Safari, Explorer, Opera...)
5.Per gestire e creare gli archivi descritti saranno necessari i software(ambiente windows): Winimage e Winrar.

Trovate un esempio attivo del codice html appena espresso sul mio sito web, per la gestione online dei pacchetti allegro.h(Non funzionerà sul vostro disco fisso ma solo su sito web!)

Alcuni piccoli cambiamenti dalla versione 3.0 alla 4.20

Allegro.h si è trasformato negli anni migliorando sempre fino alla versione 4.x, Poi la battuta d'arresto con le versioni 5.x che di fatto l'hanno trasformato in un'altra libreria decretando l'abbandono di moltissimi programmatori in favore di soluzioni alternative... Un vero peccato! Diverso sarebbe stato se si fosse continuato lo sviluppo includendo la retrocompatibilità della libreria con le versioni precedenti.

Senza divagare voglio in queste righe aggiornarvi su un costrutto di un paio di comandi per la conversione e visualizzazione delle stringhe in modalità grafica: *sprintf - textout*, che solitamente fornisce un messaggio d'errore 'deprecated...' nelle versioni successive alla 4.X della libreria...

Segue estratto preso dal listati 'inveder':

```
sprintf(score, "PUNTI: %ld VITE: %ld", (long)contapunti, (long)vite); // da il valore a score e a vite da visualizzare //

textout(buf, font, score, 1, 1, 215); // inserisce nel doppio buffer dello schermo il valore di score e vite che verrà poi visualizzato //
```

La soluzione esposta era stata ideate per le virsioni 3.x Dos di allegro.h, che nelle situazioni fornite dalle nuove versioni riportano problemi, salvo modifiche ai parametri del compilatore usato(inibizione messaggi warning...).

Con il comando qui sotto esposto si utilizza una sola funzione che fonde sprintf e textout, eliminando ogni avversità. Segue dettaglio:

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

BITMAP *bmp = puntatore di variabile bitmap

const FONT *f = font set usato

const char *s = Stringa in formato char da visualizzare

int x, int y = Posizione del testo sullo schermo

int color, int bg = Valore di colori di sfondo e testo

Conclusione

Siamo giunti al termine di questo lavoro. Oggi i programmatori di videogame usano editor totalmente costruiti per realizzare videogame con grafica di alto livello 3d... Le nozioni base però sono contenute in libri vecchio stampo come questo, dove si realizza praticamente tutto di propria mano partendo quasi da zero(salvo per i comandi di allegro.h) con molte linee in linguaggio C... Se si diventa capaci di realizzare videogame retrò con le strategie di programmazione contenute in questo ebook allora le porte si apriranno per ogni futuro sviluppo con soluzioni differenti. Certo, ai tempi del Commodore 64 o dell'Amiga si mettevano insieme autentici miracoli con il linguaggio assembler, che impressionano ancora oggi con un pugno di chilobyte. Poi con l'avvento dei PC ha preso sempre più piede la programmazione ludica con il linguaggio C che fornisce un buon compromesso sul come si creano routine di ogni genere dal semplice al complesso in rapporto ad una discreta semplicità di realizzo... Voglio aggiungere che molto del codice qui esposto ho potuto farlo funzionare di un vecchio 386dx 40mhz, giusto per far capire quello che sarebbe possibile creare con le moderne CPU!

Sulla mia pagina(http://www.bertinettobartolomeodavide.it/programmazione/ALLEGROH/) dedicata ad allegro.h potrete trovare molte chicche, aiuti e se avrò altro tempo da dedicare anche una curiosa novità a cui sto pensando da tempo... Chissà!

Grazie per aver letto il presente manuale.

Bartolomeo Davide Bertinetto.
Diplomato ISEF e laureato in Scienze Motorie con Specializzazione Tecnico-Sportivo presso l'Università di Torino. Professore a scuola e proprietario di un centro fitness... Oltre che autore del metodo anti età Matevo® dedicato alla costruzione fisica.
Da sempre appassionato di computer in vari settori: Programmazione in Assembler, C/C++, Visual C++, C#, Gwbasic, Pascal e JAVA. Creazione di circuiti elettronici e programmazione di microprocessori PIC 16F84 e Basic Stamp 2. Ancora attivo programmatore del vecchio Commodore 64.
Esperto di Lightwave 3D per la realizzazione di immagini e animazioni 3D sintetiche(sopratutto sulla gloriosa piattaforma Amiga classica), con partecipazione ad alcune edizioni del PIXEL ART di Roma.
WebMaster del sito 'www.bertinettobartolomeodavide.it', dedicato alla sue passioni tra cui la tecnica del monoallenamento per l'ultratrail di cui è ideatore, che conta migliaia di visite... Divulgatore di astronomia ed attivo astrofilo... Collezionista di computer Retrò.
email: contatto@bertinettobartolomeodavide.it