Corso Programmazione ad Oggetti

Corso di Laurea in Informatica

ALLEGRO

Pierangela Bruno
bruno@mat.unical.it

Installazione

Installare Allegro da Ubuntu PPAs

```
sudo add-apt-repository ppa:allegro/5.2
sudo apt-get update
sudo apt-get install liballegro5-dev
```

Installare i pacchetti Allegro

```
sudo apt-get install -y cmake g++ freeglut3-dev
libxcursor-dev libpng12-dev libjpeg-dev
libfreetype6-dev libgtk2.0-dev libasound2-dev
libpulse-dev libopenal-dev libflac-dev
libdumb1-dev libvorbis-dev libphysfs-dev
```

https://wiki.allegro.cc/index.php?title=Getting_Started

Installazione

Install Allegro5 From Git

```
git clone https://github.com/liballeg/allegro5.git
```

Build Allegro

```
cd allegro5
mkdir build
cd build
cmake ..
#or run cmakegui ..
make
sudo make install
```

Installazione

```
sudo ldconfig
```

ldconfig crea i link necessari delle librerie più recenti installate

Compilare un'applicazione Allegro 5 C++

```
g++ [source file(s)] -o [output]
[-I/usr/include/allegro5 -L/usr/lib] -lallegro
```

Creare un display

```
#include <allegro5/allegro.h>
```

IMPORTANTE: Effettuare l'inclusione nel main per eseguire le funzioni di Allegro, anche se non si inizializza Allegro in questo file

```
  if(!al_init()) {
     cerr << "failed to initialize allegro!\n";
     return -1;
  }
```

Inizializzazione della libreria Allegro

```
display = al_create_display(w, h);
```

Creazione di un display con larghezza e altezza specificate

Creare un display (Contd.)

```
al_clear_to_color(al_map_rgb(0,0,0));
```

al_map_rgb accetta tre argomenti (rosso, verde e blu) e restituisce una struttura **ALLEGRO_COLOR**

```
al_flip_display();
```

Allegro crea due buffer di immagine:

- quello che viene visualizzato sullo schermo
- quello che viene disegnato nel codice

al_flip_display viene chiamato per scambiare i due buffer di immagine in modo che il primo buffer di immagini (quello su cui hai disegnato) sia ora visualizzato mentre il secondo buffer di immagini diventa quello su cui disegnare.

Creare un display (Contd.)

```
al_rest(10.0);
```

al_rest specifica in secondi la "sleep" del programma

```
al_destroy_display(display);
```

IMPORTATNTE: Distruggere il display e liberare la memoria quando si sta per chiudere il programma

Display fullscreen

```
ALLEGRO_DISPLAY_MODE disp_data;
al_get_display_mode(al_get_num_display_modes()-1,
&disp_data);
```

ALLEGRO_DISPLAY_MODE memorizza le informazioni del monitor

al_get_num_display_modes () restituisce il numero totale di modalità di visualizzazione (o risoluzioni) che il monitor è in grado di gestire

al_get_num_display_modes () - 1 rappresenta l'indice per la risoluzione *minima* del monitor. Per la risoluzione *massima* si deve chiamare al_get_display_mode() con 0 come primo argomento

Risoluzione Indipendente

Adattare la risoluzione del gioco alle possibili risoluzioni del display (anche con schermo intero!)

Quando applicare la risoluzione indipendente?

- se si progetta un gioco per una risoluzione *(ad esempio)* di 640x480 e si desidera supportare risoluzioni a schermo intero

- oppure se si progetta un gioco per una risoluzione *(ad esempio)* di 320x240 e desideri che il gioco sia dotato di finestre di dimensioni pari a 800x600

Soluzione

Non importa le risoluzione effettiva usata!

E' possibile "allungare" la risoluzione prevista in modo che si adatti a qualunque altra risoluzione.

Risoluzione Indipendente (Contd.)

Transformations: consigliato quando si usano molte primitive

```
//Crea un display a schermo intero
//(utilizza automaticamente la risoluzione nativa dell'utente)
al_set_new_display_flags(ALLEGRO_FULLSCREEN_WINDOW);
display = al_create_display(windowWidth, windowHeight);

windowWidth = al_get_display_width(display);
windowHeight = al_get_display_height(display);

//larghezza e altezza della risoluzione per la quale stiamo progettando il gioco
int screenWidth = 640;
int screenHeight = 480;

//i fattori di scala
float sx = windowWidth / (float)screenWidth;
float sy = windowHeight / (float)screenHeight;

//Ridimensiona il display utilizzando una trasformazione
ALLEGRO_TRANSFORM trans;
al_identity_transform(&trans);
al_scale_transform(&trans, sx, sy);
al_use_transform(&trans);
```

Allegro_transform deve essere eseguito solo una volta durante l'inizializzazione!

Risoluzione Indipendente (Contd.)

Stretched Buffer: consigliato quando si usano molte bitmap

```
//Crea un display a schermo intero
//(utilizza automaticamente la risoluzione nativa dell'utente)
al_set_new_display_flags(ALLEGRO_FULLSCREEN_WINDOW);
display = al_create_display(windowWidth, windowHeight);

//larghezza e altezza della risoluzione per la quale stiamo progettando il gioco
int screenWidth = 640;
int screenHeight = 480;
buffer = al_create_bitmap(screenWidth, screenHeight);

//i fattori di scala
int sx = windowWidth / screenWidth;
int sy = windowHeight / screenHeight;
int scale = std::min(sx, sy);

//calculare di quanto il buffer deve essere scalato
//non possiamo semplicemente allungare il buffer sull'intera risoluzione in quanto
//un gioco progettato per schermi in 4:3 (esempio) apparirà strano su schermi larghi
scaleW = screenWidth * scale;
scaleH = screenHeight * scale;
scaleX = (windowWidth - scaleW) / 2;
scaleY = (windowHeight - scaleH) / 2;

//disegna bitmap scalata
al_draw_scaled_bitmap(buffer, 0, 0, screenWidth, screenHeight, scaleX, scaleY,
scaleW, scaleH, 0);
```

Se il buffer non si adatta all'intero schermo, per ridimensionare correttamente verranno aggiunte barre nere nel contorno.

Event

```
ALLEGRO_EVENT_QUEUE *event_queue = al_create_event_queue();

al_register_event_source(event_queue,
al_get_display_event_source(display));
```

al_register_event_source registra il display alla coda degli eventi, così da poter essere informati degli eventi che avvengono sul display, come l'evento close

```
bool get_event = al_wait_for_event_until(event_queue, &ev, &timeout);
```

Il programma aspetta l'arrivo di un evento o la scadenza del timeout

```
if(get_event && ev.type == ALLEGRO_EVENT_DISPLAY_CLOSE) {
   break;
}
```

Controlliamo se abbiamo ricevuto un evento nella coda degli eventi. Se si tratta di un evento di chiusura del display, interrompiamo il ciclo del programma

Bitmap

```
bouncer = al_create_bitmap(32, 32);
```

al_create_bitmap crea una bitmap con larghezza e altezza stabilite

```
al_set_target_bitmap(bouncer);
```

Seleziona la bitmap di destinazione sulla quale verranno applicate tutte le successive operazioni di disegno

- Ogni bitmap è legata a un display
- Un singolo display non può essere usato contemporaneamente da più thread

Bitmap (Contd.)

```
al_set_target_bitmap (NULL);
```

Necessaria per rilasciare un display

```
al_set_target_bitmap(al_get_backbuffer(display));
```

Permette di tornare normalmente al disegno sullo schermo

```
al_draw_bitmap(bouncer, bouncer_x, bouncer_y, 0);
```

Disegnare la bitmap sul display

```
al_destroy_bitmap(bouncer);
```

Distrugge la bitmap

Caricare un'immagine

```
ALLEGRO_BITMAP  *image = al_load_bitmap("image.png");
```

IMPORTANTE: Se si carica un'immagine prima della creazione di un display, verrà caricata come una Memory Bitmap che è estremamente lenta, per cui si devono caricare i bitmap solo dopo la creazione di un display

```
al_draw_bitmap(image,200,200,0);
```

Disegnare l'immagine sul display

```
al_destroy_bitmap(image);
```

Distrugge l'mmagine e liberare la memoria

Keyboard

```
enum MYKEYS { KEY_UP, KEY_DOWN, KEY_LEFT, KEY_RIGHT};
bool key[4] = { false, false, false, false };
```

Definire un array dove memorizzare lo stato delle chiavi a cui siamo interessati

```
al_install_keyboard()
```

Inizializzare la tastiera

```
al_register_event_source(event_queue,
al_get_keyboard_event_source());
```

Registra gli eventi della tastiera alla coda di eventi

Keyboard (Contd.)

```
if(ev.type == ALLEGRO_EVENT_KEY_DOWN) { .. }
```

ALLEGRO_EVENT_KEY_DOWN viene generato ogni volta che viene premuto un tasto

```
switch(ev.keyboard.keycode) {
     case ALLEGRO_KEY_UP:
          key[KEY_UP] = true;
          break;
}
```

keycode contiene una mappatura intera della tastiera e permette di determinare il tasto premuto e aggiornare la chiave corrispondente

```
key[KEY_UP] = false;
```

IMPORTANTE: ogni volta che viene rilasciata una chiave deve essere impostata a false (= tasto NON premuto)

Mouse

```
al_install_mouse()
```

Inizializzare il mouse

```
al_register_event_source(event_queue,
al_get_mouse_event_source());
```

Registra gli eventi del mouse alla coda di eventi

```
if(ev.type == ALLEGRO_EVENT_MOUSE_AXES ||
ev.type == ALLEGRO_EVENT_MOUSE_ENTER_DISPLAY) {
    bouncer_x = ev.mouse.x;
    bouncer_y = ev.mouse.y;
}
```

ALLEGRO_EVENT_MOUSE_AXES e **ALLEGRO_EVENT_MOUSE_ENTER_DISPLAY** vengono generati, rispettivamente, quando il mouse viene spostato dall'utente e quando il mouse entra nel display

Timers

I timer possono essere utilizzati per regolare gli aggiornamenti nel main, quindi è possibile eseguire un'operazione o un aggiornamento solo ad ogni frame al secondo (FPS)

```
const float FPS = 60;
timer = al_create_timer(1.0 / FPS);
al_register_event_source(event_queue,
al_get_timer_event_source(timer));
```

Registrare gli eventi del timer nella coda di eventi in modo da poterli recuperare in seguito

```
al_start_timer(timer);
```

Necessario per far partire il timer. Senza questo, nessun evento sarà generato

Timers

```
al_wait_for_event(event_queue, &ev);
if(event.type == ALLEGRO_EVENT_TIMER) {
            redraw = true;
}
```

Permette di attendere l'arrivo di un evento e innescare un ridisegno. Gli eventi in arrivo sono regolari, non dobbiamo preoccuparci di rimanere bloccati nella funzione di attesa

```
if(redraw && al_is_event_queue_empty(event_queue)) {
    redraw = false;
    al_clear_to_color(al_map_rgb(0,0,0));
    al_flip_display();
}
```

Assicurarsi che la coda degli eventi sia completamente vuota prima di eseguire il ridisegno, altrimenti il ciclo di aggiornamento potrebbe fallire