**MLX -funzioni utili-**

**void \*mlx\_new\_image(void \*mlx\_ptr, int width, int height);**La funzione mlx\_new\_image è utilizzata per creare una nuova immagine in memoria.

**Parametri**

* mlx\_ptr: Un puntatore alla connessione alla libreria MiniLibX. Questo puntatore viene ottenuto come risultato della funzione mlx\_init.
* width: La larghezza dell'immagine in pixel.
* height: L'altezza dell'immagine in pixel.

**Ritorno**Restituisce un puntatore alla nuova immagine. Questo puntatore viene utilizzato per ulteriori manipolazioni dell'immagine, come disegnarla in una finestra o manipolare i pixel.

**Funzionamento**  
Quando chiami mlx\_new\_image, la funzione alloca la memoria necessaria per memorizzare un'immagine della dimensione specificata (larghezza per altezza) e restituisce un puntatore a questa nuova immagine. Questo puntatore può quindi essere utilizzato con altre funzioni della libreria MiniLibX per manipolare l'immagine.

**char \*mlx\_get\_data\_addr(void \*image, int \*bpp, int \*size\_line, int \*endian);**

La funzione mlx\_get\_data\_addr è utilizzata per ottenere un puntatore ai dati dell'immagine creata con mlx\_new\_image, insieme a informazioni importanti come i bit per pixel, la dimensione di una linea di pixel in memoria e l'endianness del sistema. Vediamo in dettaglio come funziona questa funzione.

**Parametri**

* image: Un puntatore all'immagine precedentemente creata con mlx\_new\_image.
* bpp: Un puntatore a un intero dove verrà memorizzato il numero di bit per pixel.
* size\_line: Un puntatore a un intero dove verrà memorizzata la dimensione di una linea dell'immagine in memoria (in byte).
* endian: Un puntatore a un intero dove verrà memorizzata l'endianness dell'immagine.

**Ritorno**  
Restituisce un puntatore (char \*) ai dati grezzi dell'immagine. Questo puntatore permette di accedere direttamente ai pixel dell'immagine.

**Dettagli dei parametri:**

**image**:

Questo è il puntatore all'immagine ottenuta da mlx\_new\_image. Rappresenta l'immagine in memoria che vuoi manipolare.

**bpp (bits per pixel)**:  
È un intero che specifica quanti bit sono utilizzati per rappresentare ogni pixel dell'immagine.  
  
Un pixel è un punto nell'immagine e rappresenta un singolo colore. Per rappresentare questo colore, sono necessari un certo numero di bit.  
  
Comunemente, questo valore è 32, il che significa che ogni pixel è rappresentato da 32 bit (4 byte). Questo permette di avere 8 bit per ciascuno dei tre canali di colore (rosso, verde, blu) e 8 bit per il canale alpha (trasparenza).

**size\_line**:  
Questo parametro memorizza la dimensione di una singola linea di pixel in memoria, in byte. Questo valore è utile per calcolare l'offset quando si accede ai pixel in immagini multi-linea.

***Dimensione di una Linea di Pixel****:* size\_line indica quanti byte occupa una singola riga di pixel in memoria.  
 Questo include tutti i pixel di quella riga, indipendentemente dal loro   
 colore o formato.

***Calcolo dell'Offset****:* Quando si accede ai pixel di un'immagine, size\_line viene utilizzato per calcolare  
 l'offset in memoria tra le righe successive dell'immagine. Questo è particolarmente   
 utile quando si scorre verticalmente attraverso l'immagine.

### *Spiegazione del Calcolo dell'Offset* Quando si parla di immagini digitali, ogni pixel dell'immagine è memorizzato in un'area contigua di memoria. Per accedere ai dati di un pixel specifico, è necessario conoscere l'offset (lo spostamento) in memoria rispetto all'inizio dell'immagine.

***size\_line e Offset*** Il parametro size\_line rappresenta il numero di byte occupati da una singola linea (riga)   
 di pixel nell'immagine. Questo valore è cruciale per calcolare l'offset dei pixel quando si   
 scorre verticalmente attraverso l'immagine, ovvero quando si passa da una riga   
 all'altra. ***Accesso ai Pixel*** Per accedere a un pixel specifico in un'immagine, bisogna calcolare la posizione in  
 memoria di quel pixel. Questa posizione è determinata dall'offset, che tiene conto   
 della posizione del pixel all'interno della sua riga e della riga stessa.

**endian**:

* Questo parametro memorizza l'endianness dell'immagine. Può avere due valori:  
  - 0 per little-endian: I byte meno significativi sono memorizzati prima.  
  - 1 per big-endian: I byte più significativi sono memorizzati prima.

**Spiegazione dell'Endian**

L'endianness (endianness in inglese) si riferisce all'ordine dei byte usato per rappresentare i numeri in memoria. Esistono due principali tipi di endianness:

1. **Little-endian**: Il byte meno significativo (least significant byte, LSB) viene memorizzato per primo. Questo è comunemente utilizzato nei sistemi basati su processori Intel.
2. **Big-endian**: Il byte più significativo (most significant byte, MSB) viene memorizzato per primo. Questo è comunemente utilizzato nei sistemi basati su processori PowerPC e nelle architetture di rete.

**Dettagli dell'Endian**

**Little-endian**

In un sistema little-endian, i byte vengono memorizzati dall'LSB al MSB. Ad esempio, il numero esadecimale 0x12345678 viene memorizzato in memoria come:

| **Indirizzo** | **Valore (byte)** |
| --- | --- |
| 0 | 0x78 |
| 1 | 0x56 |
| 2 | 0x34 |
| 3 | 0x12 |

Quando chiami mlx\_get\_data\_addr, la libreria MiniLibX analizza l'immagine specificata e riempie i puntatori forniti (come bpp, size\_line, e endian) con i valori appropriati.

**Big-endian**

In un sistema big-endian, i byte vengono memorizzati dal MSB all'LSB. Ad esempio, il numero esadecimale 0x12345678 viene memorizzato in memoria come:

| **Indirizzo** | **Valore (byte)** |
| --- | --- |
| 0 | 0x12 |
| 1 | 0x34 |
| 2 | 0x56 |
| 3 | 0x78 |

**int mlx\_loop\_hook(void \*mlx\_ptr, int (\*funct\_ptr)(), void \*param);**

• La funzione mlx\_loop\_hook è una funzione della libreria MLX (MinilibX) utilizzata per registrare una funzione callback che verrà chiamata ad ogni iterazione del ciclo principale di MLX. Questo è utile per aggiornamenti continui e animazioni.

**Parametri**

* mlx\_ptr: Un puntatore alla connessione MLX, solitamente ottenuto da mlx\_init().
* funct\_ptr: Un puntatore alla funzione callback che desideri chiamare ad ogni iterazione del ciclo principale.
* param: Un puntatore a un parametro opzionale che verrà passato alla funzione callback. Può essere NULL se non si desidera passare alcun parametro.

**Valore di Ritorno**

La funzione restituisce un intero (solitamente 0) che indica il successo della registrazione del callback.

Es:

void \*mlx;    // Puntatore alla connessione MLX

void \*win;    // Puntatore alla finestra MLX

int counter = 0;  // Contatore che verrà aggiornato

// Funzione callback che verrà chiamata ad ogni iterazione del ciclo principale

int update(void \*param) {

    char str[100];

    // Cancella la finestra

    mlx\_clear\_window(mlx, win);

    // Incrementa il contatore

    counter++;

    // Crea una stringa con il valore del contatore

    sprintf(str, "Counter: %d", counter);

    // Visualizza il contatore nella finestra

    mlx\_string\_put(mlx, win, 10, 10, 0xFFFFFF, str);

    // Restituisci 0 per continuare il loop

    return 0;

}

int main() {

    // Inizializza la connessione MLX e crea una finestra

    mlx = mlx\_init();

    win = mlx\_new\_window(mlx, 800, 600, "MLX Loop Hook Example");

    // Registra la funzione callback per l'update continuo

    mlx\_loop\_hook(mlx, update, NULL);

    // Avvia il loop principale di MLX

    mlx\_loop(mlx);

    return 0;

}

**int mlx\_hook(void \*win\_ptr, int x\_event, int x\_mask, int (\*funct\_ptr)(), void \*param);**

La funzione mlx\_hook è una funzione della libreria MiniLibX utilizzata per gestire eventi specifici, come i movimenti del mouse, i click o la pressione dei tasti, all'interno di una finestra creata con MiniLibX. Permette di associare una funzione di callback a un particolare evento, in modo che questa funzione venga chiamata ogni volta che l'evento si verifica.

**Parametri**

1. **win\_ptr**:
   * Puntatore alla finestra creata con mlx\_new\_window.
2. **x\_event**:
   * Tipo di evento che si desidera catturare (ad esempio, una pressione di un tasto, il movimento del mouse, ecc.). I tipi di eventi sono definiti come costanti nella libreria X11 e nella MiniLibX.
3. **x\_mask**:
   * Una maschera che specifica le condizioni in cui l'evento deve essere generato. Questo parametro può essere ignorato (passando 0) nella maggior parte dei casi con MiniLibX.
4. **funct\_ptr**:
   * Puntatore alla funzione di callback che deve essere chiamata quando l'evento si verifica. Questa funzione deve avere una specifica firma, in genere accetta un certo numero di parametri che descrivono l'evento.
5. **param**:
   * Puntatore a una struttura o dato che si desidera passare alla funzione di callback. Questo permette di passare variabili definite dall'utente alla funzione di callback.

**Eventi Comuni**

Alcuni dei tipi di eventi più comuni e le loro costanti corrispondenti sono:

* **KeyPress (pressione di un tasto)**: 2
* **KeyRelease (rilascio di un tasto)**: 3
* **ButtonPress (pressione di un pulsante del mouse)**: 4
* **ButtonRelease (rilascio di un pulsante del mouse)**: 5
* **MotionNotify (movimento del mouse)**: 6
* **EnterNotify (entrata del cursore del mouse nella finestra)**: 7
* **LeaveNotify (uscita del cursore del mouse dalla finestra)**: 8
* **Expose (esposizione della finestra)**: 12
* **DestroyNotify (chiusura della finestra)**: 17