Libreria grafica: OpenGL

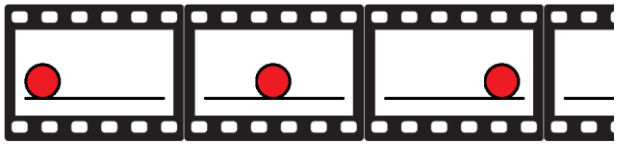
Come realizzare semplici simulazioni video in C con OpenGL

Simulazione video

- Una **simulazione** è un modello della realtà che consente di valutare e prevedere lo svolgersi dinamico di una serie di eventi a partire da certe condizioni inziali.

$$s(t) = \frac{1}{2}at^2 + v_0t + s_0$$

- Un **video** è una successione di immagini rappresentate su schermo.



- Una **simulazione video** è dunque una successione di immagini, in cui ciascuna immagine è una matrice calcolata sulla base di un modello.

Prinicipi di grafica

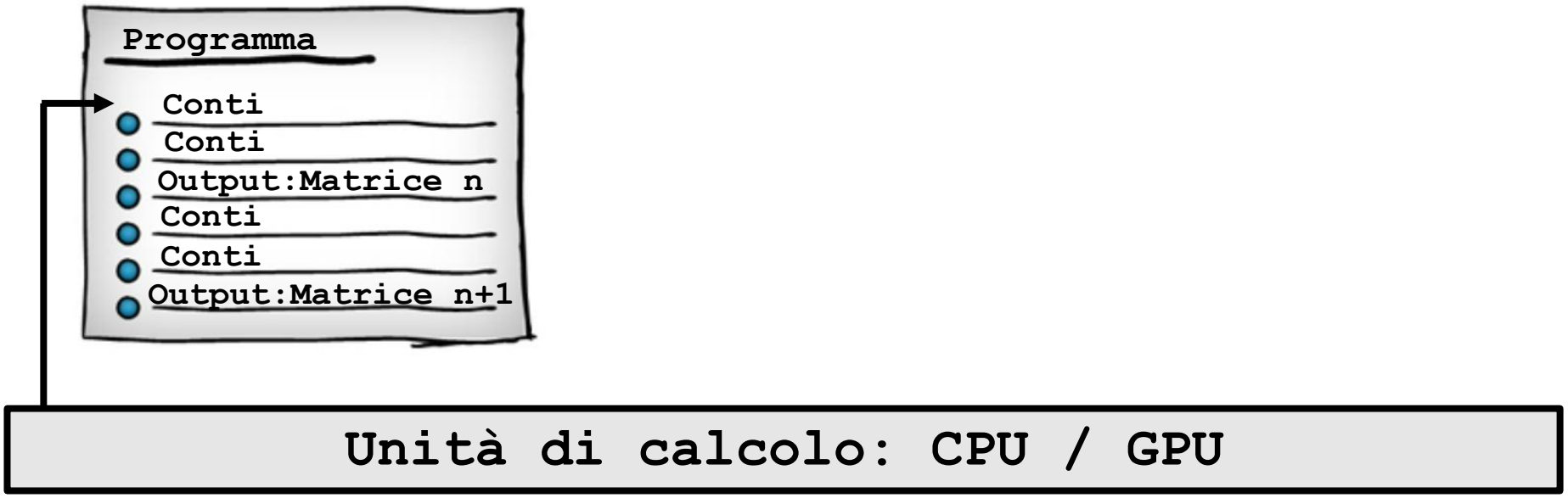
- Cosa vuol dire successione di immagini?
  - La successione non può essere un accostamento di immagini.
  - Ogni nuova immagine deve «prendere il posto della precedente»: solo così si ha l'effetto animazione.
  - È chiaro quindi che non è possibile fare una animazione stampando sullo standard output.
  - Bisogna quindi che il nostro programma sia in grado di aprire una finestra, diversa dal terminale, e di stampare in essa di volta in volta le immagini da rappresentare.

Prinicipi di grafica

- Cosa vuol dire scrivere di volta in volta la matrice da stampare nella finestra ?
  - Supponiamo che la nostra unità di calcolo (CPU o GPU), abbia calcolato la matrice n-esima e adesso la debba stampare su una finestra a video.
  - Mentre stampare su standard output è facilissimo, la lettura e la stampa a video su finestra sono operazioni molto lente rispetto alla velocità dell'unità di calcolo.
  - Quindi se la CPU/GPU comunicasse direttamente con lo schermo, senza intermediari, dopo aver calcolato ogni matrice dovrebbe aspettare che lo schermo abbia finito di leggere e rappresentare tutti i dati prima di poter proseguire con il calcolo della matrice successiva.

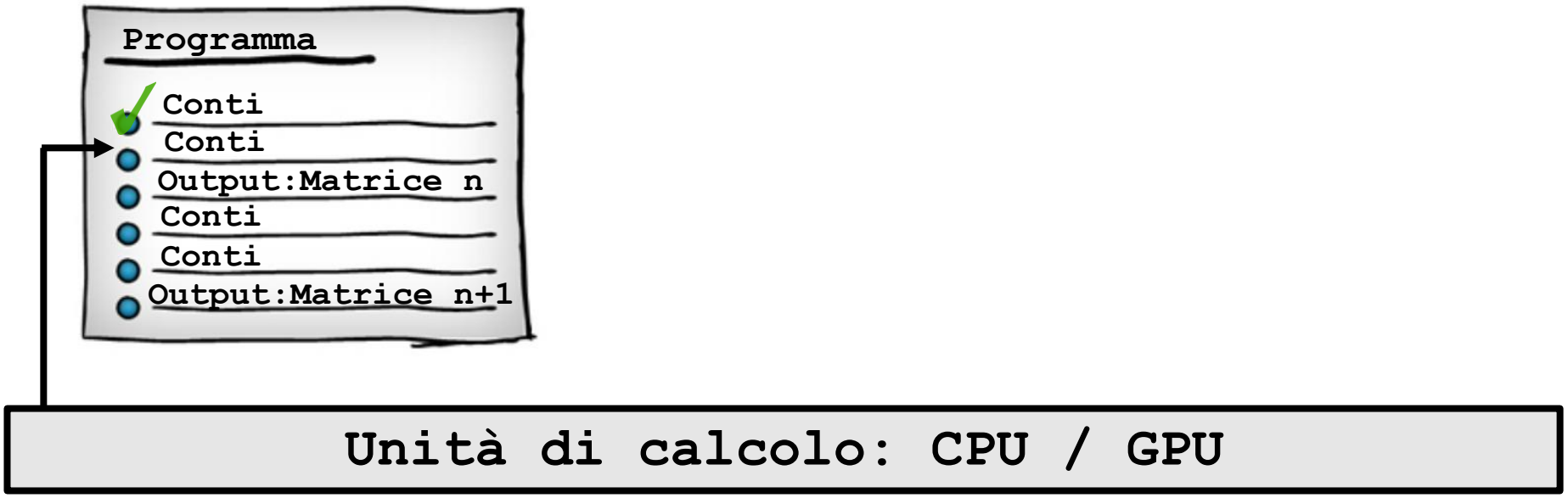
Prinicipi di grafica

- Cosa vuol dire scrivere di volta in volta la matrice da stampare nella finestra ?



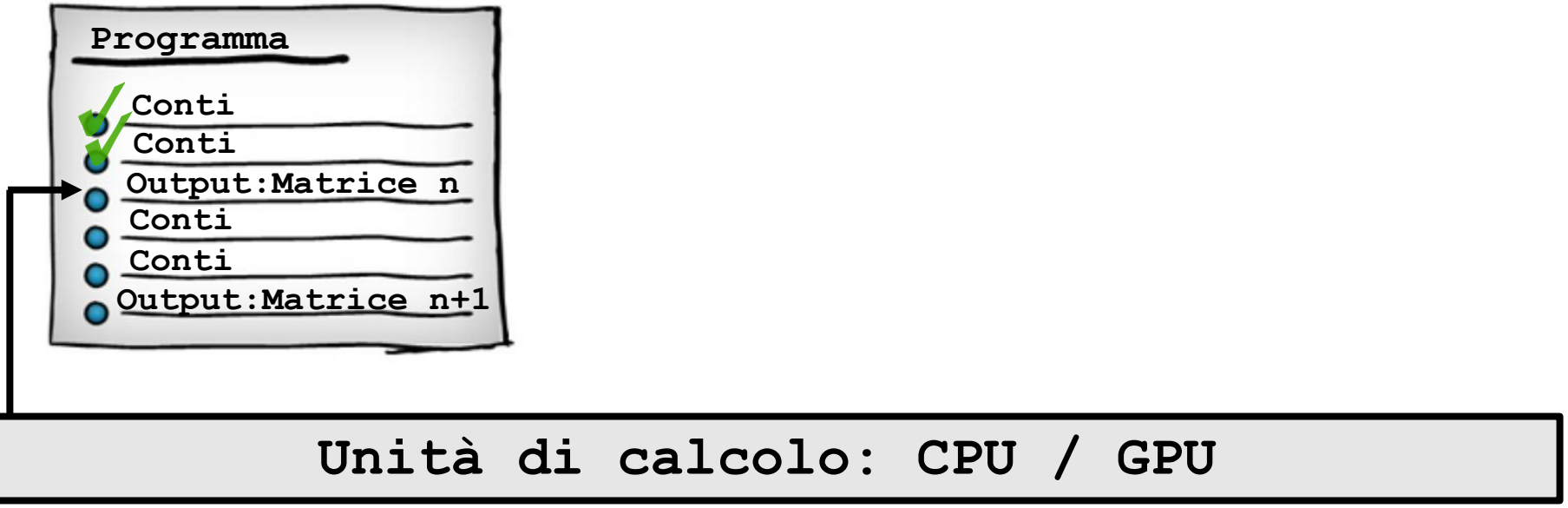
Prinicipi di grafica

- Cosa vuol dire scrivere di volta in volta la matrice da stampare nella finestra ?



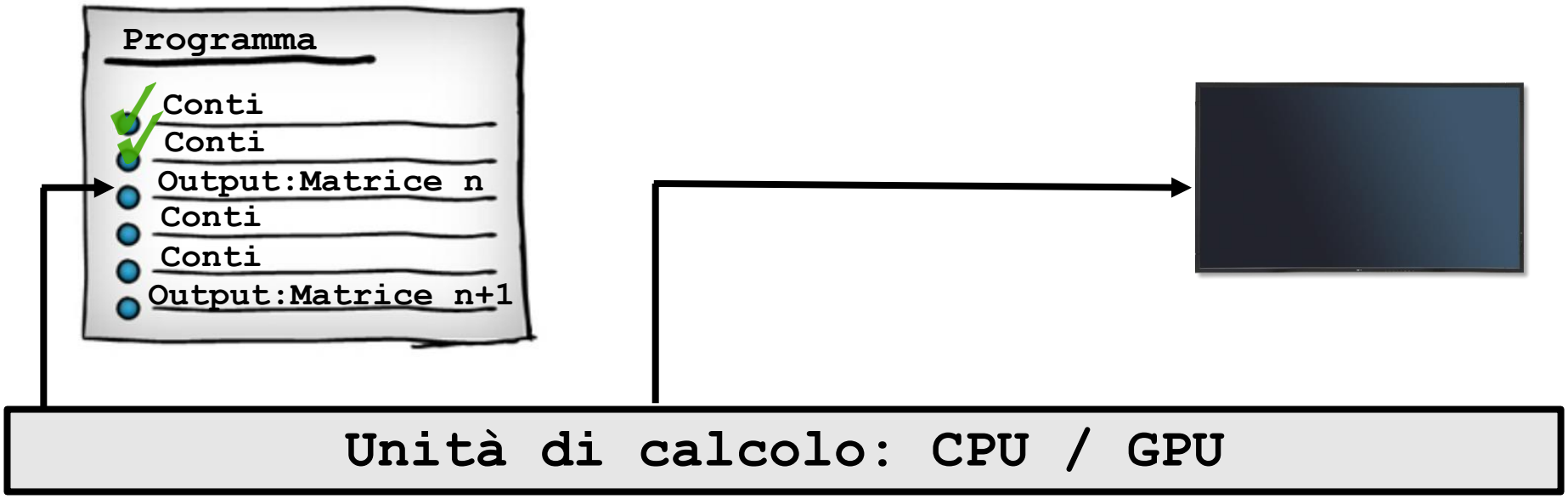
Prinicipi di grafica

- Cosa vuol dire scrivere di volta in volta la matrice da stampare nella finestra ?



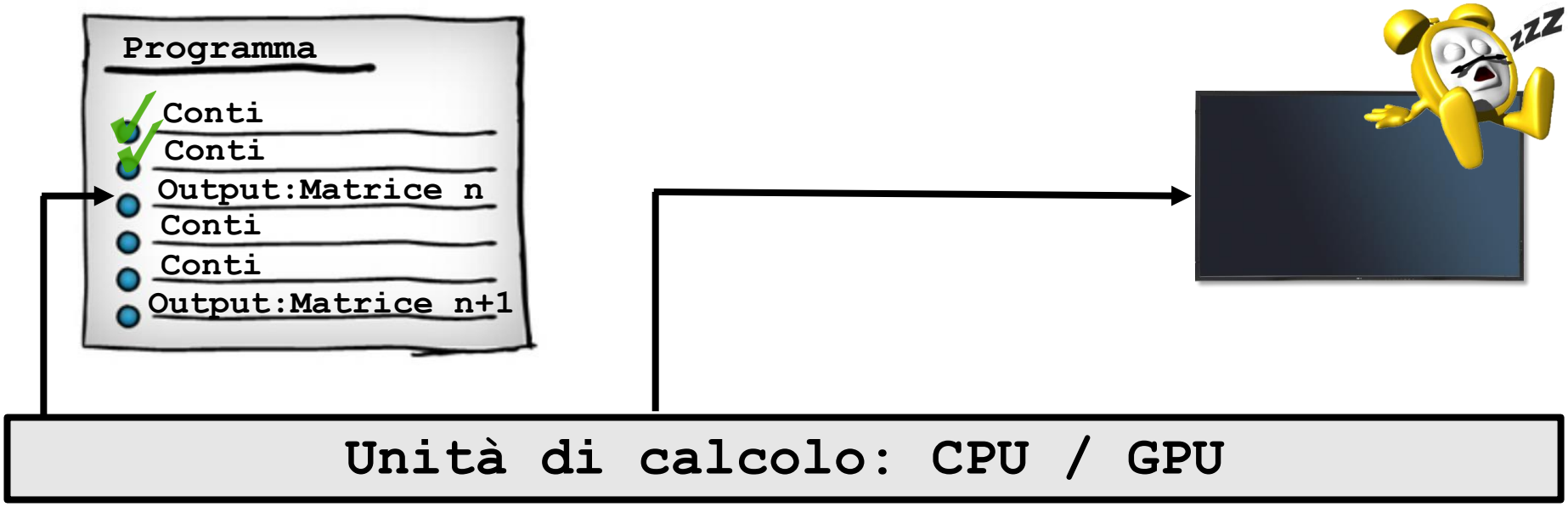
Prinicipi di grafica

- Cosa vuol dire scrivere di volta in volta la matrice da stampare nella finestra ?



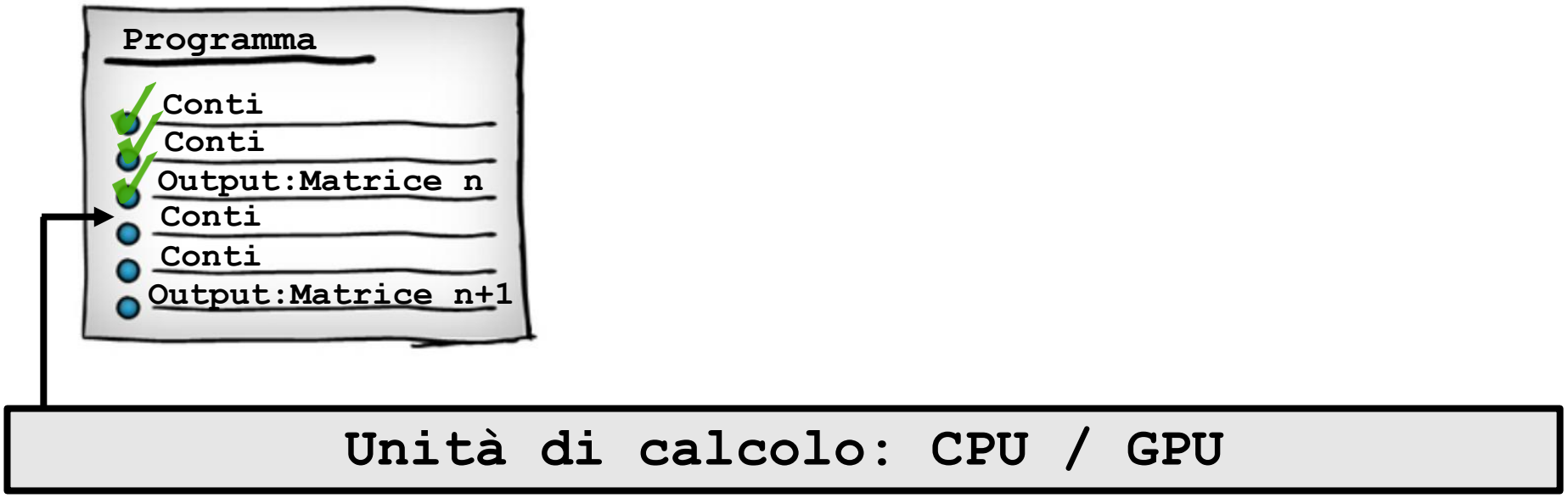
Prinicipi di grafica

- Cosa vuol dire scrivere di volta in volta la matrice da stampare nella finestra ?



Prinicipi di grafica

- Cosa vuol dire scrivere di volta in volta la matrice da stampare nella finestra ?



- Questo chiaramente peggiora la qualità del video che risulterà a scatti.

Prinicipi di grafica

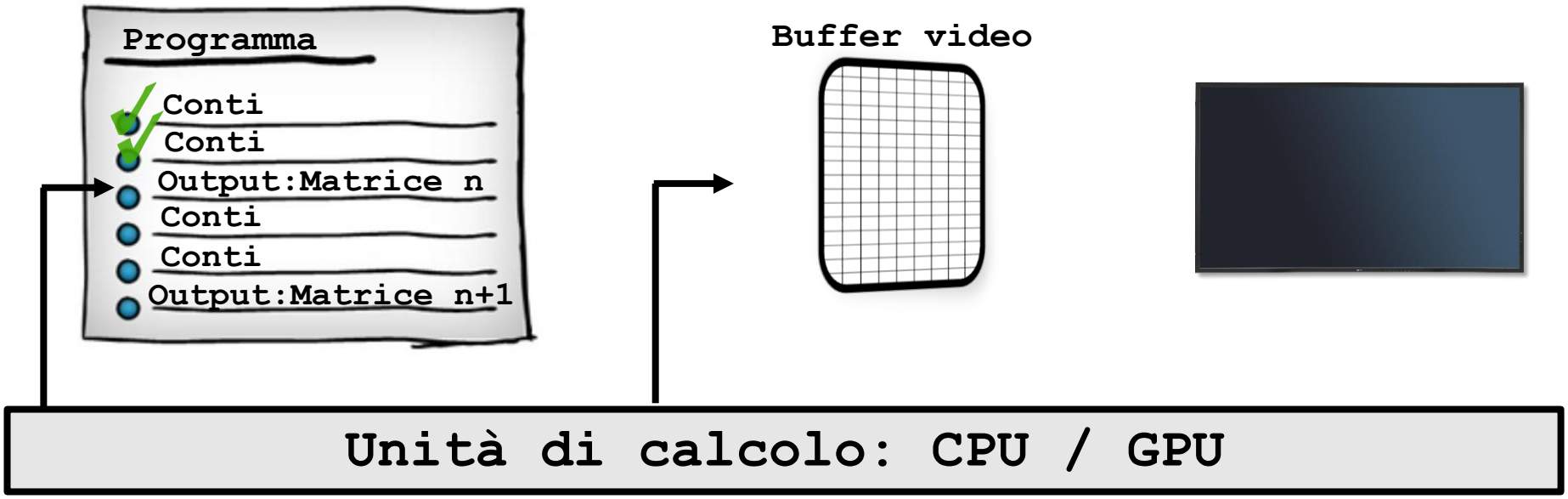
- Cosa vuol dire scrivere di volta in volta la matrice da stampare nella finestra ?
  - Per questo viene introdotto un intermediaro: il **buffer**.
  - Si tratta di una memoria temporanea in cui si memorizzano dati che verranno successivamente trasmessi ad altri dispositivi.
  - Questo passaggio aggiuntivo serve a ottimizzare la comunicazione tra unità che lavorano a velocità differenti.

Prinicipi di grafica

- Cosa vuol dire scrivere di volta in volta la matrice da stampare nella finestra ?
  - Per questo viene introdotto un intermediaro: il **buffer**.
  - Si tratta di una memoria temporanea in cui si memorizzano dati che verranno successivamente trasmessi ad altri dispositivi.
  - Questo passaggio aggiuntivo serve a ottimizzare la comunicazione tra unità che lavorano a velocità differenti.
  - Nel caso in esame si parla di **buffer video**.

Prinicipi di grafica

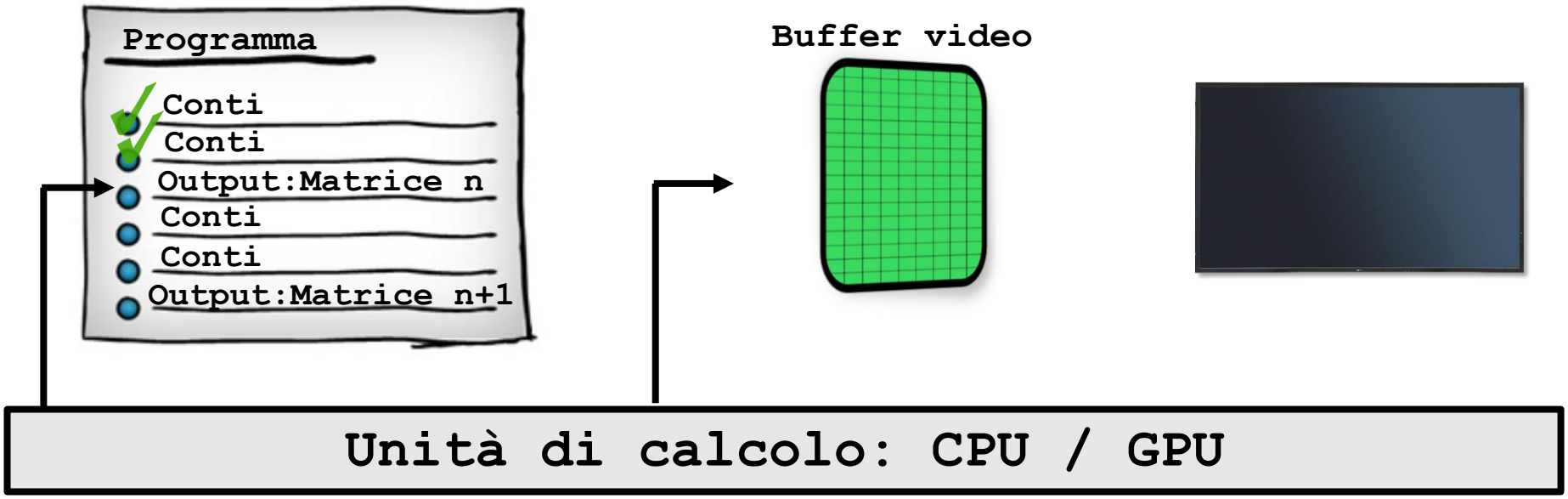
- Cosa vuol dire scrivere di volta in volta la matrice da stampare nella finestra ?



- La CPU/GPU dopo aver calcolato la matrice n-esima la scrive su un buffer video accessibile allo schermo.

Prinicipi di grafica

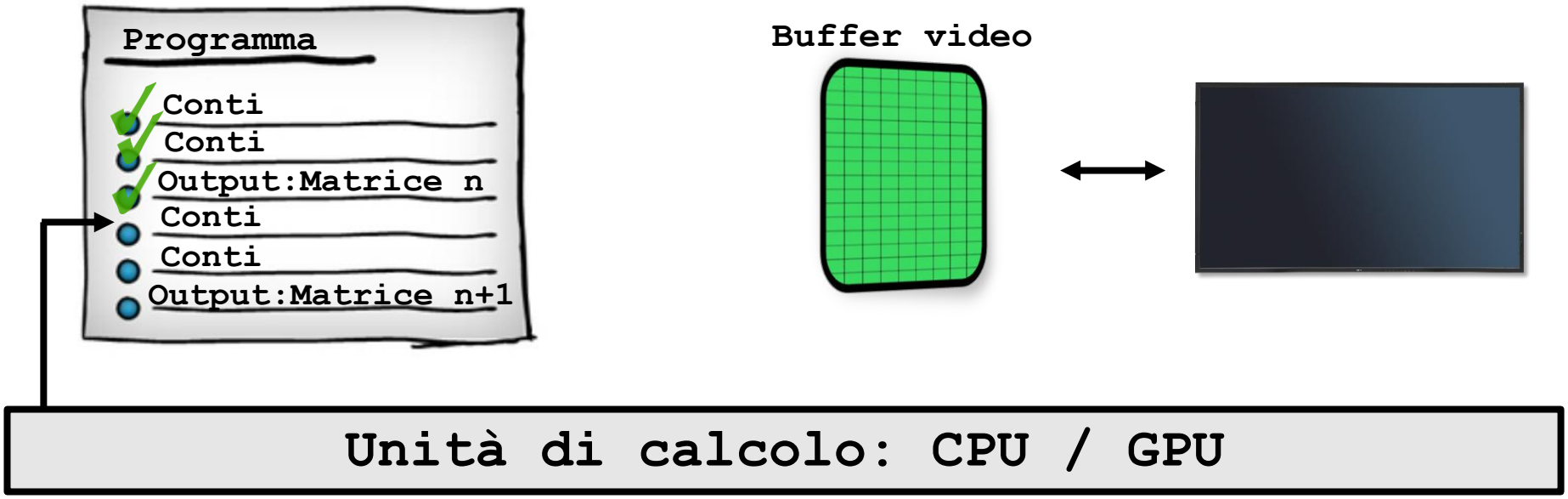
- Cosa vuol dire scrivere di volta in volta la matrice da stampare nella finestra ?



- La CPU/GPU dopo aver calcolato la matrice n-esima la scrive su un buffer video accessibile allo schermo.

Prinicipi di grafica

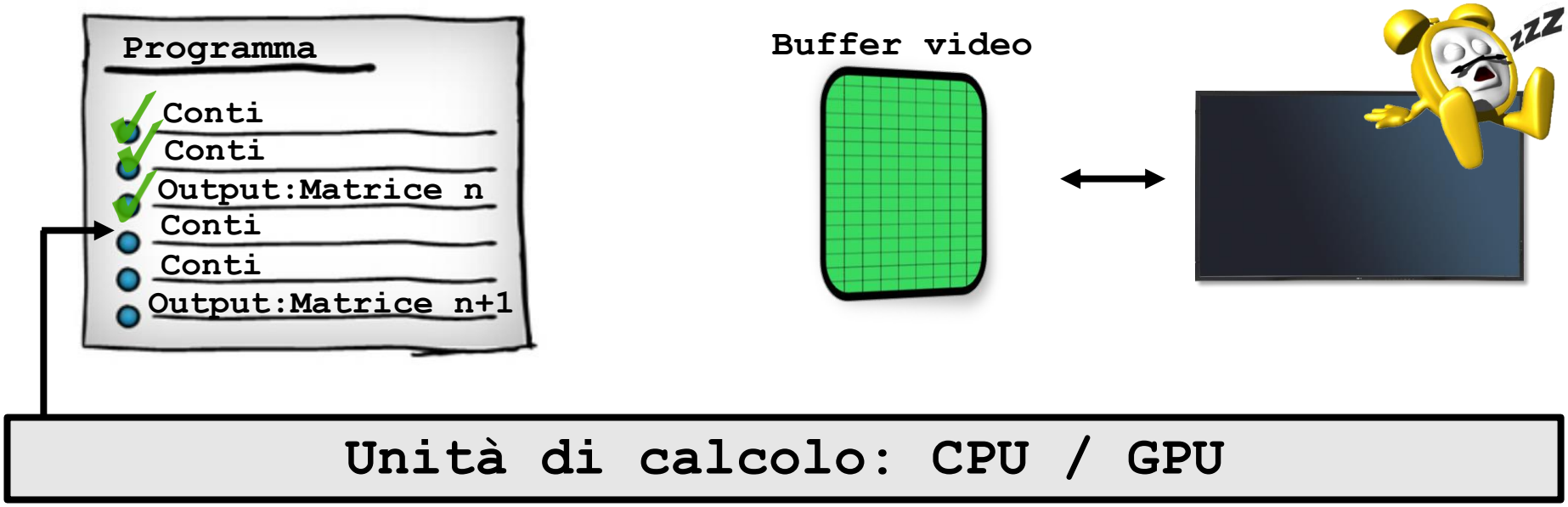
- Cosa vuol dire scrivere di volta in volta la matrice da stampare nella finestra ?



  - Subito dopo la trascrizione, la CPU/GPU torna alla simulazione, così non deve aspettare lo schermo.

Prinicipi di grafica

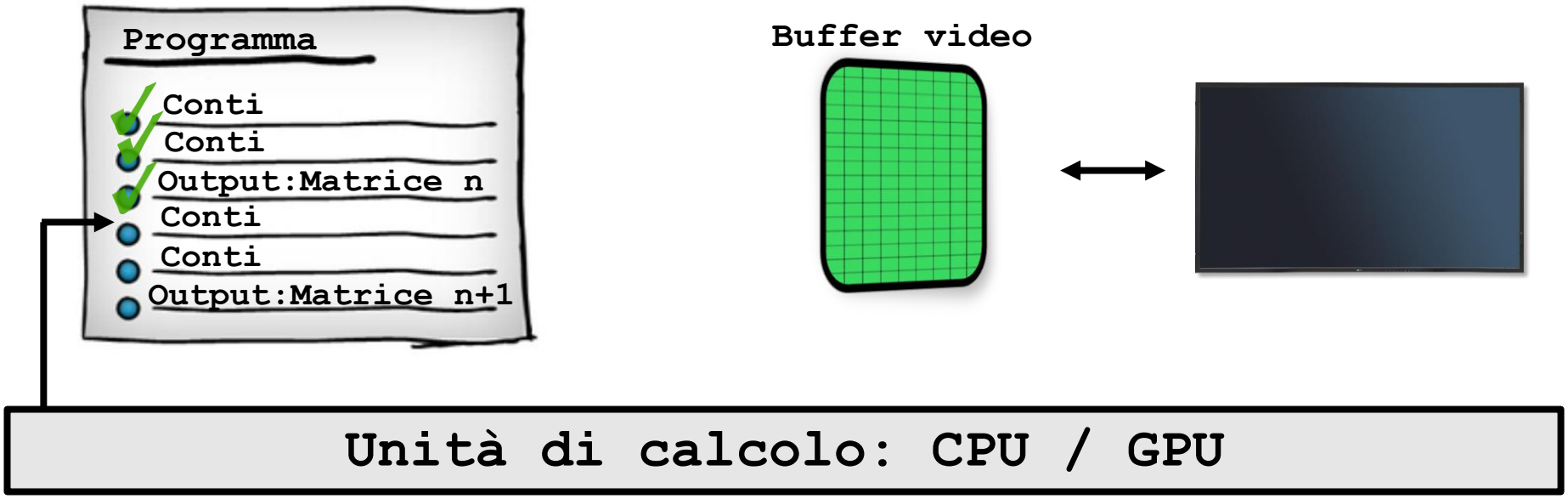
- Cosa vuol dire scrivere di volta in volta la matrice da stampare nella finestra ?



  - Lo schermo quindi non comunica con l'unità di calcolo e legge i dati da stampare su video dal buffer.

Prinicipi di grafica

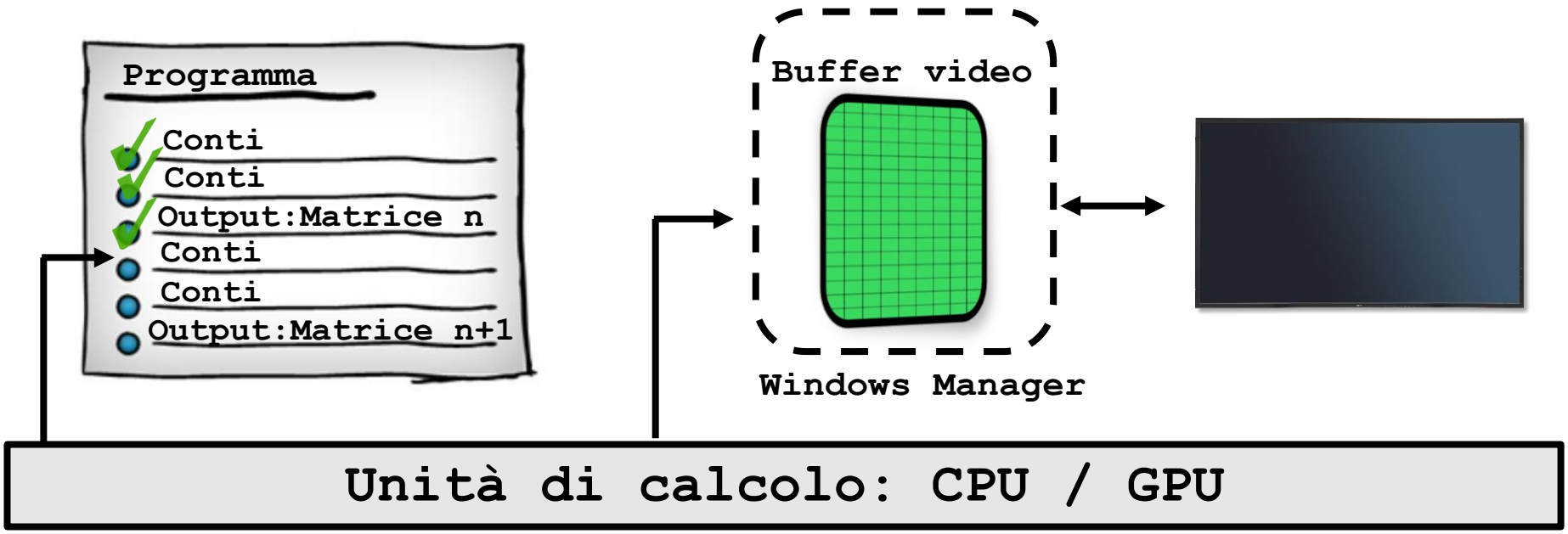
- Cosa vuol dire scrivere di volta in volta la matrice da stampare nella finestra ?



- L'accesso al buffer di una finestra video è gestita dal **Windows Manager** del sistema operativo del computer.

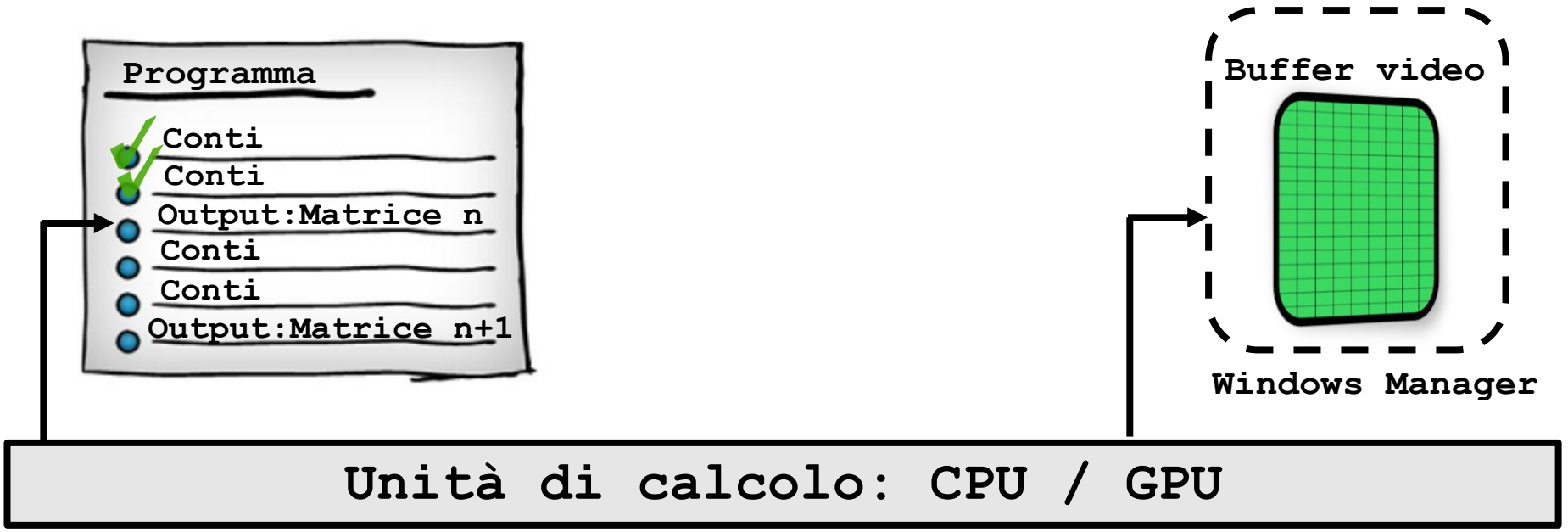
Prinicipi di grafica

- Cosa vuol dire scrivere di volta in volta la matrice da stampare nella finestra ?



- L'accesso al buffer di una finestra video è gestita dal **Windows Manager** del sistema operativo del computer.

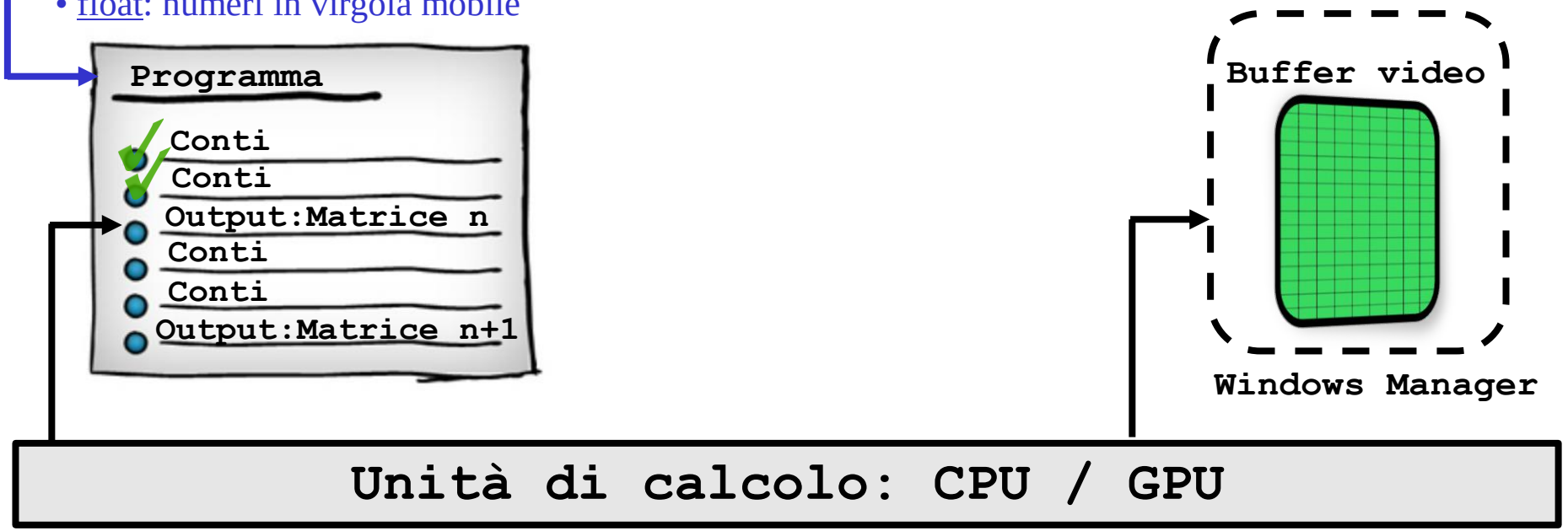
Prinicipi di grafica



Prinicipi di grafica

Programma scritto in C, un liguaggio che ha 5 type:

- char: caratteri
- int: numeri interi
- float: numeri in virgola mobile
- double: numeri in virgola mobile doppi
- void: non valori

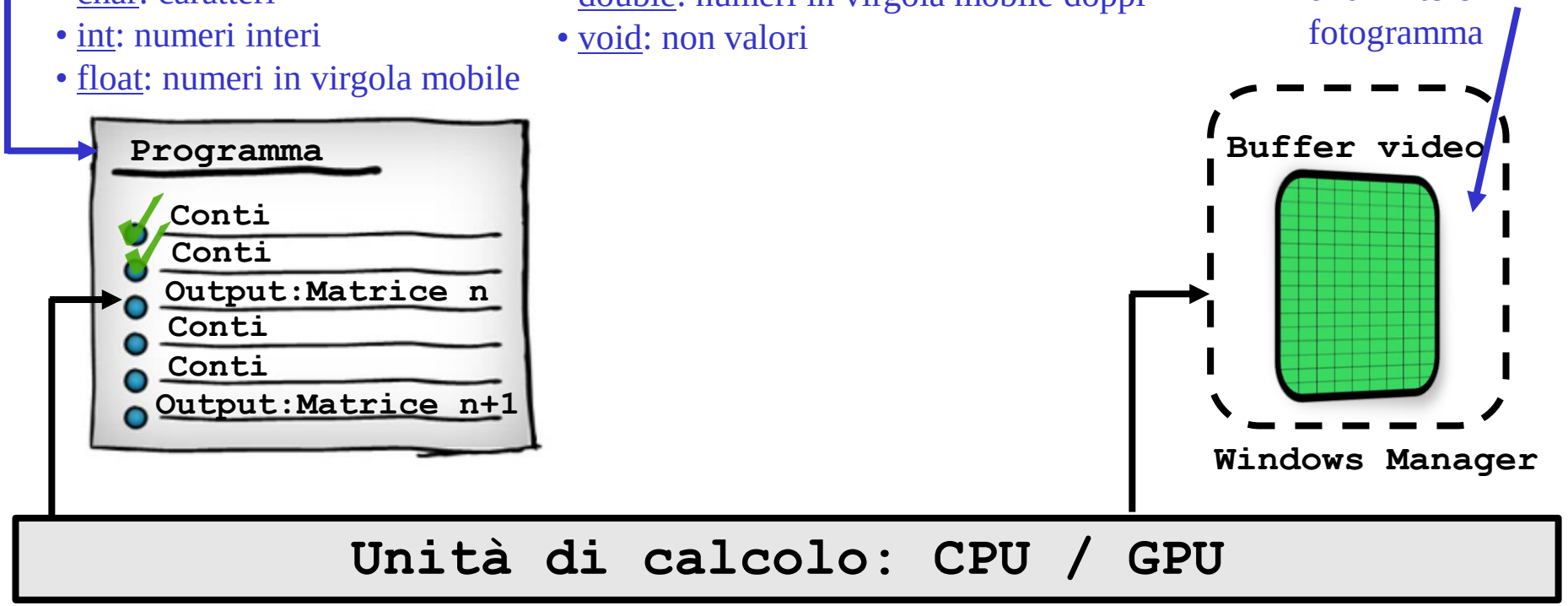


Prinicipi di grafica

Programma scritto in C, un liguaggio che ha 5 type:

- char: caratteri
- int: numeri interi
- float: numeri in virgola mobile
- double: numeri in virgola mobile doppi
- void: non valori

Informazioni destinate alla rappresentazione di un intero fotogramma

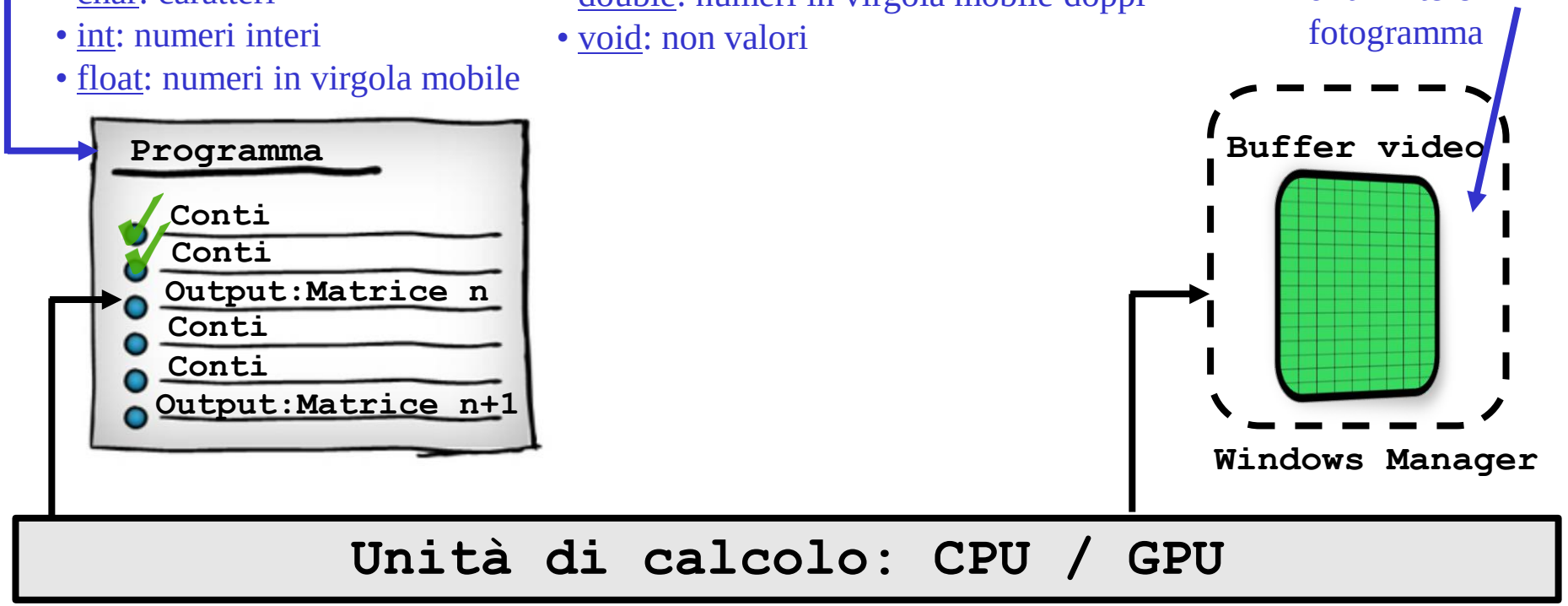


Prinicipi di grafica

Programma scritto in C, un liguaggio che ha 5 type:

- char: caratteri
- int: numeri interi
- float: numeri in virgola mobile
- double: numeri in virgola mobile doppi
- void: non valori

Informazioni destinate alla rappresentazione di un intero fotogramma

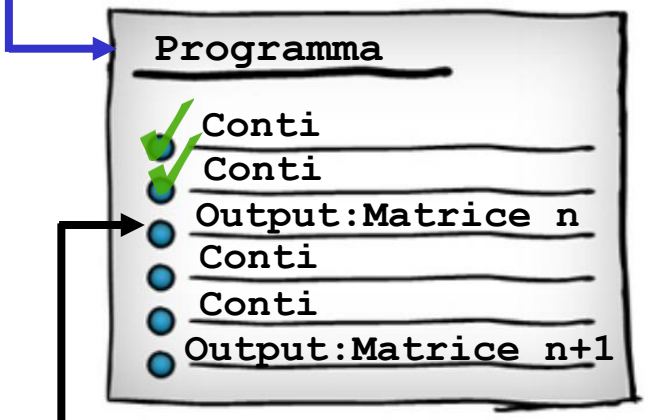


- Come creare immagini articolate in C con un codice semplice e leggibile?

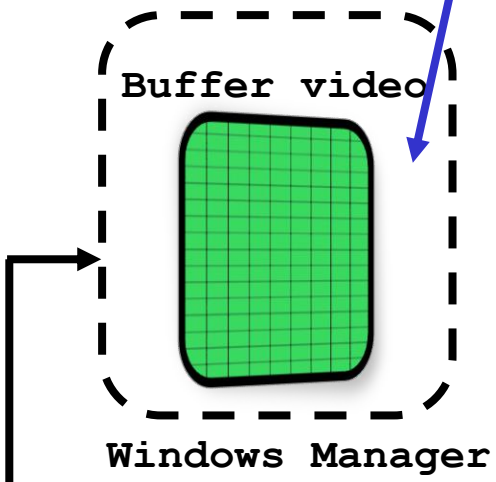
Prinicipi di grafica

Programma scritto in C, un liguaggio che ha 5 type:

- char: caratteri
- int: numeri interi
- float: numeri in virgola mobile
- double: numeri in virgola mobile doppi
- void: non valori



Informazioni destinate alla rappresentazione di un intero fotogramma



Unità di calcolo: CPU / GPU

- Come creare immagini articolate in C con un codice semplice e leggibile?
  - Tramite la libreria **OpenGL**.

OpenGL

- OpenGL è una libreria per lo sviluppo di applicazioni grafiche interattive 2D e 3D.
  - Utilizzabile con diversi linguaggi (C, C++, Python, Java...)
  - Documentazione:

    - Red Book:

    `www.opengl.org/documentation/red_book_1.0`

    - Blue Book:

    `www.rush3d.com/reference/opengl-bluebook-1.0`

  - Non dipende dalla piattaforma hardware/software

OpenGL

- OpenGL è una libreria per lo sviluppo di applicazioni grafiche interattive 2D e 3D.
  - Utilizzabile con diversi linguaggi (C, C++, Python, Java...)
  - Documentazione:

    - Red Book:

    `www.opengl.org/documentation/red_book_1.0`

    - Blue Book:

    `www.rush3d.com/reference/opengl-bluebook-1.0`

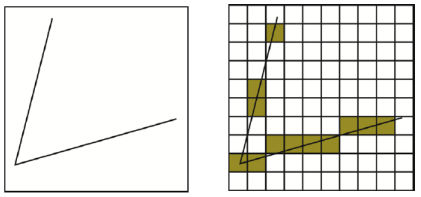
  - Non dipende dalla piattaforma hardware/software

Come è possibile visto che deve accedere al Buffer video, che è gestito dal Windows manager del sistema?

OpenGL

- OpenGL è multipiattaforma, perchè:
  - Non contiene comandi per la creazione/gestione di finestre
  - Nè per la gestione dell'input da tastiera o mouse

- Allora a cosa servono le funzioni della libreria OpenGL?
  - Costruire forme 2D e 3D anche molto complesse a partire da primitive geometriche semplici come punti, linee e poligoni.
  - Disporre oggetti nel piano 2D o nello spazio 3D
  - Calcolare il colore degli oggetti come risultato dell'effetto di condizioni di illuminazione, dell'applicazione di texture o dell'assegnazione esplicita da parte del programmatore
  - Rasterizzazione, cioè conversione della rappresentazione matematica degli oggetti in pixel



OpenGL

- Come è possibile allora usare le funzioni di OpenGL per costruire scene e poi rappresentarle su schermo?

OpenGL

- Come è possibile allora usare le funzioni di OpenGL per costruire scene e poi rappresentarle su schermo?

  - Utilizzando delle librerie esterne al nocciolo principale di OpenGL che permettono di semplificare la gestione dell'interfaccia grafica (GLU, librerie per le GUI, GLUT).

OpenGL

- Come è possibile allora usare le funzioni di OpenGL per costruire scene e poi rappresentarle su schermo?

  - Utilizzando delle librerie esterne al nocciolo principale di OpenGL che permettono di semplificare la gestione dell'interfaccia grafica (GLU, librerie per le GUI, GLUT).

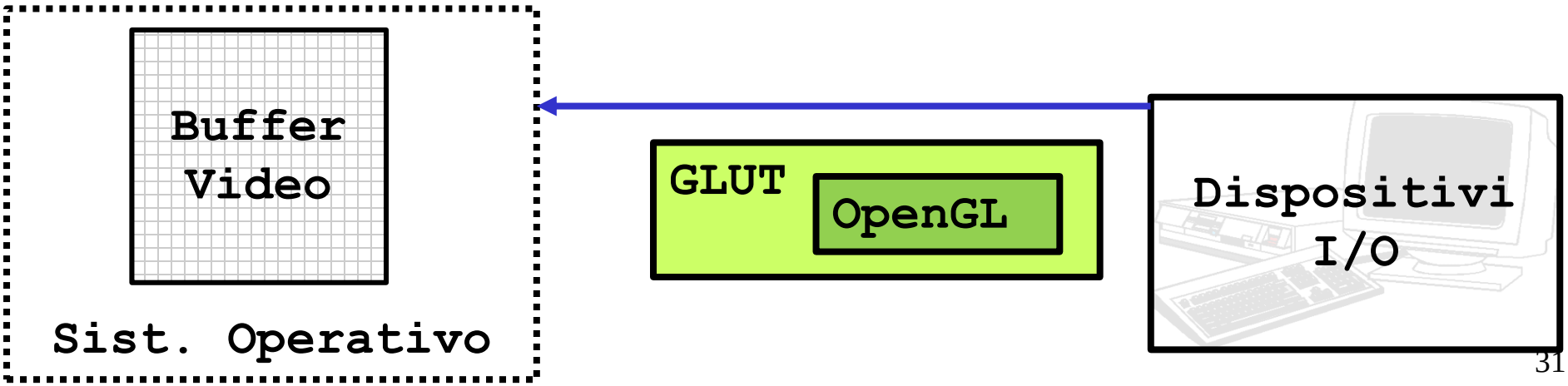
GLUT (OpenGL Utility Toolkit)

- GLUT è una API (Application Programming Interface), una libreria di funzioni che fungono da interfaccia tra l'utente e OpenGL, permettondone un utilizzo semplificato.

- Cosa fa GLUT?
  - Fornisce una interfaccia unica e personalizzabile per comunicare con i Windows Manager dei diversi sistemi operativi.
  - Permette di gestire in modo personalizzato gli eventi (della finestra, del timer, dei dispositivi di input)
  - Fornisce funzioni di chiamata alle funzioni OpenGL personalizzabili dal programmatore.

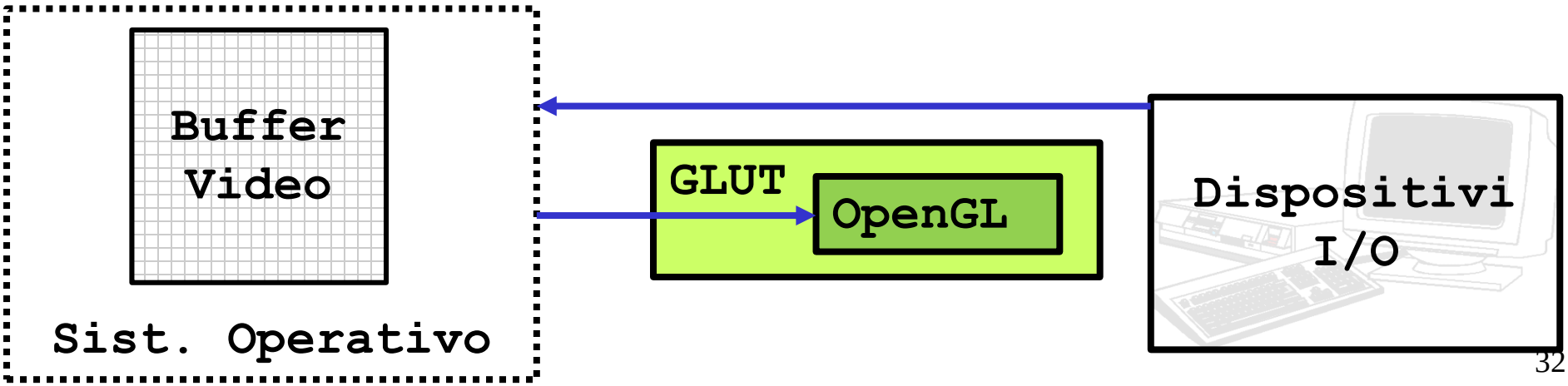
Esempio di funzionamento di GLUT

- Ogni volta che succede qualcosa ai dispositivi di input avviene un evento (l'utente preme un tasto, muove il mouse...).



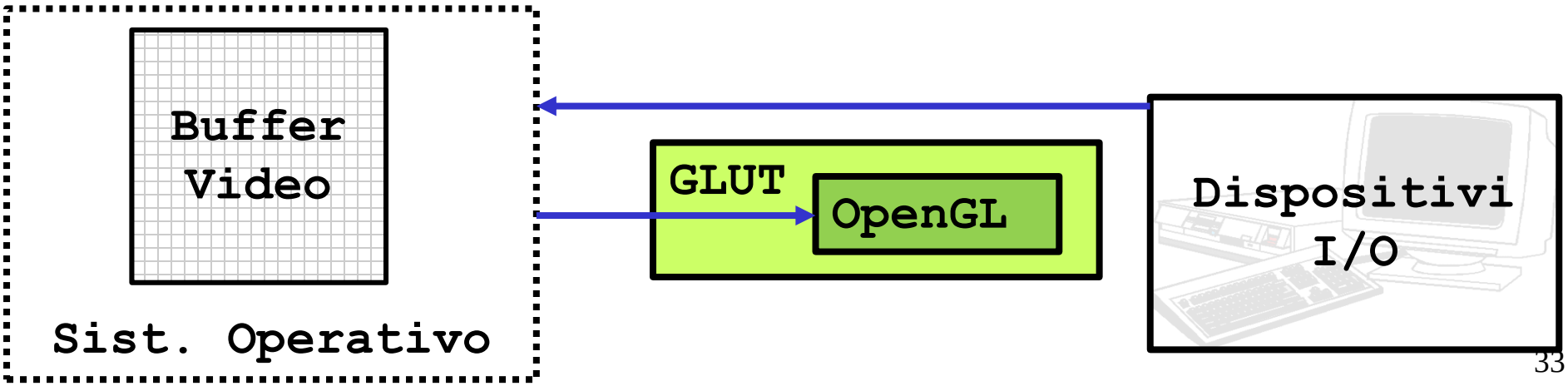
Esempio di funzionamento di GLUT

- Ogni volta che succede qualcosa ai dispositivi di input avviene un evento (l'utente preme un tasto, muove il mouse...).
- Il S.O. invia un messaggio all'applicazione chiedendo come interpretare quell'evento.



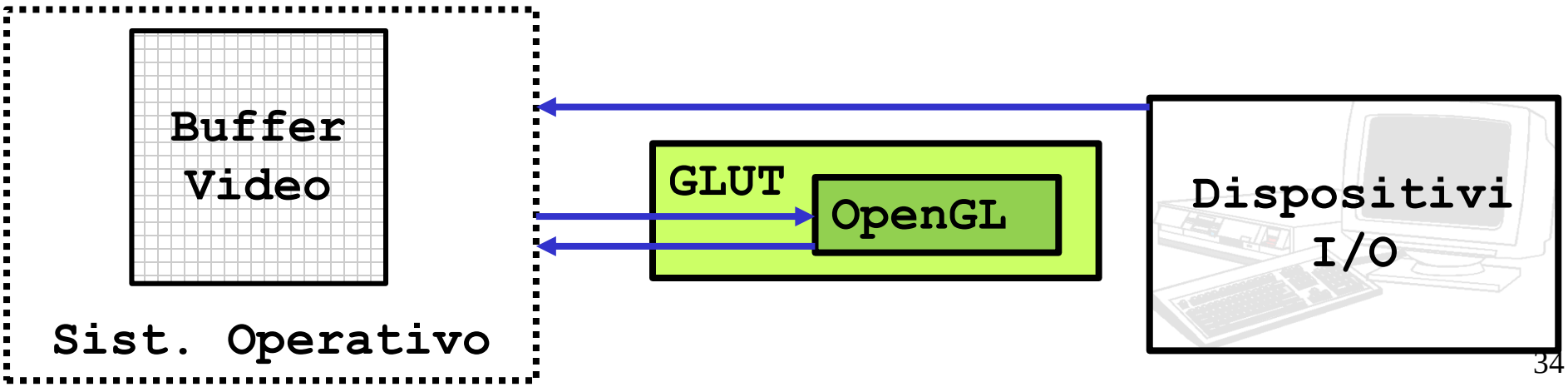
Esempio di funzionamento di GLUT

- Ogni volta che succede qualcosa ai dispositivi di input avviene un evento (l'utente preme un tasto, muove il mouse...).
- Il S.O. invia un messaggio all'applicazione chiedendo come interpretare quell'evento.
- GLUT intercetta il messaggio e, sulla base delle funzioni personalizzate dall'utente, chiama le opportune funzioni di OpenGL.



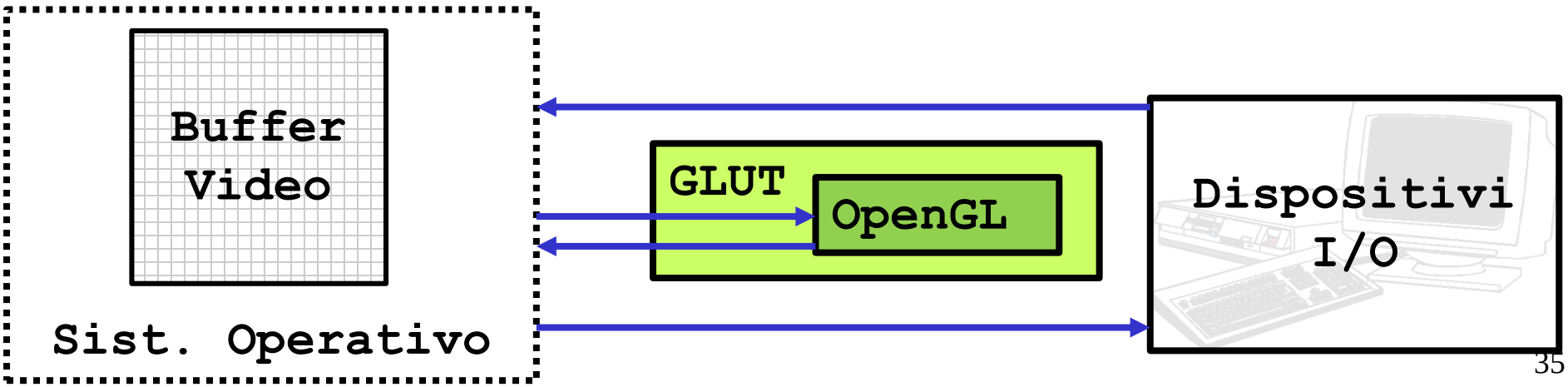
Esempio di funzionamento di GLUT

- Ogni volta che succede qualcosa ai dispositivi di input avviene un evento (l'utente preme un tasto, muove il mouse...).
- Il S.O. invia un messaggio all'applicazione chiedendo come interpretare quell'evento.
- GLUT intercetta il messaggio e, sulla base delle funzioni personalizzate dall'utente, chiama le opportune funzioni di OpenGL.
- GLUT, che sa comunicare con diversi S.O., istruisce il Windows Manager su cosa mettere nel Buffer Video.



Esempio di funzionamento di GLUT

- Ogni volta che succede qualcosa ai dispositivi di input avviene un evento (l'utente preme un tasto, muove il mouse...).
- Il S.O. invia un messaggio all'applicazione chiedendo come interpretare quell'evento.
- GLUT intercetta il messaggio e, sulla base delle funzioni personalizzate dall'utente, chiama le opportune funzioni di OpenGL.
- GLUT, che sa comunicare con diversi S.O., istruisce il Windows Manager su cosa mettere nel Buffer Video.
- S.O. infine gestisce la rappresentazione sui dispositivi di output.



Animazione: moto rettilineo uniforme

- Installare GLUT (che include anche OpenGL):

```
$ sudo apt-get install freeglut3-dev freeglut3
```

- Includere il file header della libreria:

```
#include <GL/glut.h>
```

- In fase di compilazione bisogna eseguire il link del programma con la libreria:

```
gcc -Wall -pedantic es.c -o es -lGL -lm -lglut
```

Animazione: moto rettilineo uniforme

- Le funzioni di Opengl sono tipo: **`glSomething(...)`** mentre quelle di GLUT sono del tipo: **`glutSomething(...)`**
- Per il loro utilizzo è fondamentale conoscere la sintassi.
- Questo è ancor più importante quando si usano delle funzioni GLUT dette di «Callback», ossia quelle funzioni che permettono all'utente di espandere e personalizzare alcune funzionalità della libreria.
- Per avere informazioni dettagliate sulla sintassi delle funzioni GLUT:
  - http://www.xmission.com/~nate/glut.html
  - http://www.opengl.org/resources/libraries/glut/spec3/spec3.html
  - http://www.lighthouse3d.com/opengl/glut/
- E le slides (in italiano) disponibili ai siti:
  - https://www.mat.unical.it/~donato/teaching/2011-2012/ig/ig-2011-2012-v2.pdf
  - http://www.dia.uniroma3.it/~scorzell/didattica2008/ppt/04_opengl.pdf

Moto rettilineo uniforme di una pallina (1/5)

```
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>
```

Librerie

```
#define PI 3.141592653
```

Macro

```
double radius = 0.03;
int frameNumber = 0;
int animating = 0;
double posx;
double posx0;
int dt = 30;
int nstep = 2000;
```

Variabili Globali

Moto rettilineo uniforme di una pallina (1/5)

```
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>
#define PI 3.141592653

double radius = 0.03;
int frameNumber = 0;
int animating = 0;
double posx;
double posx0;
int dt = 30;
int nstep = 2000;
```

Raggio della pallina che si muove

Moto rettilineo uniforme di una pallina (1/5)

```
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>
#define PI 3.141592653

double radius = 0.03;
int frameNumber = 0;
int animating = 0;
double posx;
double posx0;
int dt = 30;
int nstep = 2000;
```

Variabili che tengono conto dello stato di dell'animazione:

- **frameNumber** viene aggiornato per tenere conto del numero di frame a cui è l'animazione
- **animating** viene settato a 1 o 0 a seconda che l'animazione sia attiva o no.

Moto rettilineo uniforme di una pallina (1/5)

```
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>
#define PI 3.141592653

double radius = 0.03;
int frameNumber = 0;
int animating = 0;
double posx;
double posx0;
int dt = 30;
int nstep = 2000;
```

Indica la posizione della particella lungo l'asse x.
Tale variabile viene aggiornata ad ogni spostamento.

Posizione iniziale della particella lungo l'asse x (y0 = 0).

Moto rettilineo uniforme di una pallina (1/5)

```
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>
#define PI 3.141592653

double radius = 0.03;
int frameNumber = 0;
int animating = 0;
double posx;
double posx0;
int dt = 30;
int nstep = 2000;
```

Intervallo di tempo (ms) trascorso tra la visualizzazione di frame e il consecutivo.

Numero masssimo di frame dell'animazione.

Moto rettilineo uniforme di una pallina (2/5)

```
void key(unsigned char ch, int x, int y);

void display();

void timerFunction(int timerID);

void updateFrame();
```

Prototipi

Moto rettilineo uniforme di una pallina (2/5)

```
void key(unsigned char ch, int x, int y);
```

funzione che descrive come gestire i comandi inseriti da keyboard (in questo caso solo per far partire l'animazione)

```
void display();
```

funzione che descrive i pixel da stampare a video

```
void timerFunction(int timerID);
```

funzione che controlla, frame per frame, a che punto è l'animazione e che decide se bisogna aggiornare la stampa a video o se l'animazione deve terminare.

```
void updateFrame();
```

funzione che simula gli spostamenti della pallina

Moto rettilineo uniforme di una pallina (3/5)

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("OpenGL - Basic Draw");
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    posx0 = -0.9;
    updateFrame();
    glutMainLoop();
    return 0;
}
```

Main

Moto rettilineo uniforme di una pallina (3/5)

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("OpenGL - Basic Draw");
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    posx0 = -0.9;
    updateFrame();
    glutMainLoop();
    return 0;
}
```

Funzioni necessarie a:

- Inizializzare GLUT
- Predisporre i buffer necessari per la stampa a video
- Gestire la finestra di visualizzazione

Istruzioni dell'animazione:

- Registrazione delle funzioni definite dal programmatore c le Callback di GLUT
- Definizione dei parametri iniziali dell'animazione

Loop principale dell'animazione

Moto rettilineo uniforme di una pallina (3/5)

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("OpenGL - Basic Draw");
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    posx0 = -0.9;
    updateFrame();
    glutMainLoop();
    return 0;
}
```

`glutInit(&argc, argv)`

- Inizializza la libreria GLUT
- È fondamentale che questa funzione venga chiamata all'inizio del main
- Richiede al S.O. le risorse per aprire una finestra su cui disegnare.
- L'argomento è dato dagli argomenti del main e può essere usato per passare delle flag al programma in fase di esecuzione (ma noi non ne faremo mai uso)

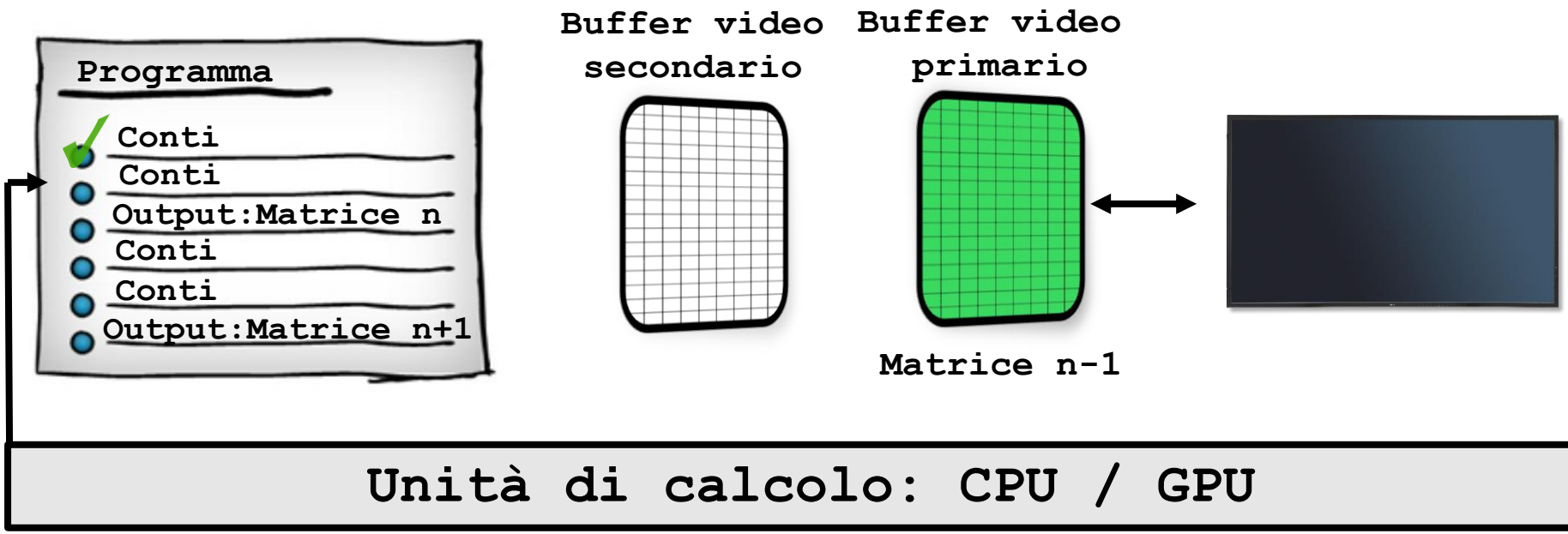
Moto rettilineo uniforme di una pallina (3/5)

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("OpenGL - Basic Draw");
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    posx0 = -0.9;
    updateFrame();
    glutMainLoop();
    return 0;
}
```

`glutDisplayMode(unsigned int mode)`

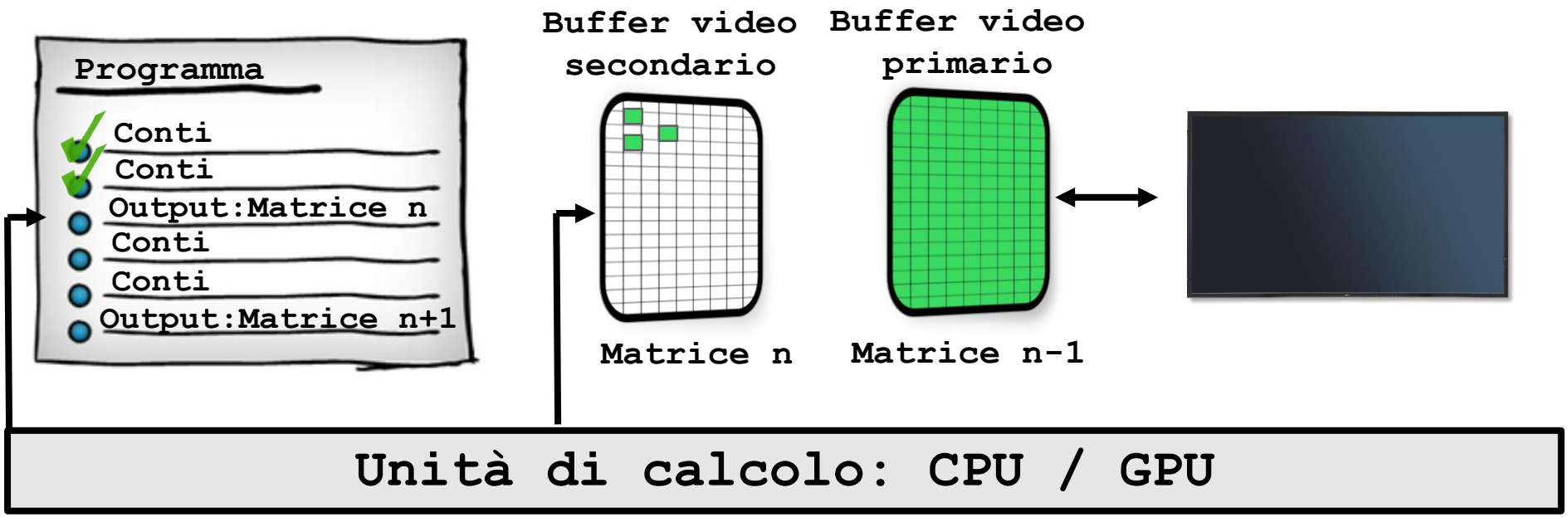
- Setta le caratteristiche della finestra
- Ci sono tantissime opzioni settabili (colori, immagini 3D...)
- Noi usiamo solo GLUT_DOUBLE
- GLUT_DOUBLE: inizializza una «double buffered window», che serve per ottenere animazioni più fluide.

GLUT_DOUBLE



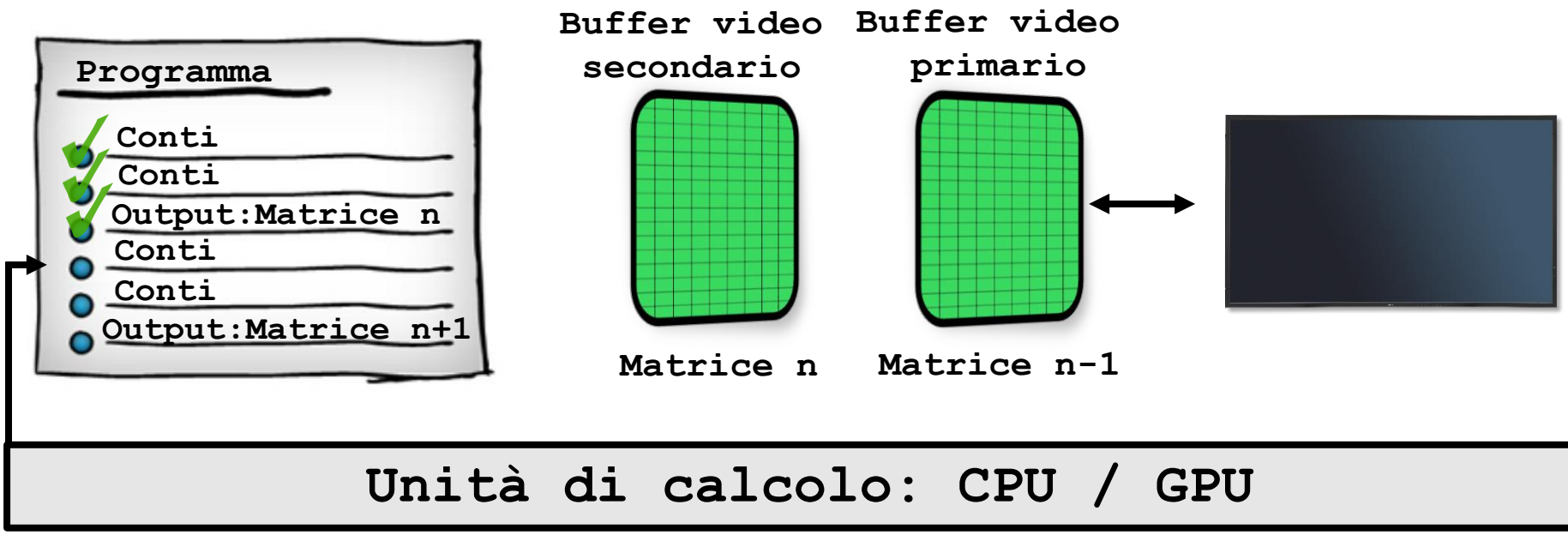
- Lo schermo legge i dati da stampare della matrice n-1 dal Buffer Primario, mentre la CPU/GPU calcola la matrice n.

GLUT_DOUBLE



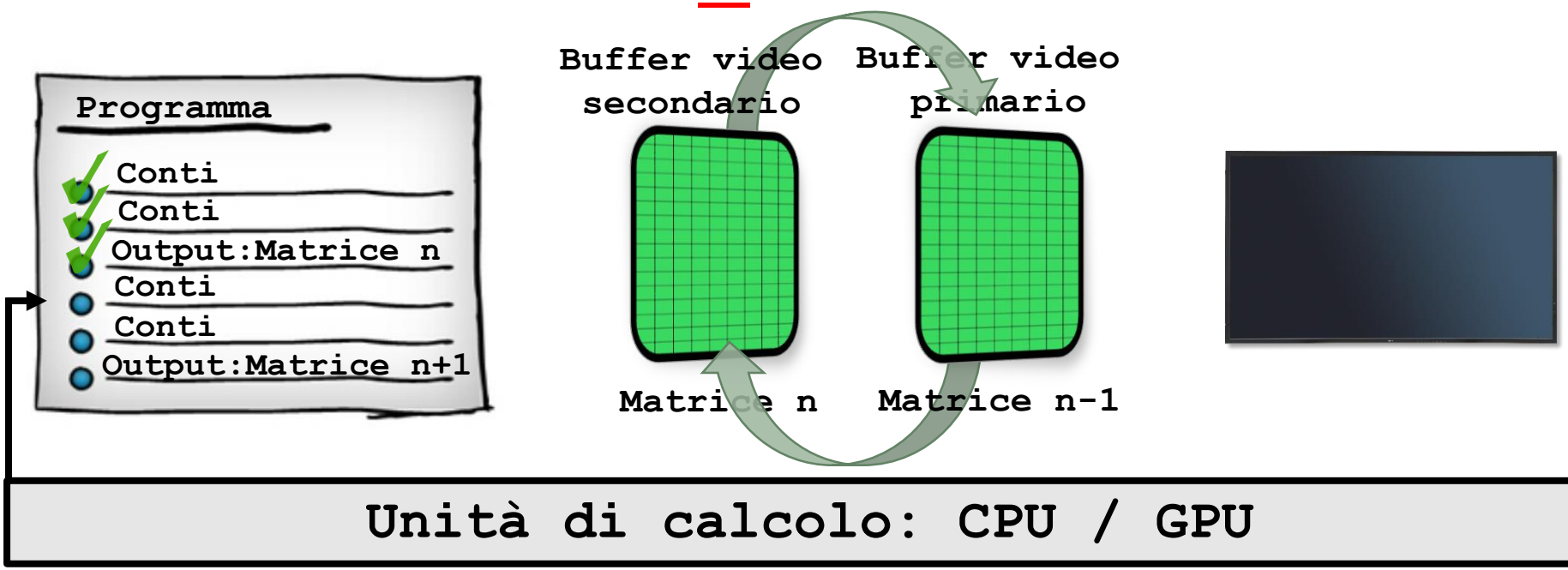
- Lo schermo legge i dati da stampare della matrice n-1 dal Buffer Primario, mentre la CPU/GPU calcola la matrice n.
- Quando la CPU ha finito, scrive la matrice n su un Buffer secondario.

GLUT_DOUBLE



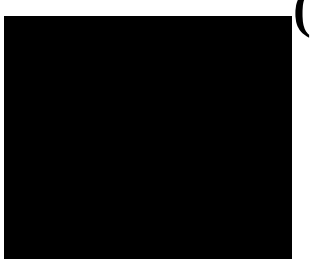
- Lo schermo legge i dati da stampare della matrice n-1 dal Buffer Primario, mentre la CPU/GPU calcola la matrice n.
- Quando la CPU ha finito, scrive la matrice n su un Buffer secondario.
- Quando è il momento di visualizzare sullo schermo un nuovo frame, Buffer secondario e Primario si scambiano.

GLUT_DOUBLE



- Lo schermo legge i dati da stampare della matrice n-1 dal Buffer Primario, mentre la CPU/GPU calcola la matrice n.
- Quando la CPU ha finito, scrive la matrice n su un Buffer secondario.
- Quando è il momento di visualizzare sullo schermo un nuovo frame, Buffer secondario e Primario si scambiano.

Moto rettilineo uniforme di una pallina (3/5)




```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("OpenGL - Basic Draw");
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    posx0 = -0.9;
    updateFrame();
    glutMainLoop();
    return 0;
}
```

- **`glutInitWindowSize( int h, int w)`:** fissa le dimensioni h*w della finestra
- **`glutInitWindowPosition(int x,int y)`:** fissa la posizione del vertice superiore sinistro della finestra a (x,y)
- **`glutCreateWindow(char *name)`:** crea la finestra e pone il titolo pari a «name»

Moto rettilineo uniforme di una pallina (3/5)

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("OpenGL - Basic Draw");
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    posx0 = -0.9;
    updateFrame();
    glutMainLoop();
    return 0;
}
```

- **`glutDisplayFunc(void (*func)(void))`:**
  registra la funzione func designata dall'utente,
  in modo che venga richiamata da GLUT ogni volta
  che bisogna ridisegnare la finestra.
- **`glutKeyboardFunc(void (*func)(unsigned char ch, int x, int y))`:**
  registra la funzione che gestisce la tastiera.
  La variabile ch contiene il carattere ASCII digitato mentre
  x e y indicano la posizione del mouse (che non useremo)

Moto rettilineo uniforme di una pallina (3/5)

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("OpenGL - Basic Draw");
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    posx0 = -0.9;
    updateFrame();
    glutMainLoop();
    return 0;
}
```

Setting della posizione iniziale della pallina

Moto rettilineo uniforme di una pallina (3/5)

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("OpenGL - Basic Draw");
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    posx0 = -0.9;
    updateFrame();
    glutMainLoop();
    return 0;
}
```

Chiamata ad una funzione creata dall'utente (infatti non comincia nè con gl nè con glut), che calcola la posizione della pallina.

Moto rettilineo uniforme di una pallina (3/5)

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("OpenGL - Basic Draw");
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    posx0 = -0.9;
    updateFrame();
    glutMainLoop();
    return 0;
}
```

Funzione di GLUT che inizia il processo d'ascolto degli eventi e attiva le callback functions.

Moto rettilineo uniforme di una pallina (4/5)

```
void key(unsigned char ch, int x, int y){
    if ( ch == ' ' && animating==0 ) {
        animating = 1;
        glutTimerFunc(dt, timerFunction, 0); }
}
```

```
void display() {
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POLYGON);
   for(double i = 0; i < 2 * PI; i += PI / 20) {
    glVertex3f(cos(i)*radius + posx, sin(i)*radius, 0.0);}
   glEnd();
   glutSwapBuffers();
}
```

Moto rettilineo uniforme di una pallina (4/5)

```
void key(unsigned char ch, int x, int y){
    if ( ch == ' ' && animating==0 ) {
        animating = 1;
        glutTimerFunc(dt, timerFunction, 0); }
}
```

Il type e gli argomenti della funzione sono imposti dalla funzione **glutKeyboardFunc**. Noi però usiamo solo char.

```
void display() {
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POLYGON);
   for(double i = 0; i < 2 * PI; i += PI / 20) {
    glVertex3f(cos(i)*radius + posx, sin(i)*radius, 0.0);}
   glEnd();
   glutSwapBuffers();
}
```

Moto rettilineo uniforme di una pallina (4/5)

```
void key(unsigned char ch, int x, int y){
    if ( ch == ' ' && animating==0 ) {
        animating = 1;
        glutTimerFunc(dt, timerFunction, 0); }
}
```

Se viene digitato uno spazio e l'animazione non è ancora iniziata...

```
void display() {
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POLYGON);
   for(double i = 0; i < 2 * PI; i += PI / 20) {
    glVertex3f(cos(i)*radius + posx, sin(i)*radius, 0.0);}
   glEnd();
   glutSwapBuffers();
}
```

Moto rettilineo uniforme di una pallina (4/5)

```
void key(unsigned char ch, int x, int y){
    if ( ch == ' ' && animating==0 ) {
        animating = 1;
        glutTimerFunc(dt, timerFunction, 0); }
}
```

... Setta animating = 1 e ...

```
void display() {
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POLYGON);
   for(double i = 0; i < 2 * PI; i += PI / 20) {
    glVertex3f(cos(i)*radius + posx, sin(i)*radius, 0.0);}
   glEnd();
   glutSwapBuffers();
}
```

Moto rettilineo uniforme di una pallina (4/5)

```
void key(unsigned char ch, int x, int y){
    if ( ch == ' ' && animating==0 ) {
        animating = 1;
        glutTimerFunc(dt, timerFunction, 0); }
}
```

glutTimerFunc(unsigned int ms, void (*func)(int value), value):
registra func come funzione da richiamare dopo un tempo (minimo) ms e le passa value.

```
void display() {
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POLYGON);
   for(double i = 0; i < 2 * PI; i += PI / 20) {
    glVertex3f(cos(i)*radius + posx, sin(i)*radius, 0.0);}
   glEnd();
   glutSwapBuffers();
}
```

Moto rettilineo uniforme di una pallina (4/5)

```
void key(unsigned char ch, int x, int y){
    if ( ch == ' ' && animating==0 ) {
        animating = 1;
        glutTimerFunc(dt, timerFunction, 0); }
}
```

Funzioni di OpenGL (cominciano con gl)

```
void display() {
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POLYGON);
   for(double i = 0; i < 2 * PI; i += PI / 20) {
    glVertex3f(cos(i)*radius + posx, sin(i)*radius, 0.0);}
   glEnd();
   glutSwapBuffers();
}
```

Moto rettilineo uniforme di una pallina (4/5)

```
void key(unsigned char ch, int x, int y){
    if ( ch == ' ' && animating==0 ) {
        animating = 1;
        glutTimerFunc(dt, timerFunction, 0); }
}
```

`glClearColor(GLfloat red,Glfloat green,Glfloat blue,Glfloat alpha)`

Setta i colori con cui la funzione `glClear` riempe la finestra quando questa viene chiamata

```
void display() {
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POLYGON);
   for(double i = 0; i < 2 * PI; i += PI / 20) {
    glVertex3f(cos(i)*radius + posx, sin(i)*radius, 0.0);}
   glEnd();
   glutSwapBuffers();
}
```

Moto rettilineo uniforme di una pallina (4/5)

```
void key(unsigned char ch, int x, int y){
    if ( ch == ' ' && animating==0 ) {
        animating = 1;
        glutTimerFunc(dt, timerFunction, 0); }
}
```

glClear(GLbitfield mask| GLbitfield mask|…):

Cancella le maschere elencate in input, secondo quanto impostato con **glClearColor.**

```
void display() {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
    for(double i = 0; i < 2 * PI; i += PI / 20) {
     glVertex3f(cos(i)*radius + posx, sin(i)*radius, 0.0);}
    glEnd();
    glutSwapBuffers();
}
```

Moto rettilineo uniforme di una pallina (4/5)

```
void key(unsigned char ch, int x, int y){
    if ( ch == ' ' && animating==0 ) {
        animating = 1;
        glutTimerFunc(dt, timerFunction, 0); }
}
```

Funzioni di OpenGL per disegnare un poligono (in questo caso un poligono a 20 lati in (posx, 0))

```
void display() {
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POLYGON);
   for(double i = 0; i < 2 * PI; i += PI / 20) {
    glVertex3f(cos(i)*radius + posx, sin(i)*radius, 0.0);}
   glEnd();
   glutSwapBuffers();
}
```

Moto rettilineo uniforme di una pallina (4/5)

```
void key(unsigned char ch, int x, int y){
    if ( ch == ' ' && animating==0 ) {
        animating = 1;
        glutTimerFunc(dt, timerFunction, 0); }
}

void display() {
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POLYGON);
   for(double i = 0; i < 2 * PI; i += PI / 20) {
    glVertex3f(cos(i)*radius + posx, sin(i)*radius, 0.0);}
   glEnd();
   glutSwapBuffers();
}
```

Funzione GLUT che scambia il buffer primario e quello secondario

Moto rettilineo uniforme di una pallina (5/5)

```
void timerFunction(int timerID) {
    if (frameNumber>nstep) {
    animating=0;
    return;}
    if (posx>0.9||posx<-0.9) {
      animating=0;
      return;}
    if (animating == 1) {
        updateFrame();
        glutTimerFunc(dt, timerFunction, 0);
        glutPostRedisplay();}
}
```

```
void updateFrame() {
   posx = frameNumber*dt/10000.0 + posx0;
   frameNumber++;}
```

Moto rettilineo uniforme di una pallina (5/5)

```
void timerFunction(int timerID) {
    if (frameNumber>nstep) {
    animating=0;
    return;}
    if (posx>0.9||posx<-0.9) {
      animating=0;
      return;}
    if (animating == 1) {
        updateFrame();
        glutTimerFunc(dt, timerFunction, 0);
        glutPostRedisplay();}
}

void updateFrame() {
   posx = frameNumber*dt/10000.0 + posx0;
   frameNumber++;}
```

Se si supera il limite massimo di frame, ferma la simulazione

Moto rettilineo uniforme di una pallina (5/5)

```
void timerFunction(int timerID) {
    if (frameNumber>nstep) {
    animating=0;
    return;}
    if (posx>0.9||posx<-0.9) {
      animating=0;
      return;}
    if (animating == 1) {
        updateFrame();
        glutTimerFunc(dt, timerFunction, 0);
        glutPostRedisplay();}
}

void updateFrame() {
   posx = frameNumber*dt/10000.0 + posx0;
   frameNumber++;}
```

Se la pallina raggiunge uno dei due limiti lungo x, la simulazione termina

Moto rettilineo uniforme di una pallina (5/5)

```
void timerFunction(int timerID) {
    if (frameNumber>nstep) {
    animating=0;
    return;}
    if (posx>0.9||posx<-0.9) {
      animating=0;
      return;}
    if (animating == 1) {
       updateFrame();
       glutTimerFunc(dt, timerFunction, 0);
       glutPostRedisplay();}
}
```

Se l'animazione è in corso bisogna mandare avanti la simulazione, risettare il timer che richiamerà questa funzione nel tempo dt e chiamare **`glutPostRedisplay()`**

Marca la finestra come da aggiornare. Quindi al prossimo giro di **`glutMainLoop`**, la finestra viene ridisegnata sulla base della funzione display.

```
void updateFrame() {
   posx = frameNumber*dt/10000.0 + posx0;
   frameNumber++;}
```

Moto rettilineo uniforme di una pallina (5/5)

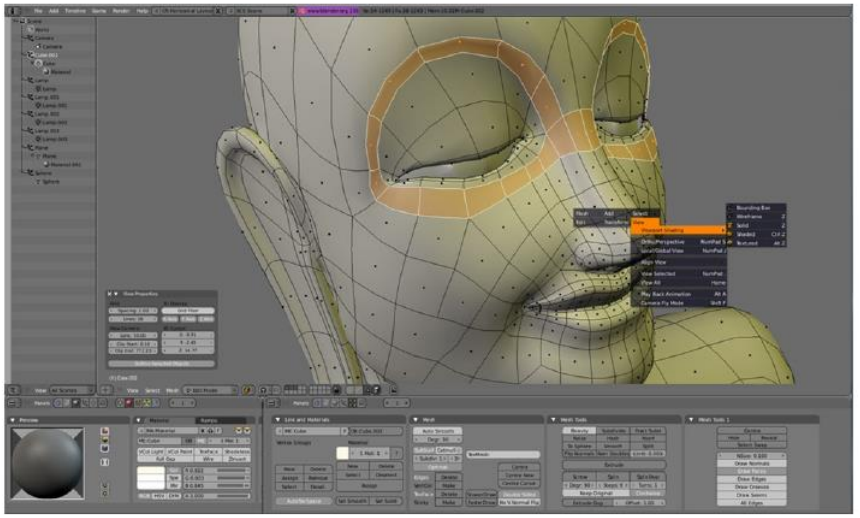
```
void timerFunction(int timerID) {
    if (frameNumber>nstep) {
    animating=0;
    return;}
    if (posx>0.9||posx<-0.9) {
      animating=0;
      return;}
    if (animating == 1) {
        updateFrame();
        glutTimerFunc(dt, timerFunction, 0);
        glutPostRedisplay();}
}
```

Manda avanti la simulazione, cioè calcola la nuova posx, ogni volta che viene chiamata e incrementa il valore di FrameNumber

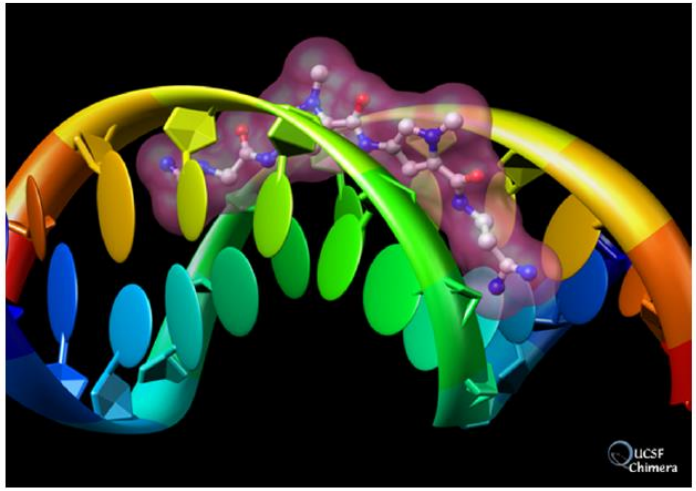
```
void updateFrame() {
   posx = frameNumber*dt/10000.0 + posx0;
   frameNumber++;}
```

Software basati su OpenGL

Unreal(`http://www.unrealtournament.com`)



Blender (`http://www.blender.org`)



Chimera (`www.cgl.ucsf.edu/chimera`)